GROUND CONTROL™ SDK 2.0

This document was designed to aid people interested in modifying parts of Ground Control™. It covers the basic directory structure, file types and how to modify different parts of the GC system.

As there is no official support for this documentation or the correctness of the information contained herein, neither Sierra nor Massive Entertainment may be held responsible for any possible damage resulting from the use of the information in this document or the tools supplied with it. Be thankful for what you got.

However, if any gross errors should be found in this document, please feel free to ask for clarification in the official Ground Control™ forums, which is monitored by knowledgeable amateurs as well as experienced professionals. If that fails, please as a last resort only, mail me at piotrr@swipnet.se - thanks.

Thank you to Tweaks, TomBob, Ninja Prime, Bazuka and all the others for your support and inspiration!

- Per 'piotrr' Edman, August 29, 2001

SECTION 0 – SETTING UP

If you want to modify Ground Control™, it is recommended that you use the SDFextract program to unpack all of Ground Controls’ SDF files, perhaps into a new directory so that you will have what we call a “test bed” directory to play around with Ground Control™ in. You will need up towards a gigabyte (1024 megabytes) of free hard disk space to do this. Follow the steps below.

1. Create a new directory where you want your “experimental” Ground Control™. This directory will act as a “workbench” for the changes you want to do, and will keep you from disturbing your original installation of Ground Control™. I will call this folder “TestBed”.
2. Copy **all** files from your Ground Control™ folder to your TestBed folder.
3. Copy the MOD tools into your TestBed data folder, where the SDF files reside. These programs are SDFextract.exe, SDFman.exe, Mp3ToMpd.exe and PackMOD.exe.
4. Using SDFextract.exe, first unpack the DATA-files. Start with DATA2.SDF, then DATA1.SDF and finally DATA0.SDF before unpacking the other SDF files you are interested in modifying. You do not have to extract all SDF files, but to save space you may delete SDF files that you have already fully extracted.

You can use the provided batch file (GC-MOD-SDK2-Install2.bat) from the GC start directory (where the GC.EXE file is) to unpack all available SDF files, including Dark Conspiracy files.

You are now ready to start modifying Ground Control!

SECTION 1 – DIRECTORY STRUCTURE

This section will help you understand the directory structure of Ground Control™. It is based in the Ground Control™ install path, where you should find **GC.EXE**, **IJL11.DLL** and a few text files. Normally located in C:\Sierra\GC\, or in the base "test bed" directory, under the Data directory, this is the structure and contents of the directories:

In the Data directory, you find GC's SDF files, maybe a **Capabilities.txt** file. If you have installed GenEd, you should also find **GenEd** here. Some read me files could lie around here too. After you install the SDK, you should find a few more programs in this directory, such as **SDFextract, PackMod, MP3toMPD** and **SDFman.**

AttributeModifiers

In this directory you should only find ATM files. This is where GC will look for Attribute modifying effects for special equipment.

Campaigns

Under this directory, every new campaign should have a directory of its own, named the same as the campaign files. For the Crayven campaign, or a replacement, there should be a directory of the name CRAYVEN and the three files necessary for the campaign should be in that directory by the names CRAYVEN.LOC, CRAYVEN.CAM and CRAYVEN.TEC.

Cutscenes

From this directory BIK format cut scenes are read. What cut scenes to read are defined in the relevant CAM campaign file, under Data\Campaigns.

Dialogs

Here are the dialog definitions for all interface screens. Files are called TXT. Somewhat longer descriptions are available in the previous chapter (on file types). Example files are enclosed.

EntFiles

Entity definition files, ENT are read from here by **GC**, **WeaponConf** and **GenEd**.

GenEdData

Contains default mission data (for when you click "new mission") and some variations on the bump map and cloud map. Also contains **Skies.txt**, which details the available skies, and **Props.txt** in which all available props are documented. To add a prop, this is the file to edit.

DefaultData\

Default bump map, cloud map, height map, light map, border for the big map, alpha for the mini-map and border for the mini-map. When creating a new map from scratch, these are supposed to be loaded.

TextureFamilies\

FAM files describing what types of ground textures belong together. These are read from GenEd to allow the mapper to texture the terrain using the textures defined.

Gfx

2D art in TGA format. Icons, among other things, are read from this directory and its substructure. Most items are 32-bit TGA with an eight bit alpha channel.

Font\

The typeset used to type in the game is placed here. Each FONT texture is accompanied by a text file defining the order of characters in the texture. It is recommended that new fonts be built on the existing textures.

InGameInterface\

The art used in the in-game interface, such as mouse pointer, team and unit icons, map markers and so forth are placed in the substructure under this directory.

Medals\

Thirteen medal icons, TGA, listed here.

Mouse\

The mouse pointer.

Progress\

Full screen splash images for different occasions in the game, such as loss and victory.

SpecialEquipmentIcons\

The icons used for special equipment, Crayven and Order.

SpecialWeaponIcons\

The icons used for special weapons, Crayven and Order.

Splash\

The splash screen displayed when loading the game.

SW_MOVIES\

Software mode replacements for the small, animated characters displayed during in-game messages in the missions.

UnitIcons\

All the icons for different types of units in the game.

Localization

All LOC localization string files are contained in this directory. They are accessed from throughout the program for descriptions, names and dialog text. Mission-specific text is not here, but in the mission itself and the same with Campaigns.

ManagementData

Weapons, Equipment, Armor and EntConfigs etc are defined in this directory. The application **WeaponConf** is also in this directory and must be, since it reads from several of the files in the directory. The single-player custom missions and multi-player technology levels are also defined here, in the files **default.tec** and **multiplayer.tec**. See **Entity File Syntax** and **Unit Creation** for reference.

Maps

In this directory, you place missions in the SDF format. Both single- and multi player missions are placed in this directory as the SDF files contain the correct path for the files inside.

Missions

Single-player missions under development are placed and modified in this directory, in subdirectories of the same filenames as the files inside. One example mission has been provided in this directory.

MissionName\

All files relevant to a specific mission (MissionName in this case) are stored in the same directory. GenEd expects it to be so, and expects all files in the directory to be named accordingly.

Models

Lightwave object and motion files are placed in this directory to be called from MDL files (placed in EntModels). Textures are not placed here, but in the separate Texture directory. Subdirectories may exist here, but are not necessary for other purposes than to enhance a sense of structured order.

Movie

CUT show box movie definitions are read from here. The file format is defined in **CUT File Syntax** and the files are used as message box faces and dialog backgrounds.

Briefings\

These show box movies are the ones used from the briefings and when receiving messages in a mission.

Multiplayer

Missions

Multi-player missions under development are placed and modified in this directory, in subdirectories of the same filenames as the files inside. One example mission has been provided in this directory.

PackMOD

LWO objects that have an MDL created for them are compiled into PackMOD *.MOD format files and placed in the directory from which GC reads them. The substructure of the PackMod directory is a copy of that in the EntModels directory with an exact copy of the directory structure but with MOD files instead of MDL files.

Se2

Some LWO files are not converted directly into the PackMod MOD format, but instead into the SE2 format, for “Serialized Entity version 2”. This format is less complex and slightly faster, but is only used for a few types objects in the entire game. The PackMod.exe recognizes these files by itself and places relevant SE2 files in this directory.

Showbox

Some image backdrops for the Object Manager show box function (not intentionally included, but some people have reported that they have found the ObjectManager.EXE inside their DATA directory. Sadly, no one has reported that they have found the WeaponConf.EXE program, which greatly helps weapon type editing, regardless of what it’s author thinks of the program itself. ☺

Sound

The directory structure of the SOUND directory is very important and is described in great length in **its own chapter**. To exchange a sound from the game with another, just place it in the correct place in the directory tree with the right name and the new sound will be loaded.

Textures

All model textures are placed here in any order or disorder you wish. It is more important to have a good describing name for the texture than to place them in a subdirectory structure, although some crude structuring indeed helps.

SECTION 2 – FILE TYPES

*.ATM

Data\AttiributeModifiers\*.atm

Some special equipment uses attribute modifiers. Each file contains a couple of key values such as effect and duration. More information is available in **Unit Creation.**

*.BIK

Data\Cutscenes\*.bik

Cutscenes are in the RAD Gametools' BINK™ format and are placed in this directory. They can later be called from *.CAM files to be played between missions in a campaign.

*.CAM

Data\campaigns\nameofcampaign\nameofcampaign.cam

Campaign definition file. Within it lies defined what missions are included in a campaign and in what order, the different squads that become available to a player progressing through the campaign, the states of the planet etc. The file format is described more closely in Section 3.

*.CUT

Ground Control™ sometimes uses small "windows" in which 3D models move around just like inside the game. These windows are called "show boxes" and the events inside are controlled by the CUT file format ("CUT" for "cut scene"). The format is explained in **CUT File Syntax.**

Dialogs (*.TXT)

Data\dialogs\*.txt

Contains definitions of all the different dialogs and screens in the game's interface. It contains many lines beginning with PICTURE, the coordinates and name of the image used and this specifies how the screen looks. PICTURE command images are found in Data\Textures\Gfx and consist of corners and buttons used throughout the game. Even though the functionality of the buttons cannot be changed, their visual representation can be changed, they can be moved around or made invisible. The Dialogs format would need a dedicated definition to ease use. See provided example files for reference so far.

*.ENT

Data\entfiles\*.ent

Entity descriptions used for AI and user-controlled units, turrets and buildings. Defined in **Entity File Syntax** and **Unit Creation**.

*.FAM

Data\GenEdData\TextureFamilies\*.fam

Files describing what texture tile sets are used to texture ground in GenEd and GC. It specifies the internal relationship between different types of tiles, and ties them to the relevant textures. See example files for reference.

*.LF

Data\EntModels\sun_lensflare.lf

A lens flare object defined in **MDL-language specification**. File is in MDL format.

*.LOC

Data\localization\*.loc etc

These files contain localized text strings, one per line where each line begins with a number, then a tab. The number is used as a “LOC ID” – or localization string identifier, which is then called from other files that define where the strings themselves are used. For example the file weapons.loc is accessed from managementdata\weapons.txt – all areas in the latter where text is substituted for a number, has a corresponding string of text in the LOC file. Please consult the included LOC files for their exact use. The file format is mentioned in **Entity File Syntax** as well as **Unit Creation** but is best defined in the example data.

*.LWO

Data\Models\*.lwo

These files are Lightwave™ Objects. Lightwave™ is a 3D-modeling tool that can create or edit three-dimensional objects, which can then be used for example in Ground Control™. The procedure for adding a Lightwave™ object to Ground Control™ is specified in the **MDL-language Specification** and the **Unit Creation HowTo.**

Map Files

Data\missions\ & data\multiplayer\missions

These files are accessed using GenEd and throughout the mission design process. All these files are documented in the **GenEd Manual,** and will not be repeated in this SDK document.

*.MDL

Data\entmodels\*.mdl

Model representation definition. This file is the link between the **.LWO Lightwave™ object file; it’s states, spline motions** (.MOT files), particle generators (*.PG files) and tying it into GC. The file format is defined in **MDL-language Specification.**

*.MOD

The PackMod file format is a compact and fast proprietary 3D model format used by Ground Control™. Models **should** exist in *.MOD format for Ground Control™ to use them. If you have *changed* any models originally existing in Ground Control™, you must first run the PackMOD.exe program to convert all your LWO and MDL data files into MOD files. You must do this to see your changes reflected in Ground Control™, if not, Ground Control™ will choose to use the MOD files that are already stored in the SDF data.

The real difference between MOD files and separate MDL files is that the MOD file is optimized for the engine and allows GC to more efficiently read and use models. Not using MOD files will lower game performance and raise memory requirements, as well as slow down loading. And, as I mentioned already, changes in MDL files will not be reflected until MOD files are removed (which can be a problem if you are trying to change a model that is already in Ground Control™ as a MOD file).

*.MOT

data\EntModels\Motions\*.mot etc

A spline motion exported from Lightwave™ 5.5 / 5.6 used to describe entity motion. Usage defined in **MDL-language Specification.**

*.MPD

Data\sounds\*.mpd

Streaming sound data converted from MP3 data by the MP3toMPD.exe program. Used in missions’ sound tracks & message boxes, briefings and debriefings. MPD files can be replayed by most MP3 software, but MP3 files cannot be directly used by Ground Control™. Described in **Section 6 – Audio & Music.**

*.PG

Data\entmodels\*.mdl

Particle generator file, defining a burst of particles, their speed, acceleration, spread, distribution pattern, fade-out and general behavior and of course what textures to be used for the particle. The file format is defined in **MDL-language Specification.**

*.SCT

See MAP FILES

*.SDF

Data\ & data\maps

SDF is a compressed file format that can hold any amount of Ground Control™ files and their directory structure. The SDF files placed in data\maps contains map data for multi- and single player missions while those normally in data\ contain all of the Ground Control™ data in a compressed format easily accessed by Ground Control™. SDF-maps are created by GenEd other SDF files (such as mod-SDFs) are created by the SDFman program. Understand that the SDF file contains a copy of the Ground Control™ directory structure, but that actual files always override files inside SDFs (and files in a MOD SDF also override the original SDF files).

*.SE2

The SE2 file is a serialized version of an LWO file that is more easily read by GC. There are only a few types of files that need to be accessed by GC in this manner

*.TEC

Data\ManagementData\default.tec; multiplayer.tec & data\Campaigns\nameofcampaign\nameofcampaign.tec

Technology level definition file. This file consists of three tags for each tech level available as well as one at 999 for unavailable squads. The three tags signify new entities, pieces of equipment and weapons made available at the specified tech level. The default.tec file is used for custom games and the multiplayer.tec file is used for multiplayer games. See the example data and **Campaign Files f**or more information.

Textures (*.TGA)

Data\Textures\*.tga

32-bit RGBA or 24-bit RGB uncompressed Targa (*.tga) format images. Used by 3D models, particle effects, interface dialogs, everywhere. Most textures can use transparency (alpha). See the enclosed data for usage. Textures are called from Dialogs, *.lwo files, *.pg files, *.lf files and many others.

*.TGA

See *TEXTURES*.

*.WAV

Data\sound\*.wav

PCM-compressed sound data for used as sound effects, squad voice feedback and other smaller sound bites. Normally in 16bit, 22khz base frequency. Described in **the sound section.**

SECTION 3 – SPECIFIC FILES' FUNCTIONS

An overview of the file structure, syntax and vocabulary used in Ground Control™ and its data files.

Unit Creation, general

A unit is composed of several different files. The entity file (*.ent) is the top file requesting the other files and is used by the game engine and GenEd.

- .ent file, for entity description, attributes, etc
- .mdl file(s), for visual representation
- .pg file(s), for visual representation (.mdl files use these)
- .lwo file(s), for visual representation (.mdl files use these)
- .tga file(s), for textures (.mdl, .pg, and .lwo files use these)

All text strings are localized and can be found in the Data\Localization\entfiles.loc. Equipment for a unit is specified in the equipment.txt (and localized in equipment.loc). Weapons for a unit is specified in weapons.txt (localized in weapons.loc).

To add a new unit to GC:

1. Create a Lightwave™ 5.5 / 5.6 model and name it. Save it under Data\Models\ in a directory structure that suits it. Remember to use a child model for aimer if you want the entity to have a prehensile cannon.
2. Create an MDL file for the model and place it under EntModels
3. Create a "small" MDL for the management interface, placed in Data\EntModels\Management by the same name as the MDL but with an added "_MANAGEMENT", for example "CC_AAVEHICLE_MANAGEMENT.MDL". The model only needs a stand animation and is only displayed in the unit selection on the drop ship loading screen.
4. Create an ENT file for the unit and place it in EntFiles\. Make sure you give it an aimer if you want it to be able to use weapons and give it a mover if it is a vehicle.
5. Open the file Data\Management\Data\weapons.txt in a text editor. Find a suitable weapon (or several) in the file that you would like your new unit to be able to use and add two new lines to that weapon, for example..
```
Unit
Cc_aavehicle
```

To add a new player selectable unit to GC:

1. Create the unit and make sure the ENT-file is correct, that the combination of *Branch* and *BranchSortOrderValue* is available, where *Branch* is the TYPE of unit – INFANTRY, SUPPORT, TANK, AERODYNE for player units, FIXEDGUN, BUILDING are also available – and *BranchSortOrderValue* means what number in the unit list that entity should hold. For example, marines are the first type of infantry, while jaegers are the second.
2. Create an armor listing for the unit in the armor.txt under Data\ManagementData.
3. List the unit in the entconfigs.txt in Data\ManagementData Each player selectable unit has four different configurations (balanced, offensive, defensive and recon). The entconfigs.txt specifies which **armor** and **weapon** each configuration has. The name strings are localized to Data\Localization\EntConfig.loc where strings can be changed, reused or appended and then called using the LOC ID (localization identification number).
4. Assign Special Equipment to the unit by editing the equipment.txt file in Data\ManagementData. A unit can as a maximum have four different pieces of equipment available. Localization is done in Equipment.loc in Data\Localization where strings can be added, changed and reused.
5. Open the file Data\Management\Data\weapons.txt in a text editor to add a special weapon. Find a suitable weapon or two or create a new weapon post in the file that you would like your new unit to be able to use and add your unit to the list of units able to use this weapon. Please note that Special Weapons **must** be marked with the SpecWeaponFlag 1 tag.
6. Add the unit, its Special Weapons and it's Special Equipment to the tech level lists in:
   - Data\ManagementData\multiplayer.tec (this sets the tech level for the unit in multiplayer games)
   - Data\ManagementData\default.tec (this sets the tech level for all custom missions)

- Data\Campaigns\yourcampaign\yourcampaign.tec (this sets the tech level for the actual campaign)

Remember: When *changing* any data files (LWO, MDL) for a model in the game, remove any MOD files in the Data\PackMod directory referring to the object you have changed, or the changes will not be available from the game. The game continually creates MOD files from MDL and LWO files when they are used, for easier access to the model data but the MOD file will not be updated once it is created and must therefore be deleted and recreated by GC. If you are modifying models that are already in the game, you must also run the PackMod.exe program to create new MOD files to replace the files inside the game's SDF files. This is only true for changing models already in the game (same name, same use).

Management Data Files

Armor.txt

Location:
Data\ManagementData\armor.txt
Localized in:
Data\Localization\armor.loc

The armor.txt file contains the armor and certain modifiers for player controlled units. Each unit requires four separate armor entries, one for each entity edition (balanced, offensive, defensive and recon).

```
ID
//ID is used to identify the armor and used in the entconfigs.txt
file.
Marine Balanced
STEALTHMODIFIER
//Stealth modifier is multiplied with the stealth value of the unit
to give the stealth value of the edition. This value was never
changed from 1 in Ground Control™ and Dark Conspiracy, but there is
no reason for it not to be usable.
1
SPEEDMODIFIER
//Speed modifier is multiplied with the max speed of the unit to give
the speed of the edition.
1
ARMORMODIFIER
//Armor modifier is multiplied by the POWER (health) of the unit,
resulting in the health of the entity edition.
1
```

In GC, we used the following modifiers on all types of units:

```
Balanced:     SpeedModifier 1.0     ArmorModifier 1.0
Defensive:    SpeedModifier 0.75    ArmorModifier 1.2
Offensive:    SpeedModifier 0.75    ArmorModifier 1.0
Speed/Recon:  SpeedModifier 1.2     ArmorModifier 1.0
```

Entconfigs.txt

Location:
Data\ManagementData\entconfigs.txt
Localized in:
Data\Localization\entconfigs.loc

The entconfigs.txt is used to connect the different unit configurations (balanced, offensive, defensive and speed) with correct armor, weapon, name, description and stats values.

Note: The Command APC is only available in one configuration and its statistics are never used.

```
//Each entconfig entry starts with [ENTCFG]
[ENTCFG]
//ID of the entity configuration, these are used in the squad files.
(This is a marine in balanced configuration).
MARINE_B
//The .ENT-file of the Entity in question.
cc_marine
//Localized   name-ID   that   appears   in   the   front-end   on   the
configuration buttons in the squad configuration screen.
0
//Localized description-ID, never displayed in GC.
1
//ID name of the assigned primary weapon. See weapons.txt.
Assault Rifle Mk. 1
//ID of the assigned armor used. See armor.txt.
Marine Balanced
//Firepower value for the equip squads screen, 0-100
20
//Armor value for the equip squads screen, 0-100
30
//Speed value for the equip squads screen, 0-100
5
//View range value for the equip squads screen, 0-100 (note that in
GC this never changed in different configurations of the entity)
60
```

Note that the values are set manually and are actually a rather bothersome task, as you need to compare with all other units. Use a handy spreadsheet with all stats… Remember also that these values do not **control** any actual behavior; it is only for visual display.

Equipment.txt

Location:
Data\ManagementData\equipment.txt
Localized in:
Data\Localization\equipment.loc

The equipment.txt lists all equipment, the duration, and number of charges, recharge time and the units that can use it. Equipment that modifies unit attributes (accuracy, stealth etc) are linked to files in Data\AttributeModifiers. See table below for more information.

EQUIPMENTID	COMMENT
SE_HEALTH1	Used by Crayven infantry. Heals 1000 health when used. Uses Health1.atm
SE_HEALTH2	Used by Crayven vehicles. Heals 1000 health when used. Uses Health2.atm
SE_HEALTH3	Used by Order infantry. Heals 1200 health when used. Uses Health3.atm
SE_HEALTH4	Used by Order vehicles. Heals 1200 health when used. Uses Health4.atm.
SE_RADARSTATION1	Crayven. Deployable radar station uses ENT-file: cc_radarstation1.ent
SE_RADARSTATION2	Order. Deployable radar station. Uses ENT-file: wc_radarstation1.ent
SE_TRIGDEMO1	Crayven. Demolition charge.
SE_VIEWRANGE1	Crayven infantry. Increases view range. Viewrange1.atm
SE_VIEWRANGE2	Crayven vehicles. Increases view range. Viewrange2.atm
SE_VIEWRANGE3	Order infantry. Increases view range. Viewrange3.atm
SE_VIEWRANGE4	Order vehicles. Increases view range. Viewrange4.atm
SE_PERSONALCLOAKER1	Order infantry. Increases stealth. Uses Personalcloaker1.atm
SE_PERSONALCLOAKER2	Not assigned. Uses Personalcloaker2.atm
SE_ACCURACY	Crayven. Increases accuracy. Uses Accuracy.atm

SE_DEPLOYABLECLOAKER	Order. Creates an LOS blocking field.
SE_AFTERBURNER	Crayven aerodyne. Increases speed. Afterburner.atm
SE_SPEED	Order vehicles. Increases speed. Uses Speed.atm
SE_COUNTERMEASURE1	Crayven. Shoots down enemy bombs, missiles and artillery
SE_COUNTERMEASURE2	Crayven. Shoots down enemy bombs, missiles and artillery
SE_PROTECTORSHIELD1	Order. Invulnerability. Uses ProtectorShield1.atm
SE_PROTECTORSHIELD 2	Reinforcement effect. Invulnerability. Uses ProtectorShield2.atm
SE_SENTRY_AUTOCANNON	Crayven. Deployable auto cannon. Uses cc_SentryAutocannon.ent and weapon: cc_sentryautocannon
SE_SENTRY_ROCKET	Crayven. Deployable rocket launcher. Uses cc_SentryRocket.ent and weapon cc_sentryrocket
SE_SENTRY_ANTIAIR	Crayven. Deployable AA launcher. Uses cc_SentryAntiAir.ent and weapon cc_sentryantiair
SE_DEPLOYABLEDEFENDER_UNIT	Order. Deployable defender gun. Uses wc_deployabledefenderunit.ent
SE_DEPLOYABLEDEFENDERGENED	GenEd. Deployable defender gun used in mission scripting. Uses cc_deployabledefendergened.ent
SE_REPAIRSTATION	Crayven. Deployable repair station. Uses cc_repairstation.ent
SE_MINEFIELD1	Crayven. Cluster of four mines.
SE_MINEFIELD2	Order. Cluster of four mines.
SE_PERCEPTION	Not assigned. Increases perception. Perception.atm

The following is a section of the equipment.txt to explain the syntax:

```
EQUIPMENTID
 The ID of the equipment. Note that the ID is hard coded and controls
 game play effect and equipment icon.
SE_HEALTH1

NAME
 The name of the equipment that is displayed in the configure squad
 screen. Localized in equipment.loc.
0

DESCRIPTION
 A description of the equipment, never displayed in GC. Localized to
 equipment.loc.
1

CHARGES
 The number of times it can be used before it is depleted. Only seven
 charges will be visible in-game, and anything more than 5 charges
 will overlap the equipment button, looking ugly, but more than 7
 charges can be used here, and will be taken into account in the game.
3

RECHARGETIME
 The time in seconds the equipment requires before it can be used
 again.
1.0

DURATION
 The duration of the equipment in seconds.
1.2

SLOTS
 GC does not use the slots tag in the retail version. It was intended
 as a "weight" value where a unit could only take a certain amount of
 equipment, comparable to a Mech game.
2
```

```
UNIT
 The units (EntFiles) that can be equipped with the equipment. Note
 that each unit listed needs to be preceded by a new UNIT keyword:
cc_marine
UNIT
cc_jaeger

ESMODELNAME
 Model of the equipment. Not used by GC, so use placeholder.mdl.
placeholder.mdl
```

AttributeModifiers (.ATM)

Location
Data\AttributeModifiers\*.atm

Attribute modifiers are used by equipment to change the attributes of a unit. Note that it is possible to have several different tags for each Attribute modifier. The following tags are available for the Attribute modifiers:

MaxSpeed
<parameter>
Affects the Max Movement Speed of the unit by a factor of this number. 2.0 doubles max speed.

Acceleration
<parameter>
Increases the acceleration of the unit by a factor of this number. 2.0 thereby doubles acceleration.

Health
<parameter>
Replenishes lost power of the unit up to this value.

Accuracy
<parameter>
Increases the accuracy of the unit by a factor of this number, so that 2.0 doubles accuracy.

ViewRange
<parameter>
Increases the view range of the unit by this many meters.

Perception
<parameter>
Increases the perception of the unit by this many meters. .

Stealth
<parameter>
Increases the stealth of the unit by this many meters.

ProtectorShield
1
Gives invulnerability to the unit. Parameter should be 1 to use.

Timer
<parameter>
The duration of the attribute modifier, in seconds.

OneShot
This tag indicates that the attribute modifier is instantaneous and permanent (i.e. health packs). It takes no parameters.

Weapons.txt

Location:
Data\ManagementData\weapons.txt
Localized in:
Data\Localization\weapons.loc

To add a new weapon available to AI units, add a weapon to the weapons.txt and assign it to the correct unit. In order to add a new weapon to player controlled units:

1. Add the weapon to the Weapons.txt file (and weapons.loc file if it has a new name).
2. Add the weapon ID to the default.tec, multiplayer.tec and correct yourcampaign.tec file.
3. Add the weapon to the correct unit in the entconfigs.txt file.

NOTE! There is an upper limit to how many weapon types that can be defined in Ground Control™ at one time. The **weapons.txt** file may contain a maximum of 255 weapons, never more.

BEGIN WEAPON

This indicates that the following is the definition of a new weapon. The definition does not end until there is another BEGIN WEAPON line.

ShortName

The ID of the weapon. This identifying name is used by GenEd.

LongName

The name of the weapon (localized into Data\Localization\Weapons.loc). Only the names of Special Weapons are displayed in game.

Accuracy

The base accuracy of the weapon. At medium range, perfect visibility and level terrain, 100 equals 100% accuracy. These conditions almost never exist in the game and therefore excellent accuracy is often higher than 100.

Range

The maximum functional range of the weapon, in meters.

Damage

The damage of the weapon, in power points. When a weapon with a Damage Multiplier of 1.0 hits an unarmored target, this is the amount of "health" that is deducted from the target.

DamageMultiplier

The damage multiplier is multiplied with the resulting damage after the armor value of the hit unit has been detracted.

`Example`: A bullet with a damage value of 120 and a damage multiplier of 3 hits a target with an armor value of 100 and a health of 200. 100 points of damage is subtracted from the damage value of the weapon and 20 remains. This amount of damage is then multiplied by the damage multiplier of 3 and a resulting 60 points or damage is therefore subtracted from the health of the target. The target's health sinks by 60 points and ends up at 140 health points (or 70%).

ReloadTime

The time in seconds between shots in a burst.

BurstCount

The number of shots in a burst where each shot is separated by the reload time mentioned above.

BurstDelay

The time delay in seconds between bursts.

BlastRadius
All objects within the blast radius receive damage. Damage is only full in the middle (point blank) and drops off towards the edges in a gauss curve. This specific blast radius value applies ONLY to bullet weapons since other types (rocket, homing, bombs) have a BlastRadius value on their mover that should be used instead.

GroundAttackFlag
Set to 1 for the weapon to be able to target entities on the ground (hovers count).

AirAttackFlag
Set to 1 for the weapon to be able to target flying entities (not hovers).

SpecWeaponFlag
Set to 1 to make the weapon a special weapon.

SpecWeaponIcon
Only used by Special Weapons. The icon displayed in the game to activate the Special Weapon. –1 means no icon, used for non-special weapons. These icons can be found (and replaced) in Data\Gfx\SpecialWeaponIcons but they are numbered instead of named. I can only urge you to look at the weapon whose icon you want, and use the same number.

Charges
Only used by Special Weapons. Charges indicate the number of charges before the weapon is depleted. Only seven charges will be visible in-game, and anything more than 5 charges will overlap the equipment button, looking ugly, but more than 7 charges *can* be used here, and will be taken into account in the game.

SpawnSound
Select sound for when the projectile is spawned. Possible sound banks are listed (and can be added) in Data\ManagementData\WeaponSounds.txt and are **Grenade1-2**, **Gun1-3**, **MachineGun1-3**, **Rocket1-3**, **Energy1-3** and **DropBomb**.

SpecialBehaviorID
Gives the weapon / projectile a special ability: Most weapons have this value at 0, meaning no special behavior. However, if this value is 1 and the projectile is homing, it will follow the ground. If the weapon is a bullet, it will "bounce" when hitting a target and might strike other targets. **Warning:** Using this flag on any other mover than BULLET or HOMING may cause unwanted results.

MinFireRange
The weapon cannot be fired at a target within the Minimum Fire Range. Mainly used for Artillery and Homing missiles.

Type
This is the name of the type of weapon this is… it was mainly used as a sorting aid and has no real relevance unless you are using the WeaponConf program (never released). However, it must be set to an existing weapon type as listed in WeaponTypes.exe in ManagementData.

Unit
Several *Unit*-lines are possible for each weapon and signifies that the weapon is available to the listed entity. Names listed are the same as the names of the Entity files (*.ENT).

GroundHitEffect, ExpireEffect, MetalHitEffect, FleshHitEffect
These four effects define what occur when the projectile hit the **Ground**, **Nothing** (time out / destroyed in flight), **Metal** and **Flesh** respectively. The hit effects can be either nothing at all or a simple pre-defined effect, listed in ManagementData\Effects.txt – **None, Blood, Small Explosion, Medium Explosion, Large Explosion, Smoke, Dirt, Spark, Energy Hit** and **Plasma Hit** – or it can be an EntModel (*.MDL) effect which can in look like anything. Whereas the predefined effects have

predefined sounds, the MDL needs a defined spawn sound, listed in EffectSounds.txt in ManagementData – **SmallExplosion, MediumExplosion, LargeExplosion, MetalHit, GroundHit, EnergyHit, HumanHit** or **Junk.** It is also possible to add more than one effect of each kind, where the effects will be randomized.

All four kinds of effects need to be listed and in this order. Whereas the simple, predefined effects need only be written on the line below the name of the effect, like this…

```
        ExpireEffect
        Smoke
```

…the EntModel effects take three parameters. To use EntModel effects, the word “EntModel” is written on the line below the name of the effect type, and then the three parameter-names and their data are listed on six lines. The parameters are **ModelName** – the entire filename of the MDL file used, including .mdl extension, **EffectSound** – one of the sounds listed above and **TimeToLive** – in seconds, how long the EntModel effect is displayed for after the impact.

MDL hit effects need to be called from EntModels\HITEFFECTS\ (or its PackMod equal).

BEGIN MOVER

This notates that the following attributes are those of the projectile “mover”. It still concerns the weapon, but now the actual moving part.

MoverType

The mover sets the weapon’s movement type and gives it certain abilities. Different movers have different attributes listed in their mover definition – the lines between BEGIN MOVE and BEGIN REPTYPE. Please note that “anti-missile equipment” indicates defender guns, missile defenders and anti-missile systems.

Name	**Attributes**	**Characteristics**
Ballistic	BlastRadius HighOrLowFlag	(Artillery, mortars) Set flag if projectile should use a high (45+ degrees) ballistic arc ONLY. Projectiles are affected by anti-missile equipment.
Bullet	Speed	Set projectile speed. Projectiles are **not** affected by anti-missile equipment.
Drop Bomb	BlastRadius	(bomber aerodyne) Projectiles are affected by anti-missile equipment.
Homing	StartSpeed MaxSpeed Acceleration BrakeFactor TrackDelay BlastRadius	Automatically hits target - no need to set accuracy. Several missile attributes must be set. Homing cannot target infantry / flesh targets and cannot be force fired.. Homing weapons time out after 10 seconds and self-destruct. Projectiles are affected by anti-missile equipment.
Flame	StartSpeed BrakeFactor TimeToLive	Gives damage over time. Not affected by anti-missile equipment. This mover type has never been used in GC or Dark Conspiracy. May not be operational.
Rocket	StartSpeed MaxSpeed Acceleration BlastRadius SpinOffset SpinSpeed	Dumb-fire / Non-homing rocket. Set max speed, acceleration etc. Projectiles are affected by anti-missile equipment. SpinOffset and SpinSpeed relate to how the projectile moves through the air around its center of movement.

BEGIN REPTYPE

This notates that the following attributes are those of the representation in 3D, i.e. how the bullet, beam, bomb, missile or rocket actually looks in the game.

RepType

There are three types of representation which all have different attributes and characteristics.

Name	**Attributes**	**Characteristics**
3DRep with Particle Trail	3Dobject TextureFile	LWO file (Lightwave 5.5/5.6 object) with a trail after it, made out of particles, looking

	SpawnInterval ParticleLifetime ParticleDistortion ParticleSize	like the texture file. SpawnInterval decides how dense the trail is; lifetime how long it lasts while distortion adds a little random effect.
3DRep with Polygon Trail	3Dobject TextureFile TrailLifetime TrailSize SpawnInterval	LWO file (Lightwave 5.5/5.6 object) with a trail built up of textured polygons. The trail has a lifetime before it ends and a size. SpawnInterval should not apply.
EntModel Rep	ModelName	EntModel (MDL) projectile.

Campaign files

NOTE: It is highly recommended that you read the GenEd Manual in conjunction with this section.

The files in the Data\Campaigns\ directory specify the outline of the campaigns. Each campaign requires it's own directory in the campaign directory. The *.TEC file specifies the tech levels set for the campaign. The *.CAM file is localized in the *.LOC file in the same directory (NOT in Data\Localization\). The three game campaigns (Tutorial, Crayven & Wrath) are hard-coded to be used but they can be replaced / overridden by new campaigns by the same names as well as expanded on (by adding more missions to the campaign cycle).

Tech levels file (.TEC)

Data\ManagementData\default.tec – tech level for custom missions.
Data\ManagementData\multiplayer.tec – tech level for multiplayer games.
Data\Campaigns\yourcampaign\yourcampaign.tec – tech level for individual campaigns.

The Tech level files exist to make sure that units, equipment and special weapons are introduced at the correct mission or in the correct multiplayer tech level.

```
[TECHLEVEL n ENTITIES]
First a list of all entities (ENT file name) available at level n and
up.
[TECHLEVEL 1 ENTITIES]
cc_marine
cc_scoutvehicle

[TECHLEVEL n SPECIAL EQUIPMENT]
List of all equipment available to n level and up. The equipment ID
is used.
[TECHLEVEL 1 SPECIAL EQUIPMENT]
SE_PERCEPTION
SE_HEALTH1

[TECHLEVEL n WEAPONS]
List of all weapons available to n level and up. The Weapon ID is
used.
[TECHLEVEL 1 WEAPONS]
HE Mini Bombs
```

Campaign file (.CAM)

```
[CAMPAIGN]
// Name, localized in the *.loc file.
0
// Force intended to play the campaign (CRAYVEN or WRATH, where wrath
is the Order of the New Dawn)
CRAYVEN
// Mdl-filename for campaign model, not used in GC.
entmodels/crayvenlogo.mdl
```

```
 // Description of the campaign, localized in the *.loc file.
1
 // Perquisite campaign (if any)
tutorial
 // The list of missions and in what order they appear. Missions are
 referred to by directory name, not their full name.
[MISSIONLIST]
Mission1
Mission2
Mission3
Mission4
Mission5
Mission6
Mission7
Mission8
Mission9
Mission10
Mission11
Mission12
Mission13
Mission14
Mission15
 //The planet states are for the progress map in the debriefing
 screen. One row for each mission in the campaign. Please note that
 the matrix can only be 15 columns wide (15 map zones).
 // 0 = zone not lit
 // 1 = zone is contested (blinking)
 // 2 = zone is friendly (blue)
 // 3 = zone is hostile (red)
 [PLANETSTATES]
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 1 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0 0 0 0 0 0 0
2 2 2 1 2 2 0 0 0 0 0 0 0 0 0 0
2 2 2 2 2 2 1 0 0 0 0 0 0 0 0 0
2 2 2 2 2 2 2 1 0 0 0 0 0 0 0 0
2 2 2 2 2 2 2 2 1 0 0 0 0 0 0 0
2 2 2 2 2 2 2 2 2 1 0 0 0 0 0 0
2 2 2 2 2 2 2 2 2 2 1 0 0 0 0 0
2 2 2 2 2 2 2 2 2 2 2 1 0 0 0 0
2 2 2 2 2 2 2 2 2 2 2 2 1 0 0 0
2 2 2 2 2 2 2 2 2 2 2 2 2 1 0 0
2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 0
 //The cutscenes tag indicates where and what cutscenes should be
 played. Usage:
 //Cutscene_name.bik <prequisite_mission_name>
 //Crayven_1.bik is played as the campaign starts
 //Crayven_2.bik is played AFTER mission6
[CUTSCENES]
crayven_1.bik
crayven_2.bik <Mission6>
crayven_3.bik <Mission10>
crayven_4.bik <Mission15>
//Newsitems is redundant and has no effect.
[NEWSITEMS]
 //The squad leader tag specifies when in the campaign the player
 receives new squads and what those squads are. E.g. Before mission 1,
 the player receives 1 cc_marine. It is possible to specify how the
```

unit is configured and which drop ship they should be inserted into. See file Data\Campaigns\tutorial\tutorial.cam for examples.

```
Format:
<__MissionName> ENTITY xxx [WEAP xxx SPECWEAP xxx ARMOR xxx EQU1 xxx
DROPSHIP xxx]
 [SQUADLEADER]
<__Mission1> ENTITY cc_marine
<__Mission2> ENTITY cc_marine
<__Mission3> ENTITY cc_lighttank
<__Mission4> ENTITY cc_lighttank
<__Mission6> ENTITY cc_scoutvehicle
<__Mission8> ENTITY cc_mediumtank
<__Mission9> ENTITY cc_artilleryvehicle
<__Mission9> ENTITY cc_artilleryvehicle
<__Mission10> ENTITY cc_lightaerodyne
<__Mission10> ENTITY cc_lightaerodyne
<__Mission12> ENTITY cc_jaeger
<__Mission12> ENTITY cc_rocketvehicle
<__Mission13> ENTITY cc_rocketvehicle
<__Mission13> ENTITY cc_bomberaerodyne
<__Mission14> ENTITY cc_heavytank
<__Mission15> ENTITY cc_heavytank
<__Mission15> ENTITY cc_aavehicle
<__Mission15> ENTITY cc_attackaerodyne
```

In-game cutscenes

Movie File Syntax (.CUT files)

Command List (with parameters):

[FRAME t]

At times t, all commands until next **[FRAME]** are executed (except triggered ones). All commands execute at the same time, in top to bottom order. t is an absolute time, from beginning of movie.

To get a relative time, it is possible to use **TRIGGER** before the command, and add an extra parameter last, which is a relative time.

MODEL model.mdl

Load a new main model (old one deleted)

CAMERA xx yy zz rx ry rz

Sets camera's position and viewing-direction. Xx, yy and zz signify the position in the scene and rx, ry and rz notate the directional vector the camera is pointed along. NOTE: only works when camera isn't connected to an entity or a spline (MOT-file).

FOV aa

Sets camera field-of-vision angle.

CONNECTCAMERA [entity name] [motionfile.mot time]

1. Neither entity name nor motionfile/time is specified: Camera is disconnected and can accept CAMERA and FOV commands.
2. Entity name specified: Camera connects to an entity and follows it around.
3. Motionfile and time: Camera follows a predefined spline. Time is spline-length (in seconds).

FOG ENABLE/DISABLE

Enables or disables fog.

FOGCOLOR r g b

Sets the color of the fog.

FOGDISTANCE m
Sets the distance to fog start (i.e. number of meters that is not fogged).

FARPLANE m
The visible distance in meters. Best used with fog

VIEWPORT sx sy ex ey
Coordinates for the view port (sx, sy) upper left corner, (ex, ey) lower right corner. Full screen is 0 0 640 480 (the resolution played in).

BACKGROUND picture.tga
Loads a full screen background (slow!)

DIRECTIONALLIGHT r g b rx ry rz
Adds a directional light to the scene (8 lights max)

CLEARDIRLIGHTS
Removes all directional lights from the scene.

AMBIENTLIGHT r g b
Set ambient light

OBJECT xx yy zz rx ry rz
Not implemented.

ANIMATION [entity-name] animation-name
Sets animation for an entity (and all its children). If entity name isn’t specified, the entire model (and all its children) changes animation state.

PLAYSAMPLE sample.wav
Play a sample once. Four samples can be played at the same time.

PLAYMUSIC music.wav
Play a looping sample

MOVIE moviefile.cut
Load a new movie file (current one stops running)

Example:
```
[FRAME 0]
MODEL entmodels/hepp.mdl
CONNECTCAMERA
CAMERA 10 0 23.4 0 0 35
FOV 90
TRIGGER CAMERA 11 1.2 21.2 0 0 30 1.2
TRIGGER CAMERA 11 4.2 21.2 0 0 30 1.8
TRIGGER CAMERA 11 6.2 21.2 0 0 30 2.3
[FRAME 3.7]
CONNECTCAMERA spaceship
TRIGGER ANIMATION spaceship explosion 1.4
[FRAME 6.0]
CONNECTCAMERA
MOVIE nextscene.igm
```

Units & Models

Entity File (.ENT)

Entity files (.ENT) are used for describing Entities in the context of Ground Control™. An Entity is for example a building or combat unit, which all have the characteristics of being “killable”, in that they have hit points and can be destroyed. Some can move (have a Mover), some can aim and shoot (have an Aimer and a Weapon), and some do not. All combinations are possible. An Entity is distinctly different from a Prop (see GenEd documentation for more information on Props).

A description of all legal tags and their syntax follows. One text / value per line, all text is NOT case sensitive unless otherwise noted (I believe the only strings that ARE case sensitive are the Mover types, see MOVER below). Tags are in headings, values follow on the lines below (when there are a fixed number of alternatives, these are noted delimited by /). Values are of floating point, integer, or string type, as noted. Comments are bulleted below. In most cases, you can omit the tag altogether to not use it and apply the default value (for example, for no Mover, omit the MoverType tag). All default values are legal (as far as we know).

If the script language appears to be “ad-hoc”, that is because it is. It has evolved during the entire development of Ground Control™ (a period of 3 years) very much free from any kind of standard. Good luck!

– Johannes Norneby, Lead Programmer, Massive Entertainment

ACCELERATION

(float)acceleration

- Acceleration of the Entity, given that it has a Mover, must be larger than FRICTION.
- m/s/s.

AMBIENTSOUND

(string)SND_ENGINE_DROPSHIP/SND_ENGINE_HOVER1/ SND_ENGINE_HOVER1/SND_ENGINE_VEHICLE1/ SND_ENGINE_VEHICLE2/ SND_ENGINE_VEHICLE3/SND_ENGINE_AIR1/SND_ENGINE_AIR2/SND_AMBIENT_BUILDING1/ SND_AMBIENT_BUILDING2/ SND_AMBIENT_BUILDING3/ SND_AMBIENT_BUILDING4/ SND_AMBIENT_BUILDING5/ SND_AMBIENT_BUILDING6/ SND_AMBIENT_BUILDING7/SND_SND_EASTEREGG_TALLER/SND_EASTEREGG_BIRDSWARM/SND_EASTEREGG_FROG

- The engine sound or ambient sound of an Entity. For some buildings, this is a low hum, or the creaking of metal gears.
- The sound files for the sound banks reside in sound\ingame\entityambient.

AMBIENTSOUNDBASEFREQUENCY

(int)frequency

- The frequency (in Hertz) at which to play the Entity’s ambient sound.
- 0 will use the wave file default (22050 in GC).
- A value higher than 22050 will play the sound at higher pitch. 44100 will raise the sound by one octave.
- Values lower than 22050 will play the sound at higher pitch. 11025 will lower the sound by one octave.
- Sounds for vehicles are pitched up by speed.

APCBUTTON

(string)filename

- Filename of the alternate unit icon for this Entity type, used when the Entity is inside an APC Entity (without path but including .tga extension).
- These icons can be found in gfx/uniticons, and are 32bit Targa format (RGBA).
- In GC, only Infantry were specified as APCLOADABLE, and were thus the only Entity types to have an APCBUTTON specified.
- Any entity that has APCLOADABLE should have an APCBUTTON.

AIMERINXCONTROL

- The Aimer’s X-axis rotation controls the X-axis rotation of the entire Entity (not used in GC).

AIMERINYCONTROL

- The Aimer’s Y-axis rotation controls the Y-axis rotation of the entire Entity (like Marines or Jaegers).

AIMERXDEGLIMIT
(int)maxdegree
(int)mindegree

- Integer values in degrees specifying the maximum and minimum angles for the aimer around the X-axis.
- 0 is horizontal (straight forward), 90 signifies straight up, positive rotation is upwards along the Y-axis.
- Only positive angles are allowed, so to specify angles below 0, positive values are used. For example, for a tank to be able to aim 60 degrees above horizontal, and 10 degrees below, one would specify 60 for maxdegree and 350 for mindegree.

AIMERYDEGLIMIT
(int)maxdegree
(int)mindegree

- Integer values in degrees specifying the maximum and minimum angles for the aimer around the Y-axis.
- 0 is straight ahead, positive rotation is clockwise.
- Only positive angles are allowed, so to specify angles counterclockwise from 0, positive values are used. For example, for a tank to be able to aim 60 degrees clockwise, and 10 degrees counterclockwise, one would specify 60 for maxdegree and 350 for mindegree.

APC
- Specifies that the Entity can load other Entities inside itself (used in GC for the Command APC).
- It should be possible to use this for buildings, but make sure that the buildings don't use a SLOPEMODIFIER that units who should enter cannot cross. APC entities also have to be allied or neutral to the player team, meaning that an enemy unit **cannot** enter into an APC entity.

APCLOADABLE
- Specifies that the Entity can be loaded into another Entity that has APC specified (used in GC for all infantry).

BARRELOFFSET
(float)x
(float)y
(float)z

- Parameters are relative the Entity TURRET pivot point (0, 0, 0 in turret local space) in meters.
- Used for Tank/Fixed Gun barrels.
- Requires an EntModel child called BARREL.
- Note that a Barrel isn't required, for example, the Light Tank doesn't use a BARREL, only a TURRET. See CHILDOFFSET for more information.

BRANCH
(string)INFANTRY/SUPPORT/TANK/AERODYNE/FIXEDGUN/BUILDING

- Branch of the Entity.

BRANCHSORTORDERVALUE
(int)value

- The sorting order for branches in the configuration interface.
- The Configure Squads-screen has four slots per branch and units are sorted among those four slots using values ranging from 0 to 3. 0 will place the unit to the far left, 3 to the far right. For some reason 4 is available and will also place the entity in the far right. Please not that it is not possible to have more than 4 different unit types available per branch and faction.
- If a unit at a higher tech level has the same BranchSortOrderValue as one on a lower tech level, the higher tech level unit will replace the lower one when that tech level is reached.

BUTTON
(string)filename

- Filename of the unit icon for this Entity type (no path but including .tga extension).
- These icons can be found in gfx/uniticons, and are 32bit tga format (RGB with alpha channel).

CLASS

(string)SQUAD/BUILDING

- This is read by GenEd, and denotes the Entity types that can be used in squads via the script.
- It is VERY IMPORTANT to specify this tag for use with GenEd, as the file will be rejected if the tag is missing.
- Has no legal default value.

CHILDOFFSET

(float)x
(float)y
(float)z

- Parameters are relative the Entity pivot point (0, 0, 0 in entity local space) in meters.
- Used for Tank/Gun turrets.
- Requires the model to have an EntModel child called TURRET.

DEATHCHUNKMODEL

(string)entmodelfilename

- An EntModel (.MDL) to be thrown out from the destruction position of an Entity. Any number is allowed, and ONE instance of each is created upon Entity destruction (thus duplicates are also allowed).
- If the FLESHFLAG is specified, each "Death Chunk" will play a 3D sound of the type HumanKill (see sound/ingame/noise), if not a sound of the type LargeExplosion (sound/ingame/noise) will be played.
- Each DeathChunk will fly in a semi-random direction, scalable by DEATHCHUNKSPEEDSCALE, and live DEATHCHUNKTIMETOLIVE (in seconds) unless I has bounced more than 10 times (hard coded), in which case it will terminate immediately.
- DeathChunkModels were only used in GC for the deaths of Infantry and animals (with the FLESHFLAG tag specified).
- See entfiles/cc_marine for more details.

DEATHCHUNKSPEEDSCALE

(float)scale factor

- DeathChunk effects use by default a speed vector that is normalized to a range of +-1 m/s in the X and Z-axes, and between +1 and +2 m/s in the Y-axis. This is to ensure that the chunks always fly upwards.
- This default speed can be scaled by this scale factor, which will scale all axes uniformly.
- Negative values will revert to the default scale factor of 5 meters.

DEATHCHUNKTIMETOLIVE

(float)time

- Time to live for a DeathChunk effect.
- Default is 5 seconds.
- Negative values will revert to the default time of 5 seconds.

DEFAULTTEAM

(int)team number

- This is read by GenEd, and is used for the Team value in AddEntity actions (for the script).
- It is VERY IMPORTANT to specify this tag for use with GenEd, as the file will be rejected if the tag is missing.
- 0 – 9 are allowed, interpreted as following by GC:
  - SINGLEPLAYER
    - 0 = CRAYVEN
    - 1 = WRATH / ORDER

    - 2 = CRAYVEN ALLY
    - 3 = WRATH / ORDER ALLY
    - 4 = NEUTRAL
    - 5 – 10 = INVALID
  - MULTIPLAYER
    - 0 – 7 = VALID PLAYER TEAMS
    - 8 = NEUTRAL
    - 9 = ENEMY TO ALL
- No legal default value.

DESCRIPTION
(int)description index

- Zero based integer localization index of the description of the Entity.
- Localized descriptions also reside in localization/entfiles.loc.

EASTEREGGANIMALTYPE
(int)animal type

- This was only used in GC for the animals we added as Easter eggs at the very end of development.
- This tag affects the behavior of the AMBIENTSOUND that is played, specifically whether or not the speed of the entity shifts the frequency of the sound.
- Valid values are:
  - 0 = Normal entities, the default value, speed will shift frequency.
  - 1 = Taller, speed will not shift frequency.
  - 2 = Bird Swarm / Birdies, speed will not shift frequency.
  - 3 = Lone Bird / Snajder, speed will not shift frequency.
  - 4 = Frog / Alien Cow, speed will shift frequency.
  - Any other values will revert to 0.
- This is absolutely NOT a smart solution. ☺

FIREOFFSET
(float)x
(float)y
(float)z

  - Used for allowing multiple “barrels”, for example a Tank with a “rocket pack” where the rockets are to be fired from different barrels. An artist would typically create a number of different animations, where the muzzle flare would be at different positions in each. The fire-offset animations will be cycled in the specified order for each projectile fired.
  - Parameters are relative Entity BARREL, TURRET, or ENTITY animation local space pivot point in meters (depending on the level defined).
  - Must include 3 coordinates X the number specified using the FIREOFFSETCOUNT tag (for example with FIREOFFSETCOUNT 3, FIREOFFSET parameters would be xyzxyzxyz).

FIREOFFSETCOUNT
(int)count

- Essentially the number of unique FIRE animations specified in the .MDL file for the Entity.
- You MUST include as many 3D coordinate triplets under the tag FIREOFFSET as you have specified in FIREOFFSETCOUNT, as well as defining FIRE animations in the .MDL file with the following syntax:
  - FIRE
  - FIRE2
  - FIRE3
  - FIRE4
  - …
  - FIREn, where n is equal to FIREOFFSETCOUNT

FLESHFLAG

- If this tag is specified, the Entity will be treated as a “fleshy” unit.
- Projectiles use this flag to determine the type of effect to spawn when an Entity is hit.
- Flesh Entities do not suffer armor penalties when being hit in the rear.
- Homing projectiles cannot target flesh Entities.
- Flesh Entities never brake - they stop immediately.
- Flesh Entities do not trigger mines.
- Probably more stuff…

FLYINGHEIGHT

(float)flyingheight

- Height above the ground for hovers or aerodynes (hover and air Movers).
- Hovers cannot fly over other units.
- The “swaying” of the hover mover is a client side effect that will not be affected by this value.

FORCE

(string)CRAYVEN/WRATH/PHOENIX

- The faction the entity belongs to. CRAYVEN is the Crayven Corporation, WRATH is the Order of the new dawn and PHOENIX is the Phoenix faction in Dark Conspiracy.

FRICTION

(float)friction

- Braking acceleration, applied every frame. Used to slow and stop the Entity.
- m/s/s.
- Must be lower than ACCELERATION.

HEADTURNSPEED

(float)yheadturnspeed
(float)zheadturnspeed

- YHeadTurnSpeed is rotation speed in radians / second around the Y-axis for the TURRET child, or for the entity if AIMERCONTROL is specified (see Marines or Jaegers).
- XHeadTurnSpeed is rotation speed in radians / second around the X-axis for the BARREL child, or for the TURRET child (if no BARREL is specified). If no TURRET or BARREL is used, this isn’t visibly apparent, but it will still aim using this speed.

HEALER

(float)speed
(float)range

- Healer data, used in GC for the Command APC and Deployable Repair Station.
- Speed is the healing speed measured in Entity power / second.
- Range is the range of the healer; all friendly Entities within range will be healed.
- “Friendly” Entities are all of the same team, all allied, and neutral.

HEIGHT

(float)height

- Height of the Entity on the Y-axis, in meters.
- Entities use bounding cylinders for collision detection.

MAXSPEED

(float)MaxSpeed

- Maximum entity speed, in meters per second.

MAXUNITS

(int)number of units

- Essentially the number of units per Entity type (for squads).

- Only relevant to INFANTRY, SUPPORT, TANK, and AERODYNE branches.
- The value can be 1 or more, up to 12 has been used. More may cause unwanted behavior.
- Dictates how many units can as a maximum be in a squad placed with GenEd, and how many units are in a squad given to the player at the start of the game.

MINIMAPIMPORTANCE
(int)importance

- This determines how the entity appears on the mini-map.
- 0 = NORMAL DOT.
- 1 = IMPORTANT DOT (used for Artillery in GC) is a colored dot with a white center.
- 2 = VERY IMPORTANT DOT (used for Command APC in GC) is a big colored dot.
- All negative values revert to 0, all values higher than 2 revert to 2.

MODEL
(string)entmodelfilename

- The main EntModel representation (.MDL) for the Entity (no path or extension, for example “building_cc_ammo_depot”. This .MDL MUST have certain animations defined, and a number of optional animations are also legal, as follows (see .MDL script language documentation for more information):
- REQUIRED ANIMATIONS:
  - STAND, for standing still, not moving.
  - WALK, for moving.
  - FIRE, for firing (this should be non-looping).
- OPTIONAL ANIMATIONS
  - FIREn (where x is 2 – number of fire offset specified in FIREOFFSETCOUNT).
  - UNHURT (triggered when Entity is healed / repaired above 50%).
  - HURT1 (triggered when Entity Hit Points are below 50%)
  - HURT2 (triggered when Entity Hit Points are below 25%)
    - HURT1 and HURT2 must be children to the main EntModel, which MUST NOT have STAND, WALK or FIRE animations defined.

MODNAME
(string)modificationname

This string was introduced in the commercial expansion/modification Dark Conspiracy and equals the name of the mod in which the entity is to be used. Its specifics are unknown to this author but experimentation is encouraged.

MOVERTYPE
(string)foot/air/vehicle/hover

- The type of Mover for the Entity, quite self-explanatory. These are case sensitive, must be lowercase. Sorry.

MULTIPLAYERSCOREPERENTITY
(int)score

- The number of points awarded to Score for a kill in multiplayer.
- This score is, as the tag name indicates, per entity, and not per squad.

NAME
(string)name

- Zero based integer localization index of the “pretty” name of the Entity (for example “Jaeger Infantry”).
- Localized names reside in Data\localization\entfiles.loc.

NODEFAULTDEATHEFFECT
- Specifying this tag will disable the default death effect when an Entity is destroyed.

- There is no default effect for FLESHFLAG entities.
- All other Entities are considered to be "made of metal", and will trigger a LargeExplosion and a number of Shrapnel effects. See DEATHCHUNKMODEL for more information.

NOTEAMMARKER

- This disables all markers for an Entity.
- No triangle, no ring, no health / power meter, no mini-map dot, no nothing.
- This is used in GC for the Easter egg animals, and certain Entities used in the tutorials.

PERCEPTION

(float)perception

- This value (in meters) counters an enemy Entity's Stealth. See STEALTH for more information.

PIVOTHEIGHT

(float)pivot height

- Negative height from the entity pivot point to the bottom of his bounding cylinder (see aerodynes for examples).

POWER

(int)power

- Hit points / power / health.
- Integer value, internally represented as a short, and should be between 1 and 32767.

RADARSIGHT

- An Entity with RADARSIGHT will see everything within its view range, regardless of Line-Of-Sight considerations (it will "see over mountains" so to speak").

PROJECTILEPROTECTION

(float)protection

- This is in essence the armor value for an Entity.
- The ingame formula is as follows:
  - Inflicted damage = (projectile damage – projectile protection) * projectile damage multiplier.
  - See the WeaponConf documentation for more information.

RADIUS

(float)radius

- Radius of the Entity in the XZ plane, in meters.
- Entities use bounding cylinders for collision detection.

SLOPEBORDER

(int)slope border

- Slope tolerance value, between 0 and 3.
- 0 is worst, 3 is the best.
- The value 3 is for aerodynes (can handle any slope, and fly over buildings).
- The value 4 has been used for aerodynes in GC, and is allowed, but there are technically only 4 levels (0 to 3).
- Infantry in GC have a value of 2, so they can handle the next toughest slopes, but not the red ones (as seen in GenEd).

SLOPEMODIFIER

(int)slope border
(float)x size
(float)height
(float)z size

- For GenEd, this is used to modify the Slope Map (which is initially generated from the mission Height Map), so that path finding will take certain Entities (for example buildings) into account. In GenEd, a square under the Entity is inserted into the Slope Map (with the value specified). The size of this square in X and Z is the size of the values specified rounded up to the nearest multiple of 6.25 meters, which is the resolution of the slope map.
- Don't use this for Entities that will move around, as the Slope Map will be modified at the point where the Entity is originally created. Only for nonmoving turrets and buildings.
- For GC, the slope value is ignored (as the Slope Map already has been modified), but the X size, height and Z size are used for generating a number of cylindrical collision objects to approximate the bounding box as specified by these three values.

STATICFIREDIR

(float)x
(float)y
(float)z

- This vector, defined in local Entity space, is used for a special firing behavior for Entities that are to fire in a predefined direction.
- Only works with Homing projectiles.
- In GC, this is only used for Rocket Vehicles.

STAYVISIBLEWHENSEEN

- Means that the Entity will remain visible on the Client once the players units have seen it (via LOS).
- Use this for buildings and nonmoving Entities that should be visible once seen.
- DO NOT use this for Fixed Guns, as the Entity will not receive any Entity updates unless it actually IS in LOS again.
- This means that turrets that are out of LOS and firing at something will probably have a bad firing angle (it will look wrong) since their barrels will be pointing the direction they were the last time they actually were in LOS. "StayVisibleWhenSeen" is a hack for use with BUILDINGS only, meaning Entities without Aimers and Weapons.

STEALTH

(float)stealth

- This value (in meters) effectively shortens an enemy Entity's view range, so that a stealthy Entity cannot be seen even though it is within the view range of the enemy (with a free Line Of Sight).
- The enemy Entity's PERCEPTION counters the effect of STEALTH, effectively increasing the view range again.
- An Entity with high PERCEPTION can never see farther than its own view range, but a high PERCEPTION value will enable it to detect enemies with high STEALTH values within it's own view range.

TURNOFFSET

(float)distance

- This is the distance an Entity must be from its intended target before turning towards it again.
- This is used for Aerodynes, to achieve the "fly by" behavior.

TURNSPEED

(float)turn speed

- Y rotation speed in radians / second.

USEAIMER

- Uses an Aimer, needed to target enemies.

VIEWRANGE

(float)view range

- View radius of the Entity, in meters, a perfect circle with the Entity in the center.

WRECKMODEL

(string)entmodelfilename

- An EntModel (.MDL) to be used for the “wreck” of an Entity. Any number is allowed, and is randomized upon Entity destruction. Repeat the tag to specify more than one.
- If the Entity uses an air Mover, the wreck will be spawned at the exact position of Entity destruction, otherwise it will be placed on the ground at the X and Z coordinates of unit destruction.
- Additional effects will be spawned upon unit destruction, unless the tag NODEFAULTDEATHEFFECT is specified.
- If the FLESHFLAG tag is specified, no default death effect will be spawned.
- If not, a LargeExplosion (hard coded) will be created, with a size relative to the RADIUS of the Entity, and a number of Shrapnel (hard coded) will also be create, the number relative to the RADIUS of the Entity.

SECTION 4 – CONSOLE MANUAL

Disclaimer

All information in this section is subject to change. Neither Massive Entertainment nor Sierra will be held responsible for program crashes due to console usage. Also, please do not ask for extensive bug fixes to console functionality, as it is primarily intended as a development and debugging tool.

Console Basics

The console has three modes, which are toggle by pressing the ~ key (English keyboard, top left of keyboard). Holding down M, S and V on the main menu of Ground Control™ and entering the keyword "Console" in the resulting input box activates the console mode.

No Output

Nothing is displayed.

Track

A limited number of output lines are displayed, and all variables in the track list are also added to output and displayed.

Input

Console output is full screen, but the track list is not output. A prompt is displayed, so the user can enter input. Command history (ala DosKey) is available with the ARROW UP key, and TAB completion is also supported.

Typing the name of a command with no parameters will simply display its values (if it is a variable), or execute it (if it is a function). To change a variable value, type it's name followed by the new value. Typically, functions need parameters, and all error handling, help output, etc is handled on a per function basis.

Debugging

For debugging purposes, all console output is logged in cm_debug.txt, which is output to the Data directory after each program execution. This file output is not buffered, so it should still be output even if the program crashes.

Console Variables and Functions

Types

All command strings recognized by the console are either variables in the program code, or functions that can be executed.

Some variables are both read and write variables, in that their values are both read and written by the program code, and thus changes made in the console will affect program behavior.

Other variables are write only, in that their values are never read by the program code, only written, and are in essence only exposed through the console for debug information.

Finally, other variables are read only, in that their values are never written by the program code, only read, and are in essence only exposed through the console for run time adjustments by the user.

Below, **var** specifies read/write variables, **var_w** specifies write only variables, **var_r** specifies read only variables, and **func** specifies functions. In addition, the data type of variables is also specified, and the types and number of parameters needed by functions. Please not that this information is only a hint, and may be slightly incorrect.

Example:

mousepos (var_w, Vector3f)

The command is named mousepos, it is a write only variable, and its type is a 3 dimensional floating-point vector.

IMPORTANT NOTE: Although variables are specified here as being read/write, write or read only, there is nothing to prevent the user from changing a write only variable and thereby potentially corrupting program state. In addition, it is always possible for the code to change values that are specified as read only variables. As the console is simply a debug and development tool, no steps are planned to prevent these kinds of errors, so proceed with caution.

Variable and Function List

App (global variables)

- **track (func, <command>)**

A special function that adds a console variable or function to the console track list. The track list is executed once every frame when the console is in Track mode, and can be used to track the current fps, number of projectiles, etc. BEWARE of using track with functions such as **shot**.

- **untrack (func, <command> or <no parameter>)**

Without parameters, this function will clear the track list. With a command parameter, the listed command will be removed from the track list, if present.

- **shot (function, <no parameter>)**

Write a screenshot (24 bit JPEG, current resolution) to the Data directory. Screenshots will automatically have successive filenames, and existing screenshots will not be overwritten.

CLSession (client variables)

- **fps (var_w, float <seconds>)**

Current frames per second. Use with **track** to constantly print fps.

- **cloudspeed (func, float <cloud speed>)**

The speed of the animated cloud map, in meters per second.

- **far_plane (func, float <distance>)**

Sets the far clipping plane distance (meters). This is the same value used for view distance and is measured in meters from the camera.

- **terrain_lod (func, int <mode>)**

The Level Of Detail mode used for the landscape. 0 = no LOD, 1 = Morph, 2 = Snap. No longer used by GC since the ground detail code was changed during final development and the terrain will always morph to the right detail level. Controlled now instead by the terrain detail slider.

- **fog_enable (var_r, int <enable>)**
- **unit_tracks_enable (func, int <enable>, saved)**
- **unit_particles_enable (func, int <enable>, saved)**
- **unit_shadows_enable (func, int <enable>, saved)**
- **particles_enable (func, int <enable>, saved)**
- **lens_flare_enable (func, int <enable>, saved)**
- **bump_map_enable (func, int <enable>, saved)**
- **cloud_map_enable (func, int <enable>, saved)**

Various 3d engine toggles for turning effects on and off, basically the same controls available from the graphic options menu.

- **vid_texel_center, var, float <alignment>**

The current texture center alignment, the same value set from the graphical options menu. Also set individually on a per-card basis.

- **lod_bias (func, float <?>, saved)**

This value will be multiplied with the distance from the camera that a unit enters it's lower detail modes (LOD for Level Of Detail). If this value is set lower than 1, 3D models will become less detailed closer to the camera than default, and if the value is set above 1, the units will have a higher detail at greater distance. Raising this value will result in higher complexity but should look better in higher resolutions.

- **linear_contrast (func, float <linear contrast>, saved)**
- **exponential_contrast (func, float <exponential contrast>, saved)**

Functions concerning the image contrast, both Linear and Exponential.

- **complete_mission (var, int <enable>)**

Flag the current mission as completed in the current Player structure. Used for campaign debugging. Set to one and then end the mission to get a success and be able to continue in the campaign.

- **//variables**
- **num_entities (var, int <enable>)**

A debug track of the number of entities present on the Client. BEWARE of changing this value, it will most surely corrupt program state.

- **movement_smoothing (var_r, float <factor>)**

Used in CL_Entity (Client Entity) interpolation.

- **//sound variables**
- **sound (var_r, int <enable>)**

A nonzero value (default) will enable the sound system. A zero value will disable it.

- **sound_ speed_shift (var_r, float <factor>, saved)**

The factor with which increasing entity speed affects the frequency of their ambient/engine sounds.

- **sound_doppler_shift (var_r, float <factor>, saved)**

The factor with which negative speed deltas between the camera and an entity affect the frequency of their ambient/engine sounds.

- **sfx_volume, (var, int <volume>)**

The volume of sound effects.

- **num_playing_ds_buffers, (var_w, int <buffers>)**

The current number of active DirectSound buffers.

IngameInterface (client variables)

- **fov (func, float <field of view>)**

Sets the current field of view. Please note that the mouse pointer is no longer tracked to the correct terrain coordinates if the FOV is changed.

- **soundreport (func, <no parameter>**

Outputs the state of all statically loaded sounds, including their id’s, filenames (if loaded), and whether they use hardware or software mixing.

- **filter_color (func, byte <red>, byte <green>, byte <blue>)**

Function concerning color blending. Currently disabled.

Variables

- **mousepos (var_w, Vector3f <position>)**

The world position of the mouse. Use with **track**.

- **debugmouse (var, int <enable>)**

A nonzero value will enable a particle line extending from the world position of the mouse 10 meters up into the air.

- **camerapos (var_w, Vector3f <position>)**

The world position of the camera. Use with **track**.

- **drawslopemap (var_r, int <enable>)**

A nonzero value will enable a limited particle line grid which shows the slope values of the area around the mouse cursors world position.

- **leftclickmovement, (var, int <enable>, saved)**
- **leftclickattack, (var, int <enable>, saved)**

Controls whether or not the left mouse button should be used for attacking and/or moving, or if it should be for selecting only (both disabled). Default is 1 for both, enabled.

Control sensitivity

- **i_sensitivity (var, float <sensitivity>, saved)**

The in-game mouse sensitivity.

- **m_sens_x (var_r, float <sensitivity>, saved)**
- **m_sens_y (var_r, float <sensitivity>, saved)**

Sets the mouse sensitivities in each axis when using mouse look. (Currently disabled.)

-
- **e_sens_x (var_r, float <sensitivity>, saved)**

- **e_sens_y (var_r, float <sensitivity>, saved)**

The mouse rotation sensitivities, used when edge rotating (mouse at screen edge). I’m not sure what units these values use.

Interface debug

- **ig_state_info, (var_r, int <enable>)**

If set, types an info string in the upper left corner signifying what part of the Ingame Interface is used.

Ingame entity marker options

- **marker_squad (var_r, int <enable>)**

A nonzero value will enable squad information for the select entity. Default enabled.

- **marker_name (var_r, int <enable>)**

A nonzero value will enable entity name information for the select entity. Default enabled.

- **marker_range (var_r, int <enable>)**

A nonzero value will enable range information from the tracked entity to the select entity. Currently not used in the code.

- **marker_pmeter (var_r, int <enable>)**

A nonzero value will enable the power meter for the select entity and damage notifiers. Default enabled.

- **marker_pwidth (var_r, float <width>)**

The width (in pixels) of the entity/damage notifier power meter, if enabled. Default 16.

- **marker_transparency (var_r, float <alpha>)**

The alpha value of the entity/damage notifier power meter, if enabled. Default 0.75.

- **marker_pnumber (var_r, int <enable>)**

A nonzero value will enable power number information for the select entity. Default disabled.

- **marker_speed (var_r, int <enable>)**

A nonzero value will enable both Client and Server speed information for the select entity. Default disabled, and will be removed, is it violates the network paradigm. May crash the program if enabled for a non-hosting multiplayer Client.

- **marker_position (var_r, int <enable>)**

A nonzero value will enable position information for the select entity. Default disabled.

- **marker_protection (var_r, int <enable>)**

A nonzero value will enable Server protection information for the select entity. Default disabled, and will be removed, is it violates the network paradigm. May crash the program if enabled for a non-hosting multiplayer Client.

- **marker_destination (var_r, int <enable>)**

A nonzero value will enable Server Mover destination information for the select entity. Default disabled, and will be removed, is it violates the network paradigm. May crash the program if enabled for a non-hosting multiplayer Client.

- **marker_entitydata (var_r, int <enable>)**

A nonzero value will enable entity data (.ent file name) information for the select entity. Default disabled.

- **marker_entitydata_child (var_r, int <enable>)**

A nonzero value will enable entity child information for the select entity. Default disabled.

- **marker_ambientsound (var_r, int <enable>)**

A nonzero value will enable ambient sound (”engine” sound) information for the select entity. Default disabled.

- **marker_transparency (var_r, float <alpha>)**

The alpha value for team markers (the unit triangles). Default is 0.7.

- **entitylist (var_r, int <enable>)**

A nonzero value will enable drawing of a team coded entity list at the top of the screen.

- **messagebox_timer (var_r, int <enable>)**

A nonzero value will enable a numerical timer display on message boxes in timer mode.

- **minimap_dots (var_r, int <enable>)**

A zero value will disabled all dots in the mini-map.

In-game markers

- **gm_heightoffset (var_r, float <height>)**

The offset above the ground, in meters, that the circular ground entity markers use.

- **gm_radiusscale (var_r, float <factor>)**

Circular ground entity markers are scaled by the entity's radius, multiplied by this factor.

- **aem_radiusscale (var_r, float <factor>)**

Circular air entity markers are scaled by the entity's radius, multiplied by this factor.

Limit drawing of certain components

- **ghostbutton_limits (var_r, Boolean <0/1>, saved)**

If set to 1 the game will draw a white box signifying the bounding space for the specials' buttons.

- **selectdebug (var_r, int <enable>)**

A nonzero value will enable drawing of the screen bounding-box used for mouse to entity selection checks.

- **chargebutton_radius (var_r, float <radius>, saved)**

This value controls the radius of the "circle" over which the "dots" signifying charges for special weapon and equipment are drawn. Lowering the value places the charge-dots closer to the button, heightening it places the dots farther from the button.

- **chargebutton_angledelta (var_r, float <radius>, saved)**

This value controls the radius of the interval at which charge-"dots" are drawn. Lowering the value places the charge-dots closer together. Heightening it places the dots farther apart.

SV_Session (server vars)

- **gravity (var, float <meters/second squared>)**

The gravity level. Used for physics (bouncing tanks, metal junk and ballistic weapons).

- **losupdate (var_r, int <enable>)**

Line-Of-Sight updating. A nonzero value (the default) will limit entity updates and visibility data sent to those entities that are actually in LOS of each respective Client's units. A zero value will send ALL entity update and visibility data to ALL Clients. Note that if **losupdate** is toggled once, all entities with the StayVisibleWhenSeenFlag will remain visible, even thought **losupdate** has been turned on again. Can only be used on the server computer such as in a single-player game or the LAN host.

AI variables

- **ai (var_r, int <enable>)**

A nonzero value (default) enables Server AI updates. A zero value disables them.

- **updates_per_frame, (var, int)**

The number of AI updates performed per frame.

- **update_frequency, (var, float <frequency>)**

The frequency of which AI updates are made.

- **updatemode, (var, int <mode>**

Some flag for setting the way AI updates are performed.

- **time_per_update, (var, int <time>)**

The elapsed time per AI update.

- **unused_time, (var, int <time>)**

The amount of AI processing time left unused, normally 0.

SECTION 5 – CREATING SKY MODELS

The skies in Ground Control™ are made in two different versions - one high-resolution sky and one low-resolution sky. Both models must have the same name, with an “ _h” appended to the high-res sky model filename.

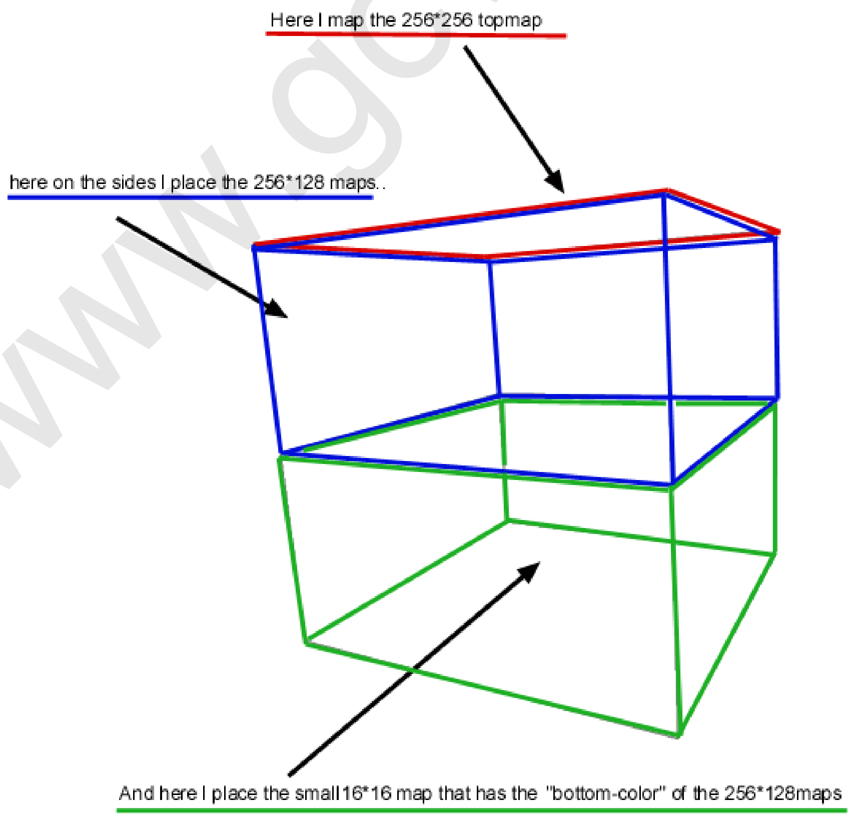
High Resolution Sky Models

The high-resolution sky model is basically a sliced box (10 surfaces) with texture maps.

1. In Lightwave™ (or another 3d program) build a scene with beautiful clouds that stretches to the horizon. Place the camera in x 0 y 0 z 0 (right in the middle of the scene). Rotate the camera 90 degrees at a time and render a shot with every new turn. When you have completed a full circle, pitch the camera 90 degrees upward and render the sky above the camera. Easy, right. =)

2. Make sure that you have the zoom level correctly set on the camera in order to make it overlap the different “wall” images correctly. Use zoom level 1 in Lightwave, when turning the camera in 90 degrees increments it overlaps perfectly. Set the image sizes to 128*128, 256*258 or 512*512 etc. You might have to experiment a bit to get it look right.

3. Render the images in 256*256 pixels resolution. The top image (on the Y axis) should be 256*256, but the image maps on the side “walls” of the box should be cut down to 256*128 in Photoshop (to save texture space). Then pick out the “bottom” color of the wall image map and use it to make a tiny 16*16 map (the fog map) and apply it on the bottom surface. The bottom surface should have the same color as the fog set in GenEd. Look at the illustration. Observe that the “Surface vertices” should be facing inwards into the box. Make sure to match the fog color of the “game map” to the fog color of the sky model.


Here I map the 256*256 topmap
here on the sides I place the 256*128 maps..
And here I place the small16*16 map that has the "bottom-color" of the 256*128maps

4.

5. It is important that you scale down the image maps one pixel. For an example, if you take the 256*256 map, you shrink it one pixel on the sides, so you end up with a one pixels wide border around the map. The map is still 256*256 pixels wide, but the motif is shrunk to 254*254. In Lightwave™ you have to size up the texture values equal to the width of two pixels in when you map it on the box so you don’t see the “border” you made earlier. Why do this? Because when wrapping, the pixels to the far right of the texture are bilinear filtered and blurred together with those to the left (or up/down) creating a seam in the sky model. By doing these operations you move the artifacts out of the visible part of the surface.

Low Resolution Sky Models

6. The low-resolution models have much less texture complexity, only 3 textures. One Small cloud map (128*128 is good). One sky texture (16*16, one color that is on the top of the model) And one fog color texture map that are mapped in the lowest part of the model. As always, make sure to match the fog color of the “game map” to the fog color of the sky model. The low-res and high-resolution version of the same sky model should have the same fog color so they both match the fog on the “game map”.

7. You also have to edit the Skies.txt file, which you’ll find in your Data\GenEdData\ directory. Write down the Filename of the model and the fog color. Note that you should only write the name of the “low-res” sky here. Look at the Skies.TXT file, and you will understand. Easy. =) Now you can select your new sky in GenEd when building a new map.

SECTION 6 – AUDIO & MUSIC

This document will briefly explain the directories located under the Data\Sound directory one by one. It will also explain what sounds they contain, how they are used, why, what, when etc.

Note! Enumerated files are expressed as "**File*[1-3]*.wav**" in this document. Please do not interpret this as a file by the name "File[1-3].wav" but instead three files by the names "File1.wav", "File2.wav" and "File1.wav".

Introduction

These are the sound files used in briefings. They are not embedded in the code, which means you can have any number of files with whatever filename desired. Playback is scripted in a mission briefing file (.ms2) located under data\missions\mission[x] (see existing .ms2-files for example).

All samples must be encoded to Mpeg Layer 3 and then converted to MPD with "Mp3ToMpd.exe" which adds info about sample length to the file header.

Ingame\EntityAmbient

Contains ambient looped samples such as engine sounds for hovers, rockets, tanks, bullets and any kind of building (as long as there exists an ent-file for the object). The sound is specified in the ent-file for that particular vehicle/building. It will loop until the entity is destroyed.

AmbientBuilding[1-7].wav
Used for various buildings and other objects within the game.

BallisticEngine1.wav
The sound accompanying a ballistic projectile rising and falling back to the ground.

BulletEngine1.wav
The whistle of a bullet flying past.

DroneEngine1.wav
The clockwork mechanics of a homing drone from the drone carrier.

DropBombEngine1.wav
The sound of bombs falling from the bomber aerodyne.

EasterEgg_BirdSwarm.wav
A flock of birds, whistling softly.

EasterEgg_Frog.wav
The tank-sized alien cow-frog slowly dragging itself across the green plains.

EasterEgg_Taller.wav
The sound of the tall two-legged flock-animals running around the deserts of Krig-7B.

EngineAir[1-2].wav
Samples used for anything flying higher than Hoverdynes.

EngineDropship.wav
Sample used for the drop ship.

EngineHover[1-2].wav
Two variations of the OND Hoverdyne engine.

EngineVehicle[1-3]wav
Made with an almost "normal" tank in mind. These two samples are used for Terradynes, APCs and several other vehicles trekking the ground plane.

HomingEngine1.wav
The sound of a homing bullet-mover moving.

RocketEngine1.wav
The sound of a rocket mover moving.

Turret1.wav
Used specifically for the engine of those hostile turrets (Savior, T-7 etc). I am uncertain if GC is currently using this sound at all.

[Example taken from cc_heavytank.ent]

```
//engine/ambient sound
//valid strings are:
//snd_engine_dropship, snd_engine_hover1-2, snd_engine_vehicle1-3,
//snd_engine_air1-2, snd_ambient_building1, snd_ambient_building2,
//snd_ambient_building3, snd_ambient_building4,
//snd_ambient_building5, snd_ambient_building6, snd_ambient_building7
AmbientSound
snd_engine_vehicle3
```

Because of RAM limitations you have to make as much as possible out of the few samples loaded. Therefore there is also the possibility of specifying an individual base frequency, i.e. at what pitch will the sample be played back. We used exclusively 16bit/22050Hz for our samples. If we then specified a base frequency of 11025, that particular sample would be played back at half speed (an octave lower).

Example:

[Taken from Building_Wc_Reasearch_Center.ent]

```
//engine/ambient sound
//valid strings are:
//snd_engine_dropship, snd_engine_hover1-2, snd_engine_vehicle1-3,
//snd_engine_air1-2, snd_ambient_building1, snd_ambient_building2,
//snd_ambient_building3, snd_ambient_building4,
//snd_ambient_building5, snd_ambient_building6, snd_ambient_building7
AmbientSound
snd_ambient_building3

//engine/ambient sound base frequency (kHz)
//default is wav file default
AmbientSoundBaseFrequency
18000
```

Ingame\Feedback\Crayven & Ingame\Feedback\OND

This folder contains the voices of Annie-1 and Ghost. They are all hard coded. In GC version 1.0.1.0 and later these will have to have a maximum length of 4 seconds (in earlier versions it was 3 seconds).

AlliedAerodyneDestroyed.wav

AlliedHoverdyneDestroyed.wav

AlliedInfantryDestroyed.wav

AlliedStructureDestroyed.wav

AlliedTerradyneDestroyed.wav

EnemyAerodyneDestroyed.wav

EnemyHoverdyneDestroyed.wav

EnemyInfantryDestroyed.wav

EnemyStructureDestroyed.wav

EnemyTerradyneDestroyed.wav

CommandAPCDestroyed.wav

CommandAPCRecharging.wav

CommandAPCTakingFriendlyFire.wav

EnemyAttackingCommandApc.wav

Squad[1-14]Destroyed.wav

Squad[1-14]Recharging.wav

Squad[1-14]TakingFriendlyFire.wav

EnemyAttackingSquad[1-14].wav

UnitLostSquad[1-14].wav

DropzoneOccupied.wav

NewDropzoneAvailable.wav

MissionAccomplished.wav

MissionFailed.wav

Ingame\Interface

Sounds used by the ingame interface. All hard coded but they can of course be exchanged for different sounds.

Chat1.wav

CompleteObjective1.wav

EnableChat1.wav

IncomingChat1.wav

Invalid1.wav

NewObjective1.wav

NotReady1.wav

ObjectiveInfo1.wav

Ready1.wav

Use1.wav

Ingame\Multiplayer\Feedback

Contains the voice of the multiplayer “Game Master”, Annie Eckles. All hard coded. Max. length 4 seconds.

ReinforcementsAvailableSq[01-12].wav

Team[0-7]ClaimedFlagzone.wav

Team[0-7]LostFlagzone.wav

Team[0-7]Scored.wav

YourTeamClaimedFlagzone.wav

YourTeamLostFlagzone.wav

YourTeamScored.wav

Ingame\Multiplayer\Zoneambient

Two short loops used for the rotating objects in Flag Zones and Score Zones. Hard coded.

FlagZone.wav

ScoreZone.wav

Ingame\Noise

Contains explosions, all kinds of hits etc. These are (with exception of those hard coded) specified in Data\ManagementData\EffectSounds.txt. There are several variations of each sample (EnergyHit[1-4], HumanHit[1-3] etc.). When played back, the sound engine will randomize between all variations.

GroundHit*1-4*.wav
For bullets hitting ground.

EnergyHit*1-4*.wav
For energy weapon hits.

HumanHit*1-3*.wav
For flesh being hit.

MetalHit*1-6*.wav
For metal striking metal.

HumanKill*1-3*.wav
For flesh being killed

Junk*1-4*.wav
For the flying pieces of junk from explosions.

LargeExplosion*1-3*.wav

MediumExplosion*1-3*.wav

SmallExplosion*1-2*.wav

Rather self-explanatory, wouldn’t you agree?

Ingame\SqLeader

Contains both Crayven and OND squad leader feedback. All hard coded and specified in the .ent-file. The sound banks have the following sounds (except APCLoad which is only for infantry units).

APCLoad.wav

AttackModeFreeFire.wav

AttackModeHoldFire.wav

AttackModeReturnFire.wav

AttackYes*1-3*.wav

AttackNo*1-3*.wav

AttackStackYes.wav

AttackStackNo.wav

MoveYes*1-3*.wav

MoveNo*1-3*.wav

MoveStackYes.wav

MoveStackNo.wav

ForceFireYes.wav

ForceFireNo.wav

MovementModeDefensive.wav

MovementModeOffensive.wav

MovementModeHold.wav

FormationBox.wav

FormationColumn.wav

FormationLine.wav

SpecialWeaponYes.wav

SpecialWeaponNo.wav

SpecialEquipmentYes.wav

SpecialEquipmentNo.wav

Example:

[This is what Cc_Marine.ent used to look like]

```
//sound bank
//valid banks are:
//CRAYVEN_ANTIAIRVEHICLE
//CRAYVEN_ARTILLERYVEHICLE
//CRAYVEN_ATTACKAERODYNE
//CRAYVEN_BOMBERAERODYNE
//CRAYVEN_COMMANDAPC
//CRAYVEN_ENGINEERTRUCKER
//CRAYVEN_FIGHTERAERODYNE
//CRAYVEN_HEAVYTANK
//CRAYVEN_JAEGER
//CRAYVEN_LIGHTTANK
//CRAYVEN_MARINE
//CRAYVEN_MEDIUMTANK
//CRAYVEN_ROCKETVEHICLE
//CRAYVEN_SCOUTAERODYNE
//CRAYVEN_SCOUTVEHICLE
//OND_ANTIAIRVEHICLE
```

```
//OND_ARTILLERYVEHICLE
//OND_ATTACKAERODYNE
//OND_BEAMPLATFORM
//OND_COMMANDAPC
//OND_CRUSADER
//OND_DRONECARRIER
//OND_ENGINEERTRUCKER
//OND_FIGHTERAERODYNE
//OND_HEAVYHOVERDYNE
//OND_LIGHTHOVERDYNE
//OND_MEDIUMHOVERDYNE
//OND_SCOUTAERODYNE
//OND_SCOUTVEHICLE
//OND_TEMPLAR
SoundBank
CRAYVEN_MARINE

 [Here we have connected an ambient sound to the marine entity]
//engine/ambient sound
//valid strings are:
//snd_engine_dropship, snd_engine_hover, snd_engine_vehicle,
//snd_engine_air, snd_ambient_building1, snd_ambient_building2,
//snd_ambient_building3, snd_ambient_building4,
//snd_ambient_building5, snd_ambient_building6, snd_ambient_building7
AmbientSound
snd_ambient_building7
```

Ingame\Weapon

Weapon sounds. These are played whenever a weapon is fired. All of these are specified in ManagementData\WeaponSounds.txt. Some samples have variations for random playback. These are the file names to use:

DropBomb1.wav

Energy1.wav

Energy2.wav

Energy3.wav

Grenade1.wav

Grenade2.wav

Gun1.wav

Gun2.wav

Gun3.wav

MachineGun1_1.wav

MachineGun1_2.wav

MachineGun1_3.wav

MachineGun2_1.wav

MachineGun2_2.wav

MachineGun2_3.wav

MachineGun3_1.wav

MachineGun3_2.wav

MachineGun3_3.wav

The sounds that are post-fixed by an extra number after an underscore will play randomly. If a weapon is specified to use MachineGun3, the actual sound played will be one of _1, _2 and _3 that is played. This is because fast, repetitive sounds are much better when varied slightly.

Rocket1.wav

Rocket2.wav

Rocket3.wav

Ingame\FalloffDistances.txt

In this file you can specify individual falloff distances for explosions, ambient sounds etc. This is very useful when trying to create a believable environment.

Journal

These are the sound files used in journal entries. They are not hard coded, which means you can have any number of files with whatever filename desired. Playback is scripted in a mission journal file (.dsc) located under data\missions\mission[x] (see existing .dsc-files for example).

All samples must be encoded to Mpeg Layer 3 and then converted to MPD with “Mp3ToMpd.exe”.

Loading

This directory was never used.

Management

Contains the music played in the management interface, such as MainTheme, CrayvenTheme, MissionFailure/Success etc. This music is all hard coded. Here you will also find the button click sounds and mouse-over sounds, which are scripted in the files located under data\dialogs. These are not hard coded and can have any name or length.

ThemeMain.wav
This is the music you’ll hear when you first start the game. This theme will play until you’ve selected to play for either Crayven or OND. The purpose of it is to bring an over all space-age-military feel together with the movie with Astrid in the background.

ThemeCrayven.wav
This is the Crayven theme. The theme will play when you select campaign one (Crayven) in campaign mode or when you play any Crayven mission in single mission mode.

ThemeWrath.wav
This is the OND theme. The theme will play when you select campaign two (OND) in campaign mode or when you play any OND mission in single mission mode.

ThemeDebrief.wav
This theme will play during the debriefing, i.e. after a successful mission.

ThemeDefeat.wav
This will play whenever your mission is a failure.

ThemeCredits.wav
This theme is to be played in the credits screen.

ButtonPress*1-4*.wav
The buttons in the Management Interface are put into four different types/classes. Each sample is assigned to a certain type of button to make them more recognizable.

MouseOver.wav
Triggered when you move your mouse over a button.

MouseOver2.wav
Triggered when you move your mouse over a button.

Error.wav
Triggered when the user tries to confirm an illegal operation.

Management\CrayvenFeedback & Management\ONDFeedback

Samples played back when turning on/off the auto load function in the drop ship load-out screen. They are hard coded and must be in MPD format.

AUTOLOADCOMPLETE.mpd

AUTOLOADDISABLED.mpd

Soundtrack

Here you’ll find the music and ambient tracks used in game. They are all specified in the mission script file. You can change dynamically between tracks during a mission. This is done in the mission script based on different conditions such as player entering an area, mission objectives, timers etc. Have to be MPD. Not hard coded. Musical scores can be easily added here and then used in new missions.

All the of the tracks are designed for different missions with different environmental preferences, i.e. snow, sand, day, night etc. The mission designer selects the soundtrack for a certain mission after his or her own liking. They can be accessed from within GenEd using the SoundTrack action, which can also be used to change soundtracks and will cross-fade between the two.

AmbGrassLands.mpd
Made with wide, subtropical grasslands in mind.

AmbWinter.mpd
You guessed it - this one is for the missions filled with snow.

AmbWinterDisturbance.mpd
A snowy soundtrack with an eerie electrical disturbance ever-present in the air. Used on Crayven mission 13 where a jamming device is the objective. Also used on the Heavy Metal 2 multiplayer map with a metallic look.

AmbJungleNight.mpd
If there’s lots of big trees and green stuff around and it’s nighttime, this is probably what you will be listening to.

AmbJungleDay.mpd
It’s still green but it’s noon.

AmbDesertNight.mpd
Big, wide, boring desert at night.

AmbDesertDay.mpd
Same boring desert, only now in daytime.

TRACK[1-8].mpd
Eight musical soundtracks used throughout the campaign at certain events.

Voice

This directory contains all the mission specific voices. This can be mission updates or any other voice you want to play back during a mission. They are specified in the mission script file. Have to be MPD. Not hard coded. New messages can be easily added to this directory and used in missions.

All of the MPD files in the directory are called from the specific mission’s script-file, using the command MessageBox. Any file placed here can be called that way at any time during the mission.

SECTION 7 – MDL SCRIPT LANGUAGE

Introduction

Genesis 3D-engine uses a proprietary file format (MDL files) to describe the object types that should be possible to instantiate during a game. This document describes the syntax of the MDL scripting language, and how the files are used from the program.

MDL files normally reside in the directory EntModels, but can be placed anywhere. The path to a MDL file could be given relative the working directory, or specified in full.

MDL files are normal ASCII text files that can be edited in any standard text editor. The language has a simple block-oriented syntax that closely mirrors the hierarchical data structures built when parsing the file. The language is not case sensitive.

C++ style comments can be used. Everything between /* and */ is ignored. In contrast to C++, such comments can be nested. Two slashes // starts a comment that runs to the end of line.

All keywords and parameters are white space delimited. There are no literal string delimiters “” in the language, strings are also white space delimited. This means that e.g. filenames cannot contain spaces.

The “atoms” of the language are individual model files, typically created by Lightwave™ (*.lwo files). The purpose of the MDL language is not to define individual objects, but describing the logical relations between such models.

Basic MDL syntax

Below is an example of the simplest possible MDL file:

```
BEGIN_ENTITY EntName
  BEGIN_ANIMATION Stand
    BEGIN_LOD –1
      FRAME models/filename.lwo
    END_LOD
  END_ANIMATION
END_ENTITY
```

As can be seen, the language syntax describes a nested structure of BEGIN / END blocks. The indentation is purely for readability and not essential for correct parsing.

The three mandatory block levels are ENTITY, ANIMATION and LOD. BEGIN_ENTITY and BEGIN_ANIMATION must be followed by a name. Top-level Entity names should be unique across all loaded MDL files, and Animation names should be unique within an Entity. If not, some operations may cause undefined results.

BEGIN_LOD should be followed by a distance value. A negative value is interpreted as infinity. If several LODs (Level of Detail) are defined, they should be specified with increasing distance:

```
BEGIN_ENTITY EntName
  BEGIN_ANIMATION Stand
    BEGIN_LOD 10
            // high detail version visible to 10m
      FRAME models/filename_L1.lwo
      SHADOW_FRAME models/shadow_L1.lwo
    END_LOD
    BEGIN_LOD 50
    // medium detail version, visible to 50m
      FRAME models/filename_L2.lwo
      SHADOW_FRAME models/shadow_L2.lwo
    END_LOD
    BEGIN_LOD –1
    // low detail version, visible to infinity
      FRAME models/filenameL_3.lwo
    END_LOD
  END_ANIMATION
END_ENTITY
```

FRAME specifies the filename of a Lightwave™ LWO file to be used in the specified view range.

SHADOW_FRAME specifies the name of a model file containing a shadow model. If defined, this model will be treated specially by the system. The file should contain polygons that represent an outline approximation of the model projected to the horizontal plane. When rendered, these polygons will be rendered on top of the ground directly underneath the model. The polygons should not overlap.

As the example shows, shadow models are not mandatory, the most distant LOD does not define a shadow. For shadows to work correctly, the application must load a World object that provides a function for looking up the elevation value for any given ground coordinates.

It should be noted that if the same LWO file were specified multiple times – in the same or different MDL files – it would only be loaded once and shared among EntModels and LODs.

Animation States

The next step up the complexity ladder is to define several animation states for an object:

```
BEGIN_ENTITY EntName

  BEGIN_ANIMATION Stand
    BEGIN_LOD –1
      FRAME models/filename_stand.lwo
    END_LOD
  END_ANIMATION

  BEGIN_ANIMATION Walk
    ANIMATION_TIME 2
    LOOPED ON
    BEGIN_LOD –1
      FRAME models/filename_L-1_Walk_Frame1.lwo
      FRAME models/filename_L-1_Walk_Frame2.lwo
      FRAME models/filename_L-1_Walk_Frame3.lwo
      FRAME models/filename_L-1_Walk_Frame4.lwo
      TEXTURE_FRAME models/filename_txa.lwo
    END_LOD
  END_ANIMATION

  BEGIN_ANIMATION Dead
  END_ANIMATION

END_ENTITY
```

Two things are worth noting. First, there can be any number of frames within an animation state. The first “Stand” animation is a static state consisting of a single frame similar to earlier examples. The second state “Walk” consists of four different frames. The LOOPED ON keywords will make the animation repeat itself, and ANIMATION_TIME will specify how long time one repetition will take. The “Dead” state is defined, but is empty. This means that the object will respond to requests to enter state “Dead” by becoming invisible.

Secondly, this example shows the use of a special texture alignment frame. LWO files do not contain texture coordinates for individual vertices, only a projection formula. To overcome this restriction, the artist can create a texture alignment object that “moves the vertices to the texture” so to say. When loading frames for an animation state where a TEXTURE_FRAME is defined, the texture projection is applied to the texture alignment object, and the texture coordinates so achieved will then be copied to the corresponding vertices of all other frames. Creating a working texture alignment object is a delicate task that takes some experience. Since all vertices and polygons must be defined in the same order in all files using a TEXTURE_FRAME, they must all be created from the texture alignment object – or vice versa - by *moving vertices only!* Adding or deleting vertices or polygons will ruin the attempt.

A third party Quake™ web page at http://www.cschell.com/tutorials.html?skinmaxtut covers how this is done on Quake™ models, which may or may not be helpful when trying to do this in Ground Control™. Thank you to TomBob for pointing this out.

Animations and LODs in combination

Within an animation state, the number and sequence of frames may be different for different levels of detail (LODs):

```
  BEGIN_ANIMATION Walk
    ANIMATION_TIME 2
    LOOPED ON
    BEGIN_LOD 50
      FRAME models/filename_walk1_L1.lwo
      FRAME models/filename_walk2_L1.lwo
      FRAME models/filename_walk3_L1.lwo
      FRAME models/filename_walk4_L1.lwo
      TEXTURE_FRAME models/filename_txa_L1.lwo
    END_LOD
    BEGIN_LOD –1
      FRAME models/filename_walk1_L2.lwo
      FRAME models/filename_walk2_L2.lwo
      TEXTURE_FRAME models/filename_txa_L2.lwo
    END_LOD
  END_ANIMATION
```

In this case, ANIMATION_TIME will ensure that the cycle time is the same regardless of the number of frames.

Hierarchical Parent – Child Relations

An EntModel can contain child objects positioned relative the parent space. There are two ways to do this; by recursively define the child inline, or by referencing another MDL file. Both versions are shown below:

```
BEGIN_ENTITY Tank
  BEGIN_ANIMATION Stand
    BEGIN_LOD –1
      FRAME models/tank_stand.lwo
    END_LOD
  END_ANIMATION
  BEGIN_ANIMATION Dead
    BEGIN_LOD –1
          FRAME models/tank_wreck.lwo
    END_LOD
  END_ANIMATION

  BEGIN_CHILD
    POSITION 0.0 2.0 0.0
    BEGIN_ENTITY Turret
      BEGIN_ANIMATION Stand
        BEGIN_LOD –1
          FRAME models/turret_stand.lwo
        END_LOD
      END_ANIMATION

      BEGIN_ANIMATION Fire
        ANIMATION_TIME 0.3
        LOOPED OFF
        BEGIN_LOD –1
          FRAME models/turret_fire_1.lwo
          FRAME models/turret_fire_2.lwo
        END_LOD
      END_ANIMATION

      BEGIN_ANIMATION Dead
      END_ANIMATION
              BEGIN_CHILD Barrel
                POSITION 0.0 0.0 1.0
                FILE entmodels/tank_barrel.mdl
              END_CHILD
    END_ENTITY
  END_CHILD
END_ENTITY
```

The first part of the *Tank* Entity is similar to earlier examples. Two animation states are defined, one showing the normal tank model, and a *Dead* state showing a wreck. After the animation definitions – although order is not important – comes a BEGIN_CHILD / END_CHILD block. In this case, the *Turret* entity is defined inline. Turret defines three states, *Stand*, *Fire* and *Dead*. Fire is animated, this time with LOOPED OFF. The state Dead is defined but empty which ensures that the turret will disappear when the hierarchy as a whole is put in state Dead. The Fire state is undefined in the parent entity Tank, which means that it will remain in whatever states it is in when the hierarchy as a whole is set to state Fire.

The Turret entity defines a grandchild called *Barrel*. This time the alternative syntax is shown, referencing an entity definition stored in a separate MDL file through the use of the FILE keyword.

Worth noticing here also is the fact that the **DEAD** animation state is not no longer necessarily defined from the MDL file, but can instead consist of one or several different MDL files defined in the ENT file for the entity using the keywords **WreckModel**

Terminating and Changing Animation States

An animation state with LOOPED OFF will be considered *expired* after the number of seconds specified by ANIMATION_TIME, i.e. after one repeat of the animation. A state with LOOPED ON will not expire unless the keyword TIME_LIMIT is also used. In this case, ANIMATION_TIME controls how fast the animation proceeds, and TIME_LIMIT controls when the animation should be considered expired.

Several things could happen when a state expires. The application can as mentioned above explicitly specify the next state by calling `instance->SetNextAnimation("AnimationName");`
This has the highest priority and overrides directives in the MDL file.

The keyword NEXT_ANIMATION followed by a state name can be used in Animation blocks. When the animation expires, the specified state will be entered provided that the application has not issued an overriding command.

Advanced Control and Geometric Animation

A number of keywords exist for controlling an instance’s position and movement relative its parent. Several of them can be defined both in ENTITY and ANIMATION blocks. Values defined on the Animation level is always used when an instance enters that animation state. If specified, values defined on the Entity level are used whenever an object is in a state that lacks own definitions.

Keywords that can be used on both the Entity and Animation levels are:

```
POSITION    x y z         // meters from parent’s origin
ORIENTATION   heading pitch roll   // degrees in parent space
VELOCITY    vx vy vz         // meters / second, local space
ROTATION    heading pitch roll   // degrees / second, local space
```

POSITION and ORIENTATION is used when an instance is created (if defined at the Entity level) or when the instance is put in a new animation state (if defined at the Animation level). Coordinates are measured in parent space. I.e. a positive z value is interpreted as “forward” in parent space, positive heading orients the object turned “right” around the parent’s y-axis.

POSITION and ORIENTATION defined at the Entity level is *only* used when the object is first created. If definitions exist in an Animation state, the instance is instantaneously “teleported” to this position and orientation whenever it enters the state. In contrast to this, a state without position and/or orientation defined will leave the object where it happens to be when the state is entered.

VELOCITY and ROTATION is measured in meters/sec and degrees/sec and can also be specified both individually for different states and also at the Entity level. The semantics are however a bit different. Defined at the Entity level, these values are not only used when the instance is first created, but for all animation states, that does not contain overriding values.

It is important to note that VELOCITY and ROTATION is specified in the instance’s *local body coordinates.* This means that a positive velocity z-value always means that the instance will move nose forward, regardless of current orientation. Similarly, rolling will always be around the local z-axis. This makes it easy to define combined motion. VELOCITY 0 0 5 combined with ROTATION 45 0 0 will for example make the instance run in circles, turning right while moving forward with 5 m/s.

By chaining animation states through the use of TIME_LIMIT and NEXT_ANIMATION, very complex motion is possible. A unit may move straight for a certain time, then start turning, etc. By *not* defining position or orientation in these states, the new velocity and rotation will start acting from whatever position and orientation the instance has when entering the new state. On the other hand, this will lead to accumulated errors if the chain of states is a continuous loop. In such a loop, at least one state should therefore contain POSITION and ORIENTATION keywords to “synchronize” the movement to parent space

The SPLINE Keyword

This is the most powerful way to animate an instance. It can be used both inside an Animation block and at the Entity level. Splines defined at the Entity level affects all animation states that does not define a spline themselves. The syntax is:

```
SPLINE  filename.mot  time
```

Optionally followed by LOOPED ON/OFF. Looped ON is default. Examples:

```
SPLINE  motionfile.mot  20
SPLINE  motionfile.mot  3.5 LOOPED OFF
```

The motion file is in the Lightwave™ MOT format. Such files can be created in the Lightwave™ Animator application program.

Not all information in this file format is yet used. The current implementation uses only position (x, y, z), orientation (heading, pitch, roll), and frame number (time). Scale factors (animation of size) are ignored, and so are the advanced spline control parameters such as tension.

In a state where SPLINE is used, VELOCITY and ROTATION is ignored and the instance is forced to move on the spline path at all times. POSITION and ORIENTATION can however still be used to force the animation to start in a well-defined state. If not used, the animation will start from the position/orientation the instance happens to have when entering the animation.

Spline paths are in Lightwave™ defined in parent space. The MDL interpretation of the motion file is however different. It uses the position and orientation of the instance at the time it entered the state to define the coordinate system for the spline.

Example: An animation state is defined with a spline path confined to the horizontal plane, i.e. all control points have y = 0. An instance that enters this state will however not necessarily move in the horizontal plane if it happens to be tilted when it enters the state. Unless forced by the ORIENTATION keyword, it will instead start moving in the plane defined by its x and z body axis at that moment.

As a result of this reinterpretation, spline paths defined in Lightwave™ must have their first control point at (0, 0, 0) or the instance entering the state will “teleport” to a new position.

The Acceleration Keyword

This keyword is especially handy for modeling gravity, although acceleration in any direction is possible. Debris from explosions is an example where gravity is most useful. The keyword can only be used within Animation blocks. Syntax:

ACCELERATION ax ay az

The units are meters per second squared. Because of the common need to model gravity, the acceleration keyword deviates from the standard method of expressing movement in local body coordinates. Parent coordinates are therefore used, and for modeling gravity correctly, the object should be a top-level object or a child to a parent with the same orientation as the world.
When using the Acceleration keyword in a state, the meaning of VELOCITY is changed. It will only be used as an initial velocity when the instance enters the state, *not* for subsequent animation. It will also denote a velocity in parent (world) space, *not* as usual a local body axis velocity.

An example:

```
BEGIN_ANIMATION Explode
  VELOCITY 10 20 10  // kick it upwards and to north east
  ACCELERATION 0 –9.81 0  // gravity acts along negative y axis
  ROTATION 100 200 300 // some arbitrary spin
  ...
  ...
END_ANIMATION
```

Misc.

An EntModel may be fitted with a Track Generator. This will produce vehicle tracks on the ground when the instance moves. As with shadow models, this requires that the application has loaded a World object that can supply elevation values. The syntax for tracks is:

```
TRACKS texture.tga fade-distance-in-meters
```

This texture is special: it should not have an alpha channel, but should have intensity of 0.5 in the RGB channels for pixels that should be transparent. The underlying ground texture will be modulated by the tracks, pixels in the track texture with values above 0.5 will lighten the ground, and values below will darken it.

Normally, the bounding sphere radius used for object culling is derived from the first model of the first animation state found during parsing. If this is not appropriate, it can be overridden by the RADIUS command: Syntax:

```
RADIUS meters
```

To disable certain entities when running on low-end systems, or replacing them by cheaper versions, the PRIORITY keyword can be used. Syntax:

```
PRIORITY  int
```
or:
```
PRIOTITY  int to int
```

The legal range is 1 to 5, which is also the default min- and max values. These three keywords should be used at the entity level.

Example:

```
BEGIN_ENTITY Vehicle
   ...
   PRIORITY  3 TO 5
   TRACKS texturename.tga  4.0
   RADIUS 8.0
   ...
END_ENTITY
```

The Particle System

Particles are rectangular, semi-transparent texture-mapped and possibly texture-animated rectangles that have a size and a 3D position in the world, but no orientation. They are always rendered facing straight towards the camera. Particles are mostly used for effects such as smoke, fire and explosions.

Streaks are particles with two defining 3D positions. They are usually linked together so that one streak’s end position is the start point of the next one.

Entmodels can be fitted with particle generators that spawn particles completely automatic. Particle generators are always defined within an Animation block, and will only be active when the instance is in that state.

Particle generators can currently not be defined inline in the MDL file. Instead, a special PG file is referenced from the MDL. Any number of particle generators can be defined in an animation state
In the MDL file, the syntax for defining a particle generator is:

```
BEGIN_ANIMATION Burn
   ...
   BEGIN_PARTICLEGENERATOR
      PRIORITY min [TO max]
      POSITION x y z
      FILE pgdefinition.pg
   END_PARTICLEGENERATOR
   ...
END_ANIMATION
```

The PG file syntax

In the PG file, the following keywords can be used:

- TEXTURE filename.tga // the texture(s) used. Can be repeated.
- ANIMATION_TIME seconds // time for one animation cycle
- TIME_LIMIT seconds // only active this number of seconds
- V0 vx vy vz // initial velocity (m/s) in local body axis
- VMAX vx vy vz // terminal velocity (m/s) in *world* space
- FRICTION value // how fast we approach terminal velocity
- ACCELERATION ax ay az // acceleration in *world* space (m/s2)
- START_SIZE meters // initial size
- START_OPACITY percent // 0.0 – 1.0
- TYPE type // PARTICLE or STREAK
- BLEND_TYPE blendtype // ADDITIVE or ALPHA
- RANDOM_ORIENTATION value // ON or OFF
- SPAWN_TIME seconds // time between particles
- BEGIN_PHASE / ENDPHASE // defines development over time. Can be repeated.
- PRIORITY min [TO max] // defaults, can be overridden in MDL file

Explanations

The TEXTURE keyword can be repeated any number of times. The particles will loop through all specified textures on ANIMATION_TIME seconds, in the order the textures appear. If only one texture is specified, ANIMATION_TIME may be omitted.

If present, TIME_LIMIT will force the generator to stop emitting particles after specified number of seconds into the state. Spawn particles will however continue to live until their lifetime expires

V0 is the initial velocity, defined in local body axis. Will default to zero if omitted. VMAX is the particles terminal velocity that it will eventually reach provided that no acceleration is used. VMAX is typically the wind vector, causing clouds to drift away. Note that VMAX uses world coordinates as opposed to V0 that uses body axis.

FRICTION is the rate in which the current speed is transformed into VMAX. (Technically, an acceleration that is proportional to the remaining difference between current velocity and VMAX) Typical values range from close to zero up to more than 1.0. In old PG files, this parameter was called DELTA_V, and that alias is still recognized for backward compatibility.

ACCELERATION can be used instead of, or together with VMAX/FRICTION. It accelerates the particle in world coordinates and is typically used to make particles representing sand, gravel etc. fall to the ground rather than drifting away. Measured in meters per square second, y = -9.81 will produce correct gravity. This keyword is optional.

START_SIZE is the particle’s size in meters. For streaks, this value is used for width only, the length is defined by the two end points.

START_OPACITY is a value between 0.0 and 1.0. Both size and opacity can develop through one or more particle phases (described below) that specify the final values after a specified time.

TYPE can be PARTICLE or STREAK. The default is PARTICLE. STREAKS will be stretched and linked in a chain so that one ends where the next begins.

BLEND_TYPE can be ADDITIVE or ALPHA. ALPHA is the default. Additive particles should have 24-bit textures where black represents transparent. They are useful for fire and explosions that should look like they emit light. Alpha particles need 32-bit textures where the alpha channel represents transparency. They should be used for dark smoke, dust clouds, sand spray etc. that should be shaded by the ambient light rather than glowing by themselves. The texture alpha is the particles maximal opacity value, which will be modulated by the particle’s current opacity to create fade-in/fade-out effects.

RANDOM_ORIENTATION controls whether all particles are spawn with the same texture orientation or not. For streaks, it should normally be OFF. For particles that represent chaotic motions like explosions, RANDOM_ORIENTATION can be turned ON to minimize repetition in the textures. The default value is ON.

SPAWN_TIME is the time in seconds between generated particles. After being spawn, the particles will live a time that is determined by the particle phases described below.

PRIORITY can be defined in the PG file, but also overridden in the PARTICLE_GENERATOR block of the MDL file. It can be used to disable certain effects when running with low detail setting, or to replace an expensive effect with a cheaper one.

Particle Phases

The syntax above does not specify final values for the variables *size* and *opacity*. The reason is that both expanding/contracting and fade-in/fade-out phases are commonly needed. Therefore, the final values are specified in a *particle phase block:*

```
BEGIN_PHASE
  FINAL_SIZE    meters
  FINAL_OPACITY percent
  TIME      seconds
END_PHASE
```

Any number of phases can be specified. The final values from one phase then become the initial values for the next. The particle’s total lifetime will be the sum of all phase’s TIME parameter. Normally, the last phase will fade the opacity to zero. Most particles will have expanding size in all phases, but especially streaks sometimes need shrinking to null size.

Examples:

This first example shows a minimal PG file that emits puffs of smoke once a second. The particles will float straight up with 2.0 m/s, expand from 1.0 to 5.0 m and fade to transparent during 5 sec, then die. Default values will make this type PARTICLE, blendtype ALPHA, no friction, no acceleration, and random orientation for every generated particle:

```
TEXTURE     Textures/smokepuff.tga
V0    0 2 0
START_SIZE    1.0
START_OPACITY    0.5
SPAWN_TIME    1.0

BEGIN_PHASE
  FINAL_SIZE    5.0
  FINAL_OPACITY    0.0
  TIME        5
END_PHASE
```

To add texture animation during the particle’s lifetime, simply add more textures and supply an animation time:

```
TEXTURE   Textures/smokepuff1.tga
TEXTURE   Textures/smokepuff.2tga
TEXTURE   Textures/smokepuff.3tga
TEXTURE   Textures/smokepuff.4tga
ANIMATION_TIME0.8
```

To make the smoke slow down its upward motion, and start drifting north, add VMAX and FRICTION:

```
VMAX  0.0  0.0  4.0
FRICTION  0.5
```

To fade in from transparent during one second, add an additional phase. The entire file should then look something like this:

```
TEXTURE   Textures/smokepuff1.tga
TEXTURE   Textures/smokepuff.2tga
TEXTURE   Textures/smokepuff.3tga
TEXTURE   Textures/smokepuff.4tga
ANIMATION_TIME0.8
V0   0 2 0
VMAX  0.0 0.0 4.0
FRICTION    0.5
START_SIZE  0.5
START_OPACITY 0.0
SPAWN_TIME  1.0
```

```
BEGIN_PHASE
  FINAL_SIZE    1.0
  FINAL_OPACITY   0.5
  TIME   1
END_PHASE

BEGIN_PHASE
  FINAL_SIZE    5.0
  FINAL_OPACITY   0.0
  TIME   5
END_PHASE
```

Static Particles

The particle generator syntax described above will produce particles that are *generated* at a certain position in an EntModels model space. However, after they spawn, they are not longer connected to that model. For the rest of their lifetime, they will develop in *world space.*

Sometimes particles are needed that have a fixed position relative a parent EntModel object. In such cases, *Static Particles* can be used. Common examples are *halos* around light sources.

Static particles are defined inline in an MDL file’s ENTITY block. They have no phases and cannot be animated in any way. Five keywords can be specified; POSITION, TEXTURE, BLEND_TYPE, SIZE and OPACITY, all within a BEGIN_PARTICLE / END_PARTICLE block.

Example:

```
BEGIN_ENTITY
  ...
  BEGIN_PARTICLE
    POSITION 0 2 0
    TEXTURE  Textures/Halo.tga
    BLEND_TYPE ADDITIVE
    SIZE 1.0
    OPACITY 0.7
  END_PARTICLE
...
END_ENTITY
```

Static particles can use the ADDITIVE and ALPHA blending modes, where ADDITIVE means that overlapping particles increase intensity towards white, while ALPHA is a more realistic transparency.

Lens flares

The static particles described above may be used as *halos* around light sources. A similar effect can be created through the use of a *lensflare block.*

The major difference between a halo and a lensflare is that the latter is an effect that imitates - as it’s name implies - the effect of glaring rings that sometimes occurs in camera lenses when exposed to a strong light source. Since the effect is assumed to be produced in the camera, the size of the particle is not affected by the distance to the light source. Furthermore, a lensflare block may specify an entire set of particles distributed on a line connecting the light source’s position on the screen with the center of the screen. Finally, the entire set of flares is only drawn if the light source itself is visible, that means that it will ‘flicker’ if the light source passes behind any obstacles.

Lens flares are specified within a `BEGIN_LENSFLARE / END_LENSFLARE` block defined inside an `ENTITY`. Within this block, a POSITION keyword specifies the model space location of the light source. Other keywords may either be entered after an INLINE keyword, or inside a special lensflare file specified with a FILE keyword:

```
// Alt. 1

BEGIN_ENTITY
  BEGIN_LENSFLARE
    POSITION 1.0 2.0 0.0
    INLINE
    // particle definitions
  END_LENSFLARE
END_ENTITY

//Alt. 2

BEGIN_ENTITY
  BEGIN_LENSFLARE
    POSITION 1.0 2.0 0.0
    FILE filename.lf
  END_LENSFLARE
END_ENTITY
```

The content of the lensflare file (or listed after the INLINE keyword) consists of any number of PARTICLE blocks, looking much like the particle blocks used for static particles. Their sizes have however no physical meaning but are quite ad hoc. (Reasonable sizes range approx. 0.01 – 2.0). The DISTANCE keyword specifies how far along the line connecting the center of the screen and the light source this particular flare is located; 0.0 denoting the center of the screen, 1.0 representing the light source. Distances outside this range are also legal:

```
BEGIN_PARTICLE
  TEXTURE textures/X-flare.tga
  SIZE 0.20
  DISTANCE 1.0
  OPACITY 0.7
END_PARTICLE

BEGIN_PARTICLE
  TEXTURE textures/Halo.tga
  SIZE 0.30
  DISTANCE 0.8
  OPACITY 0.5
END_PARTICLE
```

At most one particle block per lensflare can finally include the keyword BLINDING, followed by two floating-point arguments:

```
BEGIN_PARTICLE
  TEXTURE textures/X-flare.tga
  SIZE 2.0
  DISTANCE 1.0
  OPACITY 0.7
  BLINDING 0.1 1,0
END_PARTICLE
```

A particle containing the BLINDING keyword will fade between invisible (opacity 0.0) and the value specified with the OPACITY keyword. The two arguments are distances from the center of the screen where 1.0 is approximately the edge of the screen. In the example above, the particle will fade from invisible at the screen border to maximal opacity (0.7) when it comes close to the center.

Like static particles, lens flares are always rendered additively. The BLINDING keyword can therefore be used with a big particle (note the size of 2.0 in the example above) to fade the scene to white as if "over-exposed" by a very strong light source. Therefore, the BLINDING particle should normally have DISTANCE 1.0, which will position it right on the light source.

Side note: Used from a MDL file, lens flares are always given a 3D model-space *position*. The Lensflare class can however be used directly in a C++ program, and in such cases a 3D *direction vector* can be substituted instead. This is useful for creating lens flares for directional light sources such as the sun.

Priority System

The graphic engine can be running with a *priority level* between 1 and 5. Entities and particle generators can be assigned a *priority range*, and only if the current engine priority level falls within this range is the object rendered:

```
PRIORITY  3
PRIORITY  3 TO 4
```

The second argument can be omitted, this implies that there is no upper limit. The syntax can be used in an ENTITY block, in a PG file, and in the PARTICLE_GENERATOR block within an animation. Used in the ENTITY block, whole objects or child objects can be turned off. In a PG file, the values are defaults for all particle generators referencing the file, but these values can be overridden by using the PRIORITY keyword inside the PARTICLE_GENERATOR block of individual Entities.

Used with only one parameter, PRIORITY can be used to disable certain expensive effects when the detail level is turned down, e.g. on a slow low-end system. Used with both parameters, an expensive effect can instead be replaced with a cheaper one.

Example:

```
BEGIN_ENTITY Expensive_Version
  …
  PRIORITY 4 TO 5
  …
END_ENTITY
BEGIN_ENTITY Cheap_Version
  …
  PRIORITY 1 TO 3
  …
END_ENTITY
```

Syntax Reference

ENTITY block:

```
BEGIN_ENTITY EntName

       ANIMATION BLOCK      // may be repeated
       CHILD BLOCK          // optional, may be repeated
       STATIC PARTICLE BLOCK       // optional, may be repeated
       LENSFLARE BLOCK      // optional, may be repeated
       TRACKS filename float       // optional
       RADIUS float           // optional, default from first FRAME
       POSITION float float float    // optional, default 0 0 0
       ORIENTATION float float float // optional, default 0 0 0
       VELOCITY float float float    // optional, default 0 0 0
       ROTATION float float float    // optional, default 0 0 0
       SPLINE DEFINITION      // optional
       PRIORITY int [TO int]       // optional, default 0 5

END_ENTITY block:
```

ANIMATION block:

```
BEGIN_ANIMATION AnimationName

       LOD BLOCK          // optional, may be repeated
       PARTICLEGENERATOR BLOCK   // optional, may be repeated
  ANIMATION_TIME float      // needed when LODs have several frames
  LOOPED ON|OFF        // optional, default ON
  TIME_LIMIT float     // optional, default infinite
       NEXT_ANIMATION string       // optional
       POSITION float float float    // optional
       ORIENTATION float float float // optional
       VELOCITY float float float    // optional, default from ENTITY
       ROTATION float float float    // optional, default from ENTITY
       SPLINE DEFINITION      // optional
       ACCELERATION float float float// optional, default 0 0 0

END_ANIMATION
```

CHILD block:

```
BEGIN_CHILD

  POSITION float float float     // optional, default from child ENTITY
  ORIENTATION float float float  // optional, default from child ENTITY
  FILE filename        // FILE or ENTITY BLOCK, not both!
  ENTITY BLOCK
  STATIC          // optional

END_CHILD
```

LOD block:

```
BEGIN_LOD float

  FRAME filename     // may be repeated
  TEXTURE_FRAME filename    // optional
  SHADOW_FRAME filename   // optional

END_LOD
```

SPLINE Definition

```
SPLINE filename float [LOOPED ON | OFF]
```

PARTICLEGENERATOR block:

```
BEGIN_PARTICLEGENERATOR
  POSITION float float float
  FILE filename.pg
  PRIORITY int [TO int]// optional, overrides PG-file values
END_PARTICLEGENERATOR
```

PG (particle generator) file syntax

```
TEXTURE filename          // can be repeated
ANIMATION_TIME float        // only needed if more than one texture
TIME_LIMIT float          // optional
V0 float float float        // optional, default 0 0 0
VMAX float float float      // optional, default 0 0 0
ACCELERATION float float float // optional, default 0 0 0
FRICTION float          // optional, default 0
START_SIZE float
START_OPACITY float
SPAWN_TIME float
TYPE PARTICLE | STREAK      // optional, default PARTICLE
BLEND_TYPE ADDITIVE | ALPHA    // optional, default ALPHA
PHASE BLOCK          // may be repeated
PRIORITY int [TO int]     // optional, default 0, 5
```

PARTICLEPHASE block:

```
BEGIN_PHASE
  FINAL_SIZE float
  FINAL_OPACITY factor
  TIME seconds
END_PHASE
```

Static PARTICLE block:

```
BEGIN_PARTICLE
  TEXTURE filename   // required
  SIZE float       // required
  OPACITY float      // default 1.0
END_PARTICLE
```

LENSFLARE block:

```
BEGIN_LENSFLARE
  POSITION float float float  // default 0 0 0
  INLINE | FILE filename.lf // if INLINE, particle blocks follow,
           // otherwise found in specified file
  LENSFLARE PARTICLE BLOCK  // may be repeated
END_LENSFLARE
```

LENSFLARE PARTICLE block:

```
BEGIN_PARTICLE
  TEXTURE filename   // required
  DISTANCE float     // required, 0 = screen center, 1 = light source
  OPACITY float      // default 1.0
  SIZE float       // ad hoc units normally ranging 0.01 – 2.0
  BLINDING float float    // fading range, max one per lensflare
END_PARTICLE
```

Closing comments

```
/*   */       // may be nested
//       // one-line comment
```

Aliases:

The following keywords are obsolete, but still recognized for backward compatibility:

Keyword:	**Replaced by:**
TIME	ANIMATION_TIME
ANIM_TIME	ANIMATION_TIME
DELTA_V	FRICTION
SELF_ROTATION	ROTATION

Miscellaneous:

The order between keywords is generally not significant. The exceptions are:

FRAME statement in Animation blocks: Animation sequence specified by the order.
TEXTURE statements in PG files: Animation specified by the order.
PHASE blocks in PG files. Particle development specified by the order.
LOD blocks in Animations: Must have increasing distances.
ANIMATION blocks: Instances will by default use the first animation defined.

White space is the only delimiter used (no commas between arguments!) File names or other parameters may not contain space, tab or new line.

The language is case insensitive for both keywords and parameters.

Numerical parameters may be given in any form recognized by the atof() function.

Indentation is purely for readability and not important for correct parsing.

APPENDIX A – DICTIONARY OF TERMS

A big thank you to TomBob of the official Ground Control™ forum for the original of this dictionary, as well as through writing it, reminding me that it really is necessary to explain the language one uses in these kinds of documents.

ANIMATION	An animation of frames from within a Model. Ground Control™ can't actually play animations so animations have to call every frame separately in succession.
BEGIN_	Begins an entity, child, animation, or a predefined OBJECT.
CHILD	An OBJECT attached to a previous one.
END_	Opposite of BEGIN_
ENTITY	Anything created to be used as an item within the game engine. This "object" can be created and killed at command.
FILE	Calls an external file for the OBJECT/FUNCTION it belongs to.
FRAME	Calls a frame and/or new model into the PARENT/OBJECT it modifies.
LIGHTSOURCE	An OBJECT that creates light. Required modifiers are FILE, RADIUS, and POSITION. FILE needs LOOPED setting.
LOD	Level of Detail. Used to switch models when the camera zooms in and out in order to increase performance. Measured in meters, negative numbers are infinity. (Called in MDL ANIMATION BLOCKS under keyword FRAME as L# where the # is the # of the LOD. A model can have as many LODs as the creator decides to make.
LOOP	Defines if the PARENT is looped or not. Valid PARAMETERS are ON and OFF.
NEXT_ANIMATION	Calls a previous (or upcoming) defined ANIMATION to be played. Used as: NEXT_ANIMATION "name".
Object	The entire result of START/END quotes. There really has to be nothing within the START/END to make it an Object.
Parent	The previous Entity or action called before. Usually written before a modifier.
PARTICLE GENERATOR	Takes the FILE assigned to it and sets it's SCALE and ALPHA from 0 and 100 to 1.0 (opaque) and 0.0 (transparent) respectively, then removes the file that reached the ALPHA of 0 from the game.
POSITION	Position of an OBJECT/FUNCTION it belongs to. Usually in (x, y, z) coordinate form.
RADIUS	The approximate of the PARENT/OBJECT it modifies. Origin is the Center of the PARENT/OBJECT (usually 0, 0, 0).
SHADOW_FRAME	Calls a model to be the shadow. Classically only used in LOD 60 & 150.
STAND animation	The idle animation of the entity, when it stands still and does nothing.
START/END	Beginning and End of an Object, defined by BEGIN_ and END_
TIME	Different from TIME_LIMIT in that it assigns the time it gets to perform the PARENT, it does not restrict it.
TIME_LIMIT	Given time to perform the object it modifies.
TRACKS	Modifier – calls a texture to be created right after the PARENT OBJECT.
WALK animation	Animation for when the entity is moving.

APPENDIX B – ESSENTIAL TOOLS

This section will briefly describe the essential tools you will need to modify the different parts of Ground Control™. These tools was used and written by Massive Entertainment to assist the development of Ground Control™.

Mp3ToMpd.exe

This tool is used to convert MP3 files to MPD. An MPD file is a type of MP3 with extra information added, so that Ground Control™ knows how long the file is before playing it. Even though MPD files can be played in mp3-playing programs such as Winamp™, MP3 files cannot be played directly through Ground Control™.

SDFextract.exe

If you want to unpack a SDF file to change or to look at the files it contain you need an extractor. This tool is the one you use for this purpose. The program can extract files from all SDF archives – mods, GenEd files, maps and the original data files.

SDFman.exe

When you have created a MOD you should pack it into a SDF file – this to make it easier for others to play it. The SDFman (-ager) tool is used to do this.

PackMOD.exe

This program will compress / serialize all Lightwave™ LWO files and other model data into .MOD files and place them in the Data\PackMOD directory. This is necessary in order for the Ground Control™ retail executable to be able to read the models.