

GOLGOTHA REVIVAL TEAM



Golgotha Real Time Game Engine Manual

Patrick Grawehr

GOLGOTHA REVIVAL GROUP

Golgotha Real Time Game Engine Manual

©2002 – 2004 Golgotha Revival Group

Author: Patrick Grawehr

Golgotha is a legal trademark of crack dot com. Windows is a legal trademark of Microsoft Corporation. Other names might be trademarks of other companies. This document is Copyrighted by the Golgotha Revival Group. Consult <http://qbic.vipnet.ro> for more information about licensing this document and the related software.

Legal Notice:

I nor any other member of this group will be responsible for what you do with the supplied software or documentation. We won't be responsible for any damage caused by this software.

Contents

1	Introduction	5
1.1	About the Name	5
1.2	Features	6
1.3	System Requirements	7
1.3.1	Hardware and Operating System	7
1.3.2	Special Software requirements	7
2	Installing Golgotha.....	9
2.1	Download and Installation	9
2.1.1	Windows, End User Version (Distributed on CD)	9
2.1.2	Linux, RPM packages	9
2.1.3	Installing from Source	9
2.1.4	Checking out from the Subversion repository.....	10
2.2	The Source Tree	12
3	Using Golgotha	17
3.1	Starting Golgotha	17
3.2	Controls.....	19
3.2.1	Global keys.....	19
3.3	The Game Modes.....	21
3.3.1	Action Mode.....	21
3.3.2	Follow Mode	22
3.3.3	Camera Modes	22
3.3.4	Strategy Mode	22
4	Developer Information	23
4.1	General information about the source.....	23
4.2	Using the subversion client (svn)	23
4.3	The i4 core	24
4.3.1	The platform specifications and the data types	24
4.3.2	The initialization classes	24
4.3.3	The message passing kernel	26
4.3.4	The Memory Manager.....	27
4.3.5	The Template classes: i4_array<T>.....	27
4.3.6	The Template classes: i4_fixed_que<T> and i4_dynamic_que<T>	27
4.3.7	The String classes	27
4.3.8	The File Management Functions.....	28
4.4	The Graphics Subsystem	33
4.4.1	The video engine	33
4.4.2	The rendering engine.....	33
4.5	The Window Management.....	33
4.6	The Golgotha core.....	33
4.6.1	The objects	34

4.6.2	The Network support functions.....	43
5	Trouble Shooting.....	49
5.1	Frequently asked questions and frequent problems.....	49
5.1.1	Problems compiling the game	49
5.1.2	Problems launching the game.....	50
5.1.3	Problems running the game	50
6	Appendixes	53
6.1	References	53
6.2	Index	54

Introduction

This chapter contains the basics you need to know about Golgotha.

Golgotha is a very flexible and portable real time game engine. Its main focus is 3D Processing, but it can be used for any kind of simulation you desire.

The game engine Golgotha was originally developed by crack.com (spoken as "crack dot com") around 1998. The team of crack.com also developed the much better known game "Abuse", which is part of most linux distributions around nowadays. Unfortunately, the Golgotha was never finished because the team was out of money and crack.com went bankruptcy. The members of the team made an interesting decision at this time and released all their code and data they already had to the public.

1.1 About the Name

I do not know how Jonathan Clark and his team at Crack.com came to this name, but I know where it comes from. The cross in the logo indicates that they knew what meaning the name has. For those of you, who don't, a short fragment from the book of books: (German first, then English)

Sie übernahmen Jesus. Er trug sein Kreuz und ging hinaus zur sogenannten Schädelhöhe, die auf hebräisch Golgota heißt. Dort kreuzigten sie ihn und mit ihm zwei andere, auf jeder Seite einen, in der Mitte Jesus. Pilatus ließ auch ein Schild anfertigen und oben am Kreuz befestigen; die Inschrift lautete: Jesus von Nazaret, der König der Juden. Dieses Schild lasen viele Juden, weil der Platz, wo Jesus gekreuzigt wurde, nahe bei der Stadt lag. Die Inschrift war hebräisch, lateinisch und griechisch abgefaßt.

Joh 19,16b-20

So the soldiers took charge of Jesus. Carrying his own cross, he went out to the place of the Skull (which in Aramaic is called Golgotha). Here they crucified him, and with him two others--one on each side and Jesus in the middle. Pilate had a notice prepared and fastened to the cross. It read: JESUS OF NAZARETH, THE KING OF THE JEWS. Many of the Jews read this sign, for the

SYMBOLS



Important warnings and unexpected things that might happen.



Parts that should be revised or completed in the manual.



Do something. If a platform (i.e Windows, Linux) is mentioned next to the symbol, the paragraph applies only to this platform.



Further references



Remarks and Tips

place where Jesus was crucified was near the city, and the sign was written in Aramaic, Latin and Greek.

John 19,16-20

One should be careful what this means in the current political context.

1.2 Features

The engine's features currently include the following major topics:

- 6DOF 3D Polygonal Landscape Engine
- 6DOF 3D Object Rendering Engine
- Support for Windows and Unix (Including Linux, Sun Solaris and SGI Irix)
- Hardware and Software rendering code included
- Support for DirectX and OpenGL.
- Multi-language support
- Texture Size of up to 1024x1024 Pixels
- Up to 64k Textures at once
- Realistic Daylight simulation
- Dynamic lighting and shadows.
- More than 2100 Textures ready
- More than 500 Objects ready, including tanks, airplanes, buildings, bridges, pyramids.
- Full 3D-sound support available (no external libraries used)
- Over 40 minutes of music included, giving the different locations an unique feeling.
- Several different levels ready
- Full - Featured Window Management Subsystem
- Full - Featured Common Lisp Scripting Engine (Includes Garbage Collection, fast and reliable exception handling)
- Level - Editor included
- Object - Editor included (No external tools are needed to create new objects)
- Capable of converting 3Dstudio max files.
- Partially working network support

1.3 System Requirements

The core of the Golgotha engine is quite old, so you don't need to have one of those latest high-end gaming towers to get it running.

1.3.1 Hardware and Operating System

To execute Golgotha, you need:

- Windows 95/98/2000/XP (With software rendering, you'll get it probably to run even under NT4) or
- Linux (with at least a 2.x kernel, I suppose) or, if you have the source code release,
- Other Unixes (SUN SPARC Solaris, SGI Irix) or, if you know how to code,
- Any imaginable platform (including game stations like the PS2).
- At least 32MB RAM (128 MB recommended)
- A fast CPU (well, I can't give a speed here, since it depends on the platform). For Intel, 400MHZ should do a quite good job.
- A graphics card, preferably hardware accelerated, with DirectX or OpenGL support and the required drivers installed. Some amount of video memory would also be a good start. The graphics card should at least be capable of using a resolution of 640x480 with 64000 colors (16 Bit).
- For Windows or linux, at least 100MB of free Hard disk space is recommended although Golgotha can be configured to run completely from CD.
- (development only) You need about 700MB free Hard disk space to install and compile the complete source.
- A mouse and a keyboard (If your platform supports these, use anything that matches on a gaming platform like the PS2). For development, you definitely need these.
- (optional) A stereo sound card would be tasty. Attach it to your stereo amplifier to get the most out of the great soundtrack of this game. Sound output is currently only supported on Windows, but we are working on this issue.
- (optional) a CD-ROM drive
- (optional) a Network card for LAN games or a modem for Internet games.

If you do not know which platform (operating system) applies to you, it's most probably Microsoft Windows.

1.3.2 Special Software requirements

Windows:

- DirectX 5 or newer is required. You get the latest version from <http://www.microsoft.com/directx> or most discs that accompany your preferred gaming magazine. The current version supports DirectX 9-interfaces, so it's recommended to have DirectX 9 installed.
- If you didn't download a full installation package, be sure to have the Visual Studio runtime files installed ("msvcrt.dll"). These are automatically installed if you install Visual Studio or some Office program like MS Word.
- (development only) Microsoft Visual C++ 6.0 SP4 or newer. Visual Studio .NET 2003 is working, too; don't use Visual Studio .NET 2002, it's way too buggy. (see further down for instructions on how to compile Golgotha)
- (development only) The DirectX SDK, Version 7 or newer. Be sure that you have the V7 headers, as I've heard rumors that say they aren't included in the newer SDK distributions. (Specifically, you need `ddraw.h` and `ddraw.lib`)

Linux/Unix:

- An OpenGL package is recommended for hardware support. If hardware support is not available, mesa can be installed. But software rendering may be faster than mesa in such cases.
- An X server must be running.
- (development only) gcc (or icc) must be installed and working. The mesa-devel package and the X11 development header files are required.
- (development only) Some plain text editor will be great. So make yourself familiar with nedit or emacs whatever you like. Be aware that the Golgotha source uses DOS (CR/LF) style linefeeds. Neither emacs nor vi properly support such files. I'll recommend to use nedit, a very powerful and easy-to-use editor which does properly distinguish between DOS and Unix style text files.

Installing Golgotha

This chapter describes how to install and configure Golgotha to run on your particular system

2.1 Download and Installation

2.1.1 Windows, End User Version (Distributed on CD)

(Hint: Currently, such a release can only be ordered by postal mail. You must apply the complicated procedure in one of the next paragraphs if you downloaded Golgotha from the internet)

Just insert the Golgotha CD in your CD-ROM drive. The setup program may start automatically. If it doesn't, locate the setup file in the root directory of the disc and double click on it.

2.1.2 Linux, RPM packages

You know how to install RPM packages, do you? If not, check the documentation of your linux distribution to see how this can be done on your particular version.

2.1.3 Installing from Source

Read on if you have downloaded a source distribution. These usually come in a file called "GYMMDD.zip" or "GYMMDD.tar.gz", where the name indicates the date of the compilation in the order year, month, day.

- 1) Switch on your Computer (Hm, since you are an expert if you are trying to get this running, I guess I'll skip that...)
- 2) You need to download the following archives from the Internet: Visit <http://qbic.vipnet.ro> and check the download page there. You'll at least need the source zip or tarball and the textures. The optional sound package is very large but – well, optional. Sometimes, we also distribute incremental packages (those with an U in their name). These can only be applied if you have the full package that one was based on *and* all the intermediate incremental packages.
- 3) Unpack everything in the same directory (i.e. d:\Golgotha). Unpack the oldest archives first, since then, you can overwrite older files with newer ones. Be sure to keep the directory structure stored in the archives intact. The Engine relies heavily on it! You absolutely must use a file system and an unpacking tool that can handle long file names. So do not use an old DOS Unzip utility, since that will render the files unusable. If you are using Linux and installing Golgotha on a FAT partition (such that you can use it with windows, too), be sure to mount the partition as type vfat and not simple ms-dos. (to test whether this has been done, check that you can have long filenames on that volume)



Important: If you intend on contributing to the Golgotha project, you should check out the source instead of just downloading. Check the next section for information on how to this.



Important: After pressing F7 (or F5) for the first time since loading the project, MSVC 6 is checking the make dependencies. This may take up to 2 minutes. During this time, it might seem that Visual Studio has crashed (no reaction to mouse or keyboard events whatsoever), but it will continue as expected after some time. (Well, at least 99% of the time...)

- 4) After unpacking everything, you should have a quite complicated directory structure below your installation directory. The next paragraph shortly describes the contents of these directories.
- 5) Linux only: Make sure that `g_decompressed` is writeable. For development, all files and directories should of course be writable.
- 6) As you can see from the directory structure, the source is partitioned in different packets that together form a component. More about these components can be found further down.
- 7) If you are using Unix or Linux, skip to point 12.
- 8) In windows, open "Golgotha.dsw" (or `Golgotha.sln` if you are using Visual Studio .NET) from the Golgotha main directory. After Visual Studio opened, check that Golgotha is the active project and hit F7. After a few seconds the compiler starts working. This may take about 20 minutes to complete, so have some break.
- 9) If everything went fine, you should see something like "Golgotha.exe compiled, 0 errors, 113 warnings." at the bottom of the build log. You may safely ignore warnings that say "Loss of precision" or "signed – unsigned mismatch". If something ugly happened building the source, consult the trouble shooting section.
- 10) You may now start the program by hitting F5. (or CTRL-F5 to run outside the debugger)
- 11) Read on in the chapter about using Golgotha.
- 12) Open a console window and type `./configure`. Your system specs will be automatically detected and the detected results applied to the source.
- 13) Type "gmake" to build the source now. You should avoid typing just "make", because this might fail on some systems.
- 14) When the build completes, you are ready to run the game.

2.1.4 Checking out from the Subversion repository

This chapter gives a brief introduction to subversion, a source versioning system similar to the well-known cvs. For more information on subversion, check <http://subversion.tigris.org>. You can read the full documentation at <http://svnbook.red-bean.com>. I recommend you read at least chapter 3 "The Basic Working Cycle".

The following steps are very similar, regardless whether you are using Windows or some kind of Unix.

- 1) Download and install the subversion client. You can find the download and instructions on how to install it on the subversion website. Before downloading, you should check on the downloads page of the Golgotha website which version of the server we are currently using. Old versions of the client might not be compatible with our database setup. If you find a newer but incompatible version, you should inform us about this fact and we'll check whether it is time to upgrade the server. You can also find the client of the version we are currently using directly on our website. For windows, make sure that the directory containing `svn.exe` is in the path. Unix users won't usually have to bother about this because it is installed in `/usr/local/bin`, a directory that's always in the path.

- 2) Open a Console Window. CD to the directory below which you want the Golgotha source installed. It's obvious that you need write permissions in that directory.

- 3) Create a new directory called Golgotha and enter it.

```
prompt>mkdir golgotha
prompt>cd golgotha
```

- 4) Make sure your internet connection is active and working:

```
Prompt>ping qbic.vipnet.ro
```

- 5) If you see messages like "Connection timed out", "Target unreachable" or "Unknown Host", you either have a problem with your network connection or the server is currently down.

- 6) Check out (Download) the latest development release of Golgotha. Note that this version might be unstable. It's even possible that it does not compile on one system or the other. Check the recent commit logs when you have an unexpected problem with this version. Of course you are encouraged to ask questions on the forum or write e-mails when something doesn't work for you. Type the following command all on one line. Don't forget the dot (".") at the end, or an additional unnecessary subdirectory "worksrc" will be created below your current location.

```
Prompt>svn checkout
http://qbic.vipnet.ro/repos/golgotha/worksrc .
```

```
A    Readme.txt
A    app__app.cpp
```

```
...
```

```
Checked out revision 47
```

- 7) You'll see the list of files that is being downloaded and copied to your disk. This is called the working copy. Since you're downloading stuff, this might take some time. The overall size of the data being downloaded is approximately 14 Megabytes. This exact size is hard to estimate because it changes with every modification to the source. The last line printed tells you the current revision number – This is the version of the source. You should always keep an eye on the revision numbers, since they'll tell you whether your working copy is up-to-date.
- 8) Check out the textures. Since this is a very large but rarely changed package, it needs to be checked out independently. You could also install the packaged release of the textures now, but you won't be able to update them easily if something changes. If you're thinking about working on or adding to the textures themselves, you definitely need to check them out. The size of this package is approximately 70Mb.

```
Prompt>mkdir textures
Prompt>cd textures
Prompt>svn checkout
http://qbic.vipnet.ro/repos/golgotha/workdata/textures .
A    120mm.jpg
...
Checked out revision 47
Prompt>cd ..
```

- 9) Check out or download the sfx (Sound) package. This package is completely optional, but otherwise the same things said for the textures apply. This is by far the largest package (250Mb). (Before you

type this, make sure the current directory is again the main directory.

```
Prompt>mkdir sfx
Prompt>cd sfx
Prompt>sfx checkout
http://qbic.vipnet.ro/repos/golgotha/workdata/sfx
A      intro.wav
...
Checked out revision 47
Prompt>cd ..
```

- 10) If you're using unix or linux, you should now configure the installation. After this, you can build the source.

```
Prompt>./configure
Checking for gcc33... no
Checking for gcc32... yes
...
Prompt>gmake
```

- 11) For Windows, open Golgotha.dsw (MSVC6) or Golgotha.sln (VS.NET) and hit F7 to compile everything.

You can now execute the game or make modifications to the source exactly as you would for any other project. You will only need to return to the command prompt if you do one of the following: Creating new files to be added to the project, removing some files from the project (i.e. because they're being replaced by other files) bringing your working copy up-to-date with the latest revision or committing your changes. Check chapter 4 for an overview on how to achieve these tasks.

2.2 The Source Tree

Bellow the main golgotha directory, several subdirectories are created during the installation. A quick overview over their use is shown bellow.

```
Datenträger in Laufwerk D: EXTENDED
Seriennummer des Datenträgers: XXXX-XXXX
Verzeichnis von D:\golgotha
```

```
.          <DIR>          01.12.01  12:01 .
The current directory
It contains all the .cpp files for the project plus
the .h files of the golg__ part of the engine.
Further should it contain one resource.res file. It
also contains .dsp and .dsw files for visual studio
and several Makefiles for linux and other unixes.
After compiling, the .exe is also placed here. You
might also find several .level files with the
corresponding .scm files here. (There should at last
be a test.level and a test.scm file present). Do not
be surprised if this "main" directory contains more
than 1000 files.
..          <DIR>          01.12.01  12:01 ..
The parent directory
APP          <DIR>          06.06.02  23:53 app
Contains the headers of the application object (app.h
and registry.h)
AREA          <DIR>          01.12.01  12:05 area
Contains the header of the window area metrics
```

```

(rectlist.h)
BITMAPS      <DIR>          01.12.01  12:05 bitmaps
Contains all bitmap files that are used for the
graphical user interface. These are organized in
several subdirectories.
BUILD        <DIR>          01.12.01  12:05 build
TGA files for each object that can be built from
strategy view. The names must have the form
build_[InternalName].tga, where InternalName is the
name given in the definition macro of the
corresponding object source file.
CHEATS       <DIR>          01.12.01  12:05 cheats
Icons for the cheat menu.
CURSORS      <DIR>          01.12.01  12:05 cursors
Cursors in different formats
EDITOR       <DIR>          01.12.01  12:05 editor
The Editor Icons
LOGOS        <DIR>          01.12.01  12:05 logos
A logo image for each vehicle that is used for the
popup in strategic view.
MENU         <DIR>          01.12.01  12:05 menu
The images of the main-menu entries.
MOVIES       <DIR>          05.03.02  17:49 movies
Movies may come here. I must first find an usage for
them and a portable way of reading them.
NETWORK      <DIR>          01.12.01  12:05 network
Network startup dialog backgrounds
OLD          <DIR>          01.12.01  12:05 old
Some images that are really used, but are here for
historical reasons
OPTIONS      <DIR>          01.12.01  12:05 options
More history
RADAR        <DIR>          01.12.01  12:05 radar
The radar images of all units and buildings.
STANK        <DIR>          01.12.01  12:05 stank
These images are used for the status bar in action
mode.

CD_MAKER     <DIR>          01.12.01  12:05 cd_maker
Resource files used to create a .cd file. The format
of that file must be changed and is obsolete. These
files are not used at the moment.
CHECKSUM     <DIR>          01.12.01  12:05 checksum
The checksum.h header.
COMPRESS     <DIR>          01.12.01  12:05 compress
Rle.h
DEBUG        <DIR>          01.12.01  12:06 Debug
(perhaps not yet present) will contain the .obj and
.sbr files after compilation
DEMOS        <DIR>          01.12.01  12:07 demos
Nothing important at the moment
DEVICE       <DIR>          01.12.01  12:07 device
Several headers for the golgotha message passing
kernel. These are integral parts of the i4 core.
DLL          <DIR>          01.12.01  12:07 dll
Headers for the plug-in subsystem (not yet working)
DOCSDATA     <DIR>          01.12.01  12:07 docsdata
(unimportant) Older documentation and some tips. I'm
planing to use doxygen to generate some more
elaborate source code documentation.

```

If the subdirectory "html" is present, it contains the latest doxygen build. Open index.html from that directory. Hint: The documentation might not be from the latest version, since building it takes aproximatelly 20hrs!

EDITOR	<DIR>	01.12.01	12:07	editor	The headers and dialogs for the level editor
ERROR	<DIR>	01.12.01	12:07	error	The headers of the error handler
FILE	<DIR>	01.12.01	12:07	file	File management functions
FONT	<DIR>	01.12.01	12:07	font	Font functions (not the font files themselves, these belong to the bitmaps)
GUI	<DIR>	01.12.01	12:08	gui	Gui class definitions files
G_COMP~1	<DIR>	01.12.01	12:07	g_compressed	(automatically generated) Obsolete. Delete it (together with all contents) if it still exists in your source tree.
G_DECO~1	<DIR>	01.12.01	12:07	g_decompressed	(automatically generated) Contains the decompressed textures if needed. (windows unpacks .jpg files runtime) Also contains the tex_cache.dat files. This directory needs to be writable for everyone to avoid problems.
IMAGE	<DIR>	01.12.01	12:08	image	Image management and handling headers.
INIT	<DIR>	01.12.01	12:08	init	Headers for the load-time initialisation stuff.
IVCON	<DIR>	01.12.01	12:08	ivcon	This directory contains ivcon, a simple command-line utility to convert between different 3D file formats. It can be used to load your 3D-Studio Max files into golgotha.
LISP	<DIR>	01.12.01	12:08	lisp	The headers for the lisp engine.
LOADERS	<DIR>	01.12.01	12:08	loaders	The headers for the different image loaders
MAIN	<DIR>	01.12.01	12:08	main	Main.h and winmain.h.
MAKE	<DIR>	06.06.02	22:55	make	The i4_make tool. This is not used right now.
MAP2DEF	<DIR>	10.04.02	10:55	map2def	A tool to convert .map files to .def files to quickly generate an export definition for a dynamically loaded library (.dll)
MATH	<DIR>	08.06.02	16:34	math	Mathematical functions and compatibility definitions.
MAXTOOL	<DIR>	01.12.01	12:09	maxtool	Maxtool functions
MEMORY	<DIR>	01.12.01	12:09	memory	Memory management functions and generic templates for arrays, hash-tables and queues.
MENU	<DIR>	01.12.01	12:09	menu	Menu headers.
MUSIC	<DIR>	01.12.01	12:09	music	Stream.h
NET	<DIR>	01.12.01	12:09	net	
NETWORK	<DIR>	01.12.01	12:09	network	

```

Networking headers
OBJECTS      <DIR>      01.12.01  12:09  objects
All GMOD files that contain the models used in the
game are stored here.
OBJJS       <DIR>      01.12.01  12:09  objs
The headers for the active objects (objs__ part) are
stored here. Most of the objects have one or more
corresponding files in the objects directory,
although it's not needed that they are named the
same.
PALETTE     <DIR>      01.12.01  12:09  palette
Pal.h
PLUGINS     <DIR>      23.03.02  11:53  plugins
Currently empty.
POLY        <DIR>      01.12.01  12:09  poly
Some headers (for what?)
QUANTIZE    <DIR>      01.12.01  12:09  quantize
Image helper functions
RELEASE     <DIR>      01.03.02  11:37  Release
(automatically generated) If you do a release build,
files go here
RENDER      <DIR>      01.12.01  12:09  render
Headers of the rendering subsystem, organized in
subdirectories for the different platforms.
RESOURCE    <DIR>      01.12.01  12:09  resource
The resources for the i4 core engine. Contains
bitmaps for the gui and .res files that configure the
game.
The keys.res file in this directory contains the
control configuration. Edit it at your own desire.
SAVEGAME    <DIR>      01.12.01  12:16  savegame
Your savegames go here
SCHEME      <DIR>      01.12.01  12:09  scheme
Contains the lisp sources for golgotha and the menu
scripts.
SFX         <DIR>      01.12.01  12:09  sfx
Contains music and sound. Currently, only wav files
can be played real-time.
SOUND       <DIR>      08.06.02  16:31  sound
The sound headers
STATUS      <DIR>      01.12.01  12:12  status
Status.h
STRING      <DIR>      01.12.01  12:12  string
String.h
TEXTURES    <DIR>      01.12.01  12:13  textures
The textures. This is by far the largest of the
subdirectories (in number of files, not size). It
must contain ONLY .jpg or .tga files (and not a file
with both formats, of course). JPG is default, TGA is
used for textures that need color-keying or alpha
transparency.
THREADS     <DIR>      13.06.02   9:43  threads
The (os-dependent) thread management stuff
TIME        <DIR>      01.12.01  12:15  time
Time headers.
TRANSP~2    <DIR>      08.04.02  16:03  transport
Transport headers. Used for transportation
simulations.
UNDO        <DIR>      19.06.02  18:46  undo
(automatically generated) Undo infos. The contents of

```


this directory can safely be erased, if needed.
 (Unless the engine is running, of course)

VIDEO	<DIR>	01.12.01	12:16	video
The os-dependent video rendering subsystem.				
WINDOW	<DIR>	01.12.01	12:16	window
The core of the i4 window management.				
WIN95	<DIR>	01.12.01	12:16	win95
OS-Dependent, but mainly unused stuff.				

Using Golgotha

This chapter describes the main controls you use when playing with Golgotha.

3.1 Starting Golgotha

To start Golgotha just choose it from the start menu or double click on the “Golgotha.exe” file in windows explorer. After a short while (a progress bar shows how long it will take) you are taken to the main menu. If you get errors or crashes instead, go to the trouble shooting chapter.



Open a terminal window and change to the directory where Golgotha is installed. Then start the Golgotha executable (either “golgotha.elf” or just “golgotha”) I.e. type:



```
pgrawehr@Ibmnote1$>cd /mnt/win_d/golgotha
pgrawehr@ibmnote1$>./golgotha.elf
Initialisation in progress. 0% complete.
...
```

After a few seconds, a window should pop up and Golgotha will be ready. If not, check the output in the console window and read on in the chapter about trouble shooting.

Important: Don't try to run Golgotha from another directory than it is installed in. This may cause strange errors and sometimes leaves garbage directories or files lying around your disk.

Picture 1:
The main menu



To start playing now, click on “New Game”. A progress bar will pop up and tell you about the state of the load process. If you started Golgotha for the first time, it will build up the texture cache. This may take several minutes but must only be done once (as long as you do not change the rendering device to something special, like the software renderer). If you are running Golgotha in a window (for debugging purposes), it is currently required that you set your desktop bit depth to something above or equal to 16 or strange things might happen. On Unix, it might be necessary to set the bit depth exactly equal to 16. This ensures that at least 64000 colors can be displayed at once. If everything went fine, you find yourself inside your super tank. Just in front of you is your main base. Use the arrow keys and the mouse to move. Hint: Move quickly a bit forward from your current position or some supergun will hit you.

“Load” is for what it says: You can load old savegames. Hint: Savegames and level files are exactly the same. So if you want to load a level from the main menu, choose “Load” go up one directory (levels are in the main golg dir) and open the .level file you like.

“Save” lets you save the state of your game. Has no effect if no map is loaded.

“Network” Starts a network game. The network code is just being implemented, so perhaps you can already see something, perhaps it only crashes.

“Options” (Windows only) Lets you configure the graphics mode and your sound-card settings. You can also get some information about your system. If you’re running Unix, the only way to adjust the configuration is to manually edit the golgotha.ini file.

“Quit” No! You don’t want to use this... Hint: Quits immediately, no questions!

3.2 Controls

You can view or edit the complete list of keyboard shortcuts by opening “resource/keys.res”. The format of this file is described there.

3.2.1 Global keys

These keys work almost always. Exceptions are [should be] stated clearly.

ESCAPE – Main Menu

This key takes you back to the main menu to save a game or load another game or quit Golgotha. Hint: There’s currently an almost-invisible option in the main menu just above the “New Game” that says “Return to game”. Does not work in editor modes.

F1 – Help

Get some help (if available)

F2 – Screen Shoot


Take a screen shoot. You shoots will be saved as shootxxx.tga in the main dir. (xxx will be a number monotonically increasing)

F3 – Demo

Load the demo level (sometimes useful if other load methods fail)

F4 – Record

Toggle input recording. (undocumented feature...)

 I need to check this one out

Alt+F4 – Quit

Takes you immediately back to your OS.

F5 – Sky

Change the sky of the current level. This is actually an editor feature, but works everywhere.

F6 – Level Editor

Change to the level editor. This key brings you directly to the level editor with the current map loaded (if any). See the chapter about the editor to get more info about this huge feature. Press F6 again to get back to the game. When entering the editor, the game is automatically paused, but you can unpause it by pressing the “p” key. You could even play the game in this mode, but it’s probably a bit difficult because some of the key bindings are now quite different from the game modes.

F7 – Profile Window

Opens a window where you can see which parts of the engine use how much percent of the available CPU time. If one of the bars is significantly larger than you would expect, some optimization may be required.

F8 – Cheat

No! That’s not fair!

F9 – Music

Choose the music you want to hear. Selecting a different sound track has obviously no effect if either the sounds are not installed at all or the sound output is disabled/unavailable.

F10 – Maxtool

Opens up maxtool, the object editor. This has its own chapter, too. Press F10 again to get back to the game. The object editor runs independently from the rest of the engine, so any change made to an object will not be visible in the game itself until you reload the current level.

F11 – Lisp Interaction window

Did you know that Golgotha has its own, full-featured lisp engine running in the background? This window lets you access it directly. More info about this can be found further down.

Alt-F11 – Debug Output

Opens a small window where you can see the console output of Golgotha. Useful if the game is not connected to a debugger or a console.

3.3 The Game Modes

As *action mode*, we call it when you are inside your supertank and heading for targets.

The *follow mode* is similar to the action mode, the only difference is that you see your tank from behind. Press m to get there.


The *strategy mode* is the mode in which you see the map from above, with the radar and the build buttons on the right side of the screen. Press TAB or BACKSPACE to see this (you will use this mode very often, so make yourself familiar with its possibilities)

3.3.1 Action Mode

Use the mouse to look around. You'll notice a border around objects if you point the arrow at them. The bar above them indicates the remaining health of the object. If its getting red, it will explode soon.

If you click the left mouse button, you shoot with your main gun. If you're pointing at some of your own objects, it is also selected, if this is possible. You can drag rectangles to select more objects. The borders around the objects show that they're selected.

Press the right mouse button to fire your secondary weapon (usually the machine gun). If you have selected something, the mouse cursor may change its shape to indicate what will happen if you press the right mouse button. Usually, the selected units will try to find a way to the given location. If you selected a supergun (those great big-berta-like stationary weapons on the hills), the command will be "fire there". So click right and make your way hell out of there! Yes, nuke missiles *are* deadly.

 Hint: You always fire in the direction of the crosshairs, not the mouse pointer!

You can also select and command the supertank itself like this. Try it out: Select yourself (in action mode, you can just left click on the tube over your head) and click the right button somewhere far away (but not on the sky, of course).

To deselect everything, drag a small rectangle where there's nothing to select.

If you are in action mode, eventually red or yellow arrows will pop up indicating that the object is in range for the main gun (red) or the rocket launcher (yellow). Press ENTER to fire your guided missiles at your enemy. This is your most powerful weapon, but you usually only have two of them.

When you run out of ammo, you need to return to the base. If you stay inside your headquarter, your ammo and health are slowly refilled.

3.3.2 Follow Mode

The follow mode is quite similar to the action mode, except that you see your supertank from behind. Press “M” to switch to this mode. Press the same key again to return to action mode.

3.3.3 Camera Modes

Several cameras have been placed at interesting locations around the map. You can press the “C” key to check these locations. Which camera is used is determined dynamically depending on the happenings in their range.

3.3.4 Strategy Mode

Press TAB to switch to the strategy mode. You can now see the entire terrain from a top-down view. This is the preferred mode for commanding other units. Click on one of the build buttons to send a new unit against your foe. Clicking on the paths in the radar view chooses which path your newly created units should take. Ground paths are white, aerial paths are pink. Hint: You cannot change the path of an unit after starting it. You can “unlink” an unit from its path by selecting it and sending it somewhere else by right clicking the mouse in the main view. Click the right mouse button in the radar view to reposition the main view. Or use the arrow keys to move around. You can also press Alt+Up and Alt+Down to zoom in and out.

Press H to open a small radar window which can also be used in action mode.

You can always use the keyboard shortcuts to build your army. See the section about the unit types for a description of each of the keys.

Developer Information

This chapter describes some of the main features of the Golgotha engine source code distribution. Read on if you intend on participating in the project or want to write your own game on top of the golg (short for Golgotha) or i4 core

4.1 General information about the source

The source is partitioned in several groups. Each group serves its own purpose. The .h files of each group are in a subdirectory having the same name as the group. The corresponding .cpp files are always directly in the Golgotha main directory but have a name that starts with the group name followed by two underscores. So the implementation of init/init.h is in init__init.cpp. The main reason for this ugly scheme is that msvc (unlike gcc) doesn't like to find its source files all over the place. The exception from this rule makes the golg group: Those .h files are in the main directory.

Oh, and before I forget: For the sake of mother nature, do not try to print out the whole source, ok? It's more than 190'000 lines of code, which would use approximately 2300 sheets of paper! But *do* make backup copies of your work (on disks or cds I mean, not hardcopies, since you do not want to retype everything if your hard disk crashes, do you?).

A documented and annotated version of the source will be available online. That one is built using doxygen (a source documentation program) and is a good source of information. It can be used to search for identifiers, find cross-references (which functions all reference this one? What happens if I change this interface?) or get a graphical class hierarchy. If you've installed doxygen yourself (available from sourceforge) you can also create the documentation yourself. The necessary "Doxygen" file is included in the distribution. I highly recommend using an unix-based system to build the docs, because windows-systems seem to be much slower here (The reason being that windows needs much more time to create new processes; And a lot of them are spawned by doxygen – this is by design). Hint: If you have the idea that you could also build the latex version of the docs – forget it. The standard latex distribution gives an out-of-memory error at around page 6200. I couldn't even get it to compile when I extended the memory allocation a whole lot. I doubt that a paper or pdf manual of this size will be handy for some purpose.

4.2 Using the subversion client (svn)

We use subversion (<http://subversion.tigris.org>) for version control. The full documentation can be found at <http://svnbook.red-bean.com>. This chapter gives a brief overview of the most used commands.



Important: Inside the entire source (and therefore also in this manual), you must always use unix path name syntax. This applies to user dialogs and #include directives. The engine uses the correct slashes depending on the target system.

[To be filled in when the setup is complete]

4.3 The i4 core

Golgotha has its own programming library that extends the features of the standard C ANSI headers. These components are called the i4 core. They make golg independent of STL or other external libraries. Most of the names of the functions in the core begin with i4, so that they can be identified quickly. These functions can easily be used for other projects. Most of these functions can also be used in command-line based applications (except for the windowing subsystem, of course).

The different parts of the core are described bellow. Together with the dependencies, I also state the files that belong to the given group and the importance. Note: The dependencies do not mention the very basics (error.h, arch.h and others) nor any side dependencies that might easily be gotten rid of.

4.3.1 The platform specifications and the data types

To add or manipulate the settings of your platform, open arch.h. There you can state the details (i.e. the data sizes) of your platform and/or compiler. Of course, you should manipulate this file in such a way that declarations for existing compilers are not affected.

Internal Name: Architec- ture definition file Files: arch.h, num_type.h Importance: required Dependencies: None allowed.
--

When coding with Golgotha, always use its own data types: w8, w16, w32 for unsigned word values, sw8, sw16 and sw32 for signed word values of the respective size. This ensures that the code can easily be ported to other platforms. Arch.h *must* ensure that these data types have the given size. If you need it, there's also w64 for very long integer values. Unfortunately, many parts of the source have been written before anybody thought of 64bit architectures. Therefore pointer-to-integer conversations can be found at various places and it's probably not (yet) possible to compile and run the game on a 64bit architecture. Since I cannot afford one, I have no way of testing this. Any help would be appreciated.

The floats are called i4_float, but since floating point operations are standardized, every compiler should see them equal, anyway. Float values use a size of 32 bits.

Arch.h also contains a flag that indicates whether the current machine is little or big endian. This setting is needed to use the correct translations when reading binary data, since all binary files must be portable across platforms. As a rule of thumb, all Intel-architecture based CPUs (Intel, AMD, IBM) are little-endian and all others are big-endian.

4.3.2 The initialization classes

The class i4_init_class is responsible for initialization of all instances of classes that derive from it.

Internal Name: Init Files: init__init.cpp, init/init.h Importance: required for all applications Dependencies: None
--

The value that init_type() of some instance returns will be used to order the initialization of the objects. The lower the number, the earlier the object gets its init() method called. Objects can rely upon the fact that other objects with lower

`init_type()` have been initialized before, but nothing can be said about objects with the same priority. The uninitialization is done in reverse order. No object except the memory manager itself is allowed to have priority `I4_INIT_TYPE_MEMORY_MANAGER`.

See the source for many examples on how to use this class. The names of the order constants can be seen as hints to what they should be used for. If you know why, you can use them in other contexts, too.

```
// these type numbers determine the order i4_init_class'es
// are initialized in
// deinitialization occurs in the reverse order
enum {
    I4_INIT_TYPE_MEMORY_MANAGER,    // main i4 memory manager
    I4_INIT_TYPE_PRIORITY,          // Anything that only relies on mem-
man.
    I4_INIT_TYPE_THREADS,           // initialized thread info
    I4_INIT_TYPE_LISP_MEMORY,        // for lisp object allocations - uses
i4 memory manager,
    I4_INIT_TYPE_LISP_BASE_TYPES,    // adds lisp types into the system
    (li_int.. etc)
    I4_INIT_TYPE_LISP_FUNCTIONS,     // adds lisp functions (li_load & any
user li_automatic..
    I4_INIT_TYPE_STRING_MANAGER,
    I4_INIT_TYPE_FILE_MANAGER,
    I4_INIT_TYPE_DLLS,
    I4_INIT_TYPE_BEFORE_OTHER,
    I4_INIT_TYPE_OTHER,
    I4_INIT_TYPE_AFTER_ALL           // For anything that relies upon oth-
ers
};

class i4_init_class
{
public:
    static i4_init_class *first_init;
    i4_init_class *next_init;

    virtual int init_type() { return I4_INIT_TYPE_OTHER; } //overwrite this
to determine dependencies

    virtual void init() {}
    virtual void uninit() {}

    i4_init_class();
    virtual ~i4_init_class();
};

// should be called at the beginning of main
void i4_init();

// should be called at the end of main
void i4_uninit();
```

The following is an example of the use of this class:

```
class i4_stream_wav_self_reference_unloader:public i4_init_class
{
public:
    virtual void uninit()
    {
        if (self_references.size()!=0)
        {
            i4_warning("Warning: Stream wav player ref-
erences leaked.");
        }
        self_references.uninit();
    }
    //uninit before mem-man, but after sound manager.
    virtual int init_type(){return I4_INIT_TYPE_PRIORITY;};
};
```

```
}i4_stream_wav_self_reference_unloader_instance;
```

The most interesting part is the last line: We directly declare an instance of this type. This alone suffices that the `init()` and `uninit()` method of this class are called at startup and/or shutdown. (The example uses only the `uninit()`-call). Because we directly instantiate classes, such declarations need to be placed in `.cpp` files and not in the headers!

Be aware that the constructor of this class is executed just at the beginning of the program, even before `main()` is called. Therefore, you should avoid doing anything else than initializing member variables – particularly, you should not allocate any memory here or even call library functions.

Internal Names: device,
kernel
Files: device/*.*, de-
vice__device.cpp
Importance: Always
required
Dependencies: Init.

Picture 2:
A small part of the
UML diagram of the
descendants of
`i4_event_handler`

4.3.3 The message passing kernel

The engine has its own message passing subsystem (similar to the one implemented in windows). It mainly serves to dispatch user input to the windows that require them. Devices – objects that generate events - can be registered in the kernel. There are usually at least two devices present in all systems: The input class (that gets keyboard and mouse input) and the time device (that can be

Fehler! Es ist nicht möglich, durch die Bearbeitung von Feldfunktionen Objekte zu erstellen.

asked to generate an event after a certain amount of time). All classes that generate or receive events must be descendant of `i4_event_handler_class`. A very small part of the event handler class hierarchy is shown in the picture. Only the most important descendants are shown. You can see the complete hierarchy by pressing the “Derived Classes” toolbar button in Visual Studio. This only works if the browse info file was generated during the build (very recommended). I also recommend activating the “Browse” Toolbar. Unfortunately, this feature is unavailable in Visual Studio .NET. If you’re using that one, you can use the doxygen-generated docs instead.

Internal Name: Memman,
memory.
Files: mem-
ory__malloc.cpp,
memory/malloc.h
memory/new.h
Importance: Required
Dependencies: Init

4.3.4 The Memory Manager

Golgotha can use its own memory manager. Instead of using `malloc()` and `free()`, you can use `I4_MALLOC()` and `i4_free()`. This feature is still present to help supporting platforms with bad runtime support. Preferably use “new” and “delete” now. Do not mix these functions! Be sure that any memory you allocated using `I4_MALLOC()` is freed before the memory manager shuts down. This especially applies to `i4_arrays`. Hint: Be sure to “#undef new” if you need to overwrite the new operator for some class, because there are preprocessor-overrides for new in the windows-version. They are used to help finding memory leaks.

Internal Name:
`i4_array<T>`.
Files: memory/array.h
Importance: A template
for your conven-
ience.
Dependencies: memory.

4.3.5 The Template classes: `i4_array<T>`

This is by far the most used template in the project. It implements a dynamically growing array of type `T`. The constructor of such an array takes two parameters: The initial size and the growth. To add an object to the array, call `add(T)` [which returns the index of the newly added element] or `add()` [which returns the address of the next free entry]. Use the operator[] to access elements. Important: If you need global `i4_arrays` or `i4_arrays` as part of some class that is globally in-

stantiated (i.e. a derivative of `i4_init_class`), you *must* use 0 as the size parameter, since the memory manager won't be ready when the constructor is first called. And you *must* call `uninit()` before the memory manager terminates (i.e. in the `uninit()` function of the owning class). If you don't keep to this rules, you'll receive errors saying that you were attempting to call `i4_free()` after `i4_uninit()` when you quit the game. If you specify 0 as the growth parameter, you will receive an error if you attempt to add more elements than the size indicates. For most cases, this is therefore not recommended.

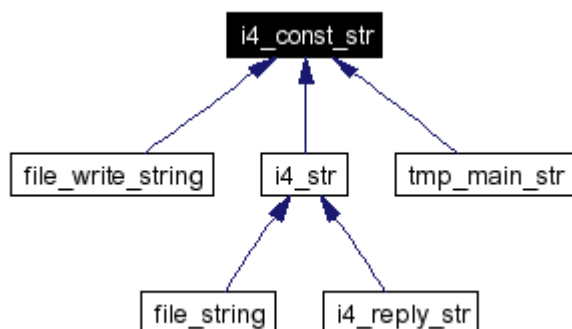
4.3.6 The Template classes: `i4_fixed_que<T>` and `i4_dynamic_que<T>`

These two templates define first-in-first-out queue structures. The fixed queue cannot grow, but is faster than the dynamic queue, since it doesn't need to be reorganized from time to time. The same precautions as for `i4_arrays` should be taken when using global instances of these classes.

4.3.7 The String classes

For most purposes, Golgotha uses its own string classes instead of C-Strings. This allows for greater flexibility and easier heap-management of string structures. The const strings are constant (who guessed that?), so they cannot be changed after creation. If you need a string that can be edited after creation, use `i4_str`. The edit control is a good example that uses these. The preferred way of

Internal Name:
<code>i4_fixed_que<T></code> ,
<code>i4_dynamic_que<T></code>
Files: <code>memory/que.h</code> ,
<code>memory/dynque.h</code>
Importance: Templates
for your conven-
ience.
Dependencies: <code>memory</code> ,
<code>i4_array</code>



Picture 3: `i4_const_str` and its offsprings. `i4_str` is by far the most important of these

passing strings (of either type) to functions is by reference, which allows creation of the strings on the stack. If possible, any function accepting a reference to an `i4_const_str` should also accept a const reference like in the following example:

```
char *i4_os_string(const i4_const_str &name, char *buffer, int buflen);
```

This function accepts as first argument a const reference to an `i4_const_str` object. Since this reference is const and `i4_const_str` has an implicit conversion from `char*` (More precisely, a constructor that takes a `char*` argument), one can also supply a c-type string. BTW: The above function is used to convert the string class types back to so called os-strings (since they are os-dependent). It can be used whenever you need C-Type `char*` strings instead of class-type strings. Be sure that the buffer size is large enough for the string that you want to copy. Although this function is not vulnerable to buffer-overflow attacks, you do not want a path to be shortened by it, do you? Important recommendation: For all platforms, file paths should support at least 256 characters.

Internal Name: <code>i4_str</code> ,
<code>i4_const_str</code>
Files: <code>string/string.h</code> ,
<code>string__string.cpp</code>
Importance: Required
Dependencies: <code>memory</code> ,
<code>kernel</code>

The `i4_str` class is very similar to `std::string` of the standard template library. It has almost the same methods as that one. The main reason we don't use `std::string` is that the STL has sometimes huge portability problems and an independent implementation was therefore easier to handle.

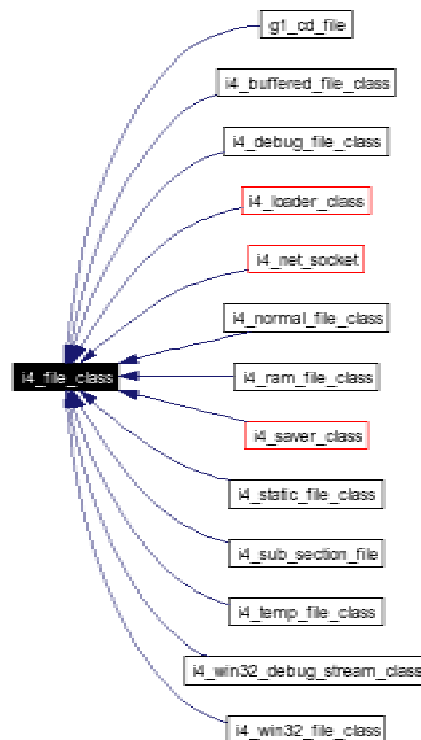
The other shown child classes of `i4_const_str` are deprecated.

4.3.8 The File Management Functions

Internal Name: file,
file_manager
Files: file/*, file__file.cpp
Importance: Required for
every I/O operation
Dependencies: string

This chapter describes the functions used for reading and writing data to the disk. It is highly recommended to always use these functions instead of the OS-supplied `_open()` and `_close()` functions (from the C runtime library). The main advantage of this is that this class is highly portable. Only a handful of lines need to be changed (using some `#if` directives) to support a new platform. Another advantage of using a class-pointer as file handle is that you can feed a function with any type of file. This allows for instance to pass memory-files to functions which usually take disk files (i.e. to encode an image file to memory for further processing). Even the network code uses pseudo-files as source or target for the data streams. You could be compressing an image file to a network stream and uncompress it on the other side – just as you would for a file stored on the disk.

An `i4_file_class` (an abstract class) is used to access files. A pointer to an instance can be obtained by opening the file. It is opened through `i4_file_manager_class::open()`. That function can be abbreviated by calling `i4_open()`, which scans through all file managers that are currently active to check whether one of them can open the given file.



Picture 4: A part of the class diagram of `i4_file` class.

A file manager defaults to going through the operating system's `open()`, but like in the unix model other file managers can be mounted on top of directories. When a file is accessed in this directory, the open call will be transferred to this file manager. This allows for redirection, tar-like-files, memory-files, and other goodies.

In most cases, you will just call `i4_open(filename, open_flags)` to open a file. This function returns a pointer to an `i4_file_class` which is actually a pointer to some subclass (like `i4_win32_file_class`) but you don't have to worry about this. You can now use the different `read()` or `write()` methods, depending on the access mode you specified. To close the file, just delete the pointer. This

must not be forgotten! The specialized subclasses `i4_loader_class` (and its derivative `g1_loader_class`) and `i4_saver_class` (with `g1_saver_class`) are used to load

or save “directorial” files. Such files have an internal directory structure that is handled internally by this classes to allow direct access to any data chunk stored in the file. It’s also possible to include version info, such that upon loading one can check whether a specific chunk version can be loaded by the actual loader version. See the object classes for huge usage of these. A short description of the `i4_file_class` (in `file/file.h`) follows.

```
#ifndef __I4FILE_HPP
#define __I4FILE_HPP

#include "arch.h"
#include "isllist.h"
```

The following structure is used in conjunction with the directory functions. This allows highly efficient directory scans.

```
enum { I4_FILE_STATUS_DIRECTORY=1};
struct i4_file_status_struct
{
    w32 last_modified;
    w32 last_accessed;
    w32 created;
    w32 flags;
} ;

class i4_file_class
```

```
{
public:
```

These are the core functions that will be overwritten by a specific implementation of the file class. Of course, a ram file will have different read functions than a disk file or a network stream.

```
virtual w32 read (void *buffer, w32 size)          = 0;
virtual w32 write(const void *buffer, w32 size)    = 0;
virtual w32 seek (w32 offset)                     = 0;
virtual w32 size ()                               = 0;
virtual w32 tell ()                               = 0;
```

```
i4_bool eof() { return tell()==size(); }
```

The following functions support asynchronous reads and writes. Just call `async_read()` with the target buffer, the amount of data to be read and a pointer to the callback function. Optionally you can add a context structure pointer which will be passed to the callback as-is. This can be usefull if you want to transmit i.e the file pointer or the pointer to the buffer to the callback function. The priority defines how fast you need your data. A low value (<100) will mean you want low delay and high accuracy. Your callback function must not waste any time if you do this (i.e doing huge calculations there for low priority values is bad). The callback function is called as soon as the data has been read from the disk.

```
typedef void (*async_callback)(w32 count, void *context);

// returns i4_F if an immediate error occurred
virtual i4_bool async_read (void *buffer, w32 size,
                           async_callback call,
                           void *context=0, w32 priority=255);

virtual i4_bool async_write(const void *buffer, w32 size,
                           async_callback call,
                           void *context=0);

// abort current operation
virtual void abort() {}
```



Warning: The callback function will be called from a different thread. So be shure all your calculations there are thread-safe!

These functions make life much easier, and you will be using them a lot. Actually, you *must* use the `read_16()/write_16()` and `read_32()/write_32()` pairs if you can, because that's the only way to assure that the code stays portable: These functions automatically convert words and integers to the corresponding endianness for the local architecture. All files are stored in little-endian (Intel) format, so for windows and linux, no conversion is needed, but for sun, all values will be inverted, hopefully without the user ever noticing the difference.

```
// Nice Read Methods
w8      read_8    ();

w16     read_16   ();

w32     read_32   ();

float   read_float();

i4_str* read_str(w32 len);
i4_str* read_counted_str();
```

```
// C++ stream operators
```

I newer use these as I don't like this notation, but if you prefer the stream-like syntax...

```
i4_file_class& operator>>(w32& v)   { v = (w32) read_32(); return *this; }
i4_file_class& operator>>(sw32& v)  { v = (sw32)read_32(); return *this; }
i4_file_class& operator>>(w16& v)   { v = (w16) read_16(); return *this; }
i4_file_class& operator>>(sw16& v)  { v = (sw16)read_16(); return *this; }
i4_file_class& operator>>(w8& v)    { v = (w8)  read_8();  return *this; }
i4_file_class& operator>>(sw8& v)   { v = (sw8) read_8();  return *this; }
i4_file_class& operator>>(float& v) { v = read_float();   return *this; }

// Nice Write Methods
w32      write_8    (w8 num);

w32      write_16   (w16 num);

w32      write_32   (w32 num);

w32      write_float(float num;

w32 write_str(const i4_const_str &str);
w32 write_counted_str(const i4_const_str &str);

// same as fprintf, but with the addition %S is a i4_const_str *
int printf(char *format, ...);

// write_format takes a different set of % symbols than the typical
printf to elimante
// confusion with sizes and be easily usable with read_format
// write_format("124fS", &a_w8_var, &a_w16_var, &a_w32_var,
//                    &a_float_var, &a_i4_const_str_pointer);
int write_format(char *format, ...);

// format is same as write_format
int read_format(char *format, ...);

// C++ stream operators
i4_file_class& operator<<(w32 v)   { write_32(v);      return *this; }
i4_file_class& operator<<(sw32 v)  { write_32((w32)v); return *this; }
```

```

i4_file_class& operator<<(w16 v) { write_16(v); return *this; }
i4_file_class& operator<<(sw16 v) { write_16((w16)v); return *this; }
i4_file_class& operator<<(w8 v) { write_8(v); return *this; }
i4_file_class& operator<<(sw8 v) { write_8((w8)v); return *this; }
i4_file_class& operator<<(float v) { write_float(v); return *this; }

virtual ~i4_file_class() {}
};

```

Check the source for examples if you need odd combinations of access specifiers. (i.e for appending and reading) Use `I4_NO_BUFFER` if you will be reading the entire file to memory in one single operation anyway. This bypasses the internal cache manager.

```

// open flags
enum { I4_READ=1,
       I4_WRITE=2,
       I4_APPEND=4,
       I4_NO_BUFFER=8,
       I4_SUPPORT_ASYNC=16 // this flag is needed if you intend to call
       async_read/write
};

```

What follows are the portable equivalents to the known operating system commands. Be sure to always use these instead of those of the runtime library for portability reasons.

```

// returns NULL if unable to open file
i4_file_class *i4_open(const i4_const_str &name, w32 flags=I4_READ);

// return i4_F on failure (will delete the given file)
i4_bool i4_unlink(const i4_const_str &name);

// returns i4_F if file does not exist (to get file date, size etc)
i4_bool i4_get_status(const i4_const_str &filename,
                     i4_file_status_struct &return_stat);

// return i4_F on failure
i4_bool i4_mkdir(const i4_const_str &path);
i4_bool i4_rmdir(const i4_const_str &path);
i4_bool i4_chdir(const i4_const_str &path);
i4_bool i4_copy_file(const i4_const_str &source, const i4_const_str
&dest);

struct i4_directory_struct
{
    i4_str **files;
    w32 tfiles;

    i4_str **dirs;
    w32 tdirs;

    i4_file_status_struct *file_status; // array of file stats correspond-
ing to above files

    i4_directory_struct() { tfiles=tdirs=0; files=dirs=0; file_status=0; }
    ~i4_directory_struct();
};

// returns i4_F if path is bad (tfiles and tdirs will be 0 as well)
// you are responsible for deleting both the array of strings and each
string in the array
// file_status is a pointer to an array of file_status's that will be
created, you
// must free these as well. status may be 0 (default), in which case no
array is created
class i4_status_class;
i4_bool i4_get_directory(const i4_const_str &path,
                        i4_directory_struct &dir_struct,
                        i4_bool get_status=i4_F,
                        i4_status_class *status=0);

```



```

struct i4_filename_struct
{
    char path[256];
    char filename[256];
    char extension[256];

    i4_filename_struct() { path[0]=0; filename[0]=0; extension[0]=0; }
};

// returns i4_F if path cannot be split
i4_bool i4_split_path(const i4_const_str &name, i4_filename_struct
&fname_struct);

// return 0 if full path cannot be determined
i4_str *i4_full_path(const i4_const_str &relative_name);

// returns the shortest relative path
i4_str *i4_relative_path(const i4_const_str &path);
#endif

```

4.4 The Graphics Subsystem

4.4.1 The video engine

Internal Name: video,
display
Files: video/* video*.cpp
Importance: Required if
graphic output is
needed.
Dependencies: core

The video engine is responsible for all 2D components visible on the screen. It is also responsible for initialization and shutdown of the entire windowing subsystem. A subpart of this will also translate events from the operating system to Golgotha events. For instance the windows message callback function is a part of the display class for windows. Of course, X11 (unix) has different functions there. Currently, Windows uses DirectX for its graphic subsystem and linux uses X11 and OpenGL. Since the operating system dependent message loop is highly coupled to the main window, it is the same class that will receive the events from the OS and create the game's main window (`x11_input_class` or `win32_input_class`). As main window we refer to the game window as it is seen by the operating system. It usually has a title bar and a frame that looks like the rest of the windows on your screen. On some platforms, the main window can't be seen as such, because the game might run in full screen mode.

4.4.2 The rendering engine

Internal Name: render
Files: render*
Importance: For 3D apps
only
Dependencies: core,
video

These are the 3D components of the game and only required if such behaviour is needed (for Golgotha meaning always). These components are usually hardware-accelerated, but a software rendering device is available. For most platforms, you won't need those, because in most cases the underlying accelerator driver will emulate all behavior that cannot be hardware accelerated on a given system. This means that for instance under DirectX, Golgotha will use the RGB-Emulation device if no HAL is available. Under some rare circumstances it might of course happen that the default choice is not the best. The options let you therefore choose which device/render settings to use. Under Windows, you can check which driver is currently active by going to the info page in the options dialog.

4.5 The Window Management

Golgotha has its own full-featured, highly customizable graphical user interface. Some of the available GUI elements are: Windows (with and without title bars), buttons, images, text edit controls and much more. The renderer itself uses a special kind of window for the output.

4.6 The Golgotha core

From here on, the supplied information is about Golgotha itself, no more the underlying generic graphics engine.

4.6.1 The objects

Internal Name: objects
 Files: golg* objs* objs/*.*,
 objects/*.gmod
 Importance: These
 make up the actual
 game.
 Dependencies: Every-
 thing else.

In this chapter, I will describe the functionality that makes up the things about the visible objects in the game. So in this chapter, an object might not only mean some part of memory for c++ programming (i.e a class) but some touchable object, like a tank or a building. All such objects are derived from `g1_object_class`, whose interface is shown below. This code can be found inside `g1_object.h`.

```
// Golgatha Generic game object class
//
// The g1_object_class is the base type for all building/ground objects
in the game.
```

```
#ifndef G1_OBJECT_HH
#define G1_OBJECT_HH
```

```
//... skipping unimportant stuff
```

A mini object is used as part of a large object. All parts that can be moved in respect to the master object should be modeled using mini objects. One example are the turret towers of tanks or wheels. See one of those for examples how to use this class. Consider that the position (x, y and h coordinates) are relative to the master object, and not relative to the world. The best way to find the local object's coordinates for the attachment of components is the use of mount-points.

```
//g1_mini_object stuff
class g1_mini_object
{
public:
    void draw(g1_draw_context_class *context,
              i4_transform_class *transform,
              g1_screen_box *bound_box,
              g1_player_type player,
              i4_transform_class *use_this_transform=0, //if you want to
supply a local space transform
              i4_bool pass_world_space_transform=i4_T, // if you don't
want lighting pass i4_F
              i4_bool use_lod_model=i4_F);

    i4_3d_vector offset; // tells where the center of mini_object
model is (where to rotate)
    i4_3d_vector loffset;

    i4_float x,y; // location in game map in game coordinates
    i4_float lx,ly; // last tick position (use to interpolate
frames)
```

```

i4_float h;                // height above (or below ground)
i4_float lh;               // last tick height

i4_3d_vector rotation;     // contains rotation information
i4_3d_vector lrotation;

gl_model_id_type defmodeltype; //model used by this mini object. can be
overridden with draw()
gl_model_id_type lod_model;

w16 frame;                // current animation frame
w16 animation;            // current animation number in model

void calc_transform(i4_float ratio, i4_transform_class *trans);
// calculates transform to parent system

void position(const i4_3d_vector &pos) { lx=x=pos.x; ly=y=pos.y;
lh=h=pos.z; }

void grab_old()
//be sure this is called by any parent object grab_old()'s
{
    lx=x;
    ly=y;
    lh=h;
    lrotation = rotation;
    loffset = offset;
}
};

```

A class for chaining objects together. Used mainly for enumeration.

```

class gl_object_chain_class
{
public:
    gl_object_class      *object;
    gl_object_chain_class *next;
    w16 offset;

    inline gl_object_chain_class *next_solid(); // defined after
gl_object_class
    gl_object_chain_class *next_non_solid() { return next; }
};

```

The following enumeration describes the kind of “dot” that appears on the radar for the given object if no radar image is supplied or no images are to be drawn (the small radar window doesn’t show images, whereas the default one in strategic view does). The value defines the size of the dot. The color will be the one of the corresponding player.

```

enum gl_radar_type
{
    G1_RADAR_NONE,
    G1_RADAR_VEHICLE,
    G1_RADAR_STANK,
    G1_RADAR_WEAPON,
    G1_RADAR_BUILDING,
    G1_RADAR_PATH_OBJECT
};

```

Each derivative of `gl_object_class` has an unique identifier. This is used to allow easier casting to the different subclasses if it is needed. The identifiers itself are created dynamically at load time of the engine, so they might change at random. Be sure to always obtain type identifiers using `gl_get_object_type()` at runtime instead of hardcoding them.

```

typedef sw16 gl_object_type;

```

```

class gl_object_class
{
protected:
    friend class gl_reference_class;
    friend class gl_map_class;
    friend class gl_map_cell_class;
    friend class gl_remove_manager_class;


```

Each time one changes something in the way an object is serialized (written to disk) for instance by adding additional members, a new version is introduced. To be able to read older save files and levels it is *absolutely required* to keep the old load code intact and use the correct function depending on the version word.


```

void load_v2(gl_loader_class *fp);
void load_v3(gl_loader_class *fp);
void load_v4(gl_loader_class *fp);
void load_v5(gl_loader_class *fp);
void load_v6(gl_loader_class *fp);
void load_v7(gl_loader_class *fp);
void load_v8(gl_loader_class *fp);
void load_v9(gl_loader_class *fp);

```

 If very large objects will be added to the map (entire building complexes), something here needs probably to be changed because if none of the cells the object is registered on is in the viewing frustum, the model is not drawn.

These lists are used for memory management purposes. This system works very well if handled correctly. Don't change anything in these `occupy_location()` and in the `unoccupy_location()` functions if you do not know *exactly* what you are doing. With a call to `occupy_location()` an object registers itself at the map location referenced by the `x` and `y` data members. With `unoccupy_location()` that reference is removed. Be sure that all objects are always registered after processing them, or you will lose them. You must call `unoccupy_location()` before moving an object and `occupy_location()` after. The different `occupy_location...` functions bellow are used for differently sized objects. An object can be registered in at most 4 cells of the map.

 Warning: This code has proven to be very stable. Don't change any of these protected functions unless you know really what you do!

```

// a list of all objects that reference us, so we
// can remove the reference when we delete ourself
i4_isl_list<gl_reference_class> ref_list;

// list of squares this object is on (max of 4), if
// value is 0xffff then it is not a reference
// the last one holds the center of the object, if necessary (ie, the
// object takes
// up more than 1 square and his corners are far enough apart)
i4_array<gl_object_chain_class> occupied_squares;

gl_object_chain_class *new_occupied_square()
{
    gl_object_chain_class *ret = occupied_squares.add();
    ret->object = this;
    return ret;
}

i4_bool occupy_location_model(const gl_model_draw_parameters&
draw_params);
i4_bool occupy_location_corners();
i4_bool occupy_location_center(); // only adds the center of the
object
i4_bool occupy_location_model_extents(const gl_model_draw_parameters
&draw_params);
public:

    virtual w32 private_safe_to_cast() const { return 0; }

```

This `global_id` is a very important member of the class. It can be used as replacement for the address of the object. It's advantage: It is (usually, see excep-

tions in the network section) valid everywhere and even on a remote pc. With `gl_global_id.get(w32 id)` you can get the corresponding object pointer. Use `gl_global_id.checked_get(w32 id)` if you are not sure whether the given id really corresponds to an object. This method will return 0 if the passed id was invalid. The other data members are explained in the comments.

```
w32 global_id;                // unique to the object, used to
reference via networking

gl_object_type id;            // this is the object's 'type' id

i4_float x,y;                 // location in game map in game
coordinates
i4_float lx,ly;               // last tick position (use to interpolate
frames)

i4_float h;                   // height above (or below ground)
i4_float lh;                  // last tick height

i4_float theta;               // facing direction in the x-y game plane
i4_float ltheta;              // last tick theta (yaw)

i4_float pitch;               // pitch (y-z)
i4_float lpitch;              // last tick pitch

i4_float roll;                // roll (x-z)
i4_float lroll;               // last tick roll

sw16 health;                  // the health its got left
```

Accessing lisp members of an object is a bit tricky (see examples), but it has the advantage that the values can directly be edited in the editor. You should consider using these for objects you usually place in the editor.

```
li_class *vars;               // lisp variables used for dialog's.
These load and save automatically.
```

The player this object belongs to. In single player mode, 0 is the neutral player – he usually owns trees and civil buildings, player 1 is the local human player, all others are computer players. The radar type is explained above. The radar image is a reference to some bitmap or NULL if no bitmap for this type of object exists or is needed (i.e. trees don't need to show up on the radar).

```
gl_player_type player_num;
gl_radar_type radar_type;
gl_team_icon_ref *radar_image;
```

Change the owner of this object. Called particularly for buildings that can be captured.

```
virtual void change_player_num(int new_player_num); // don't change
player_num directly
gl_team_type get_team() const;
```

These flags are very important. Some of them describe capabilities of the object (like HIT_GROUND, HIT_AERIAL) some are used to speed up some functions (if the THINKING flag is set, `request_think()` can return immediately, as it is sure that the object is already in the think list).

```
enum flag_type
{
    SELECTED      =1<<0,    // if object has been selected by player
    THINKING      =1<<1,    // if object is in a level's think list

    SHADOWED      =1<<2,    // if object should have shadows drawn for it
    MAP_INVISIBLE =1<<3,    // if object is invisible
    DELETED       =1<<4,    // if object has been removed, but remove has
```

```

not been processed yet
    IN_TUNNEL      =1<<5,    // if the object is in a tunnel

    SELECTABLE     =1<<6,    // if the user can select the object in the
game
    TARGETABLE     =1<<7,    // if object can be attacked

    GROUND         =1<<8,
    UNDERWATER     =1<<9,
    AERIAL         =1<<10,

    HIT_GROUND     =1<<11,
    HIT_UNDERWATER =1<<12,
    HIT_AERIAL     =1<<13,    // can attack objects in the air

    BLOCKING       =1<<14,    // if object blocks passage of other objects

    SCRATCH_BIT    =1<<15,    // use this for temporary calculations, but
set it back to 0 when done
    MAP_OCCUPIED   =1<<16,    // makes sure you don't call
occupy_location/unoccupy_location twice

    DANGEROUS      =1<<17,    // object should be killed

    RADAR_REMOVE   =1<<18,    // object needs to be removed from radar
    CAN_DRIVE_ON   =1<<19,    // if objects can drive over / stand on you
    ON_WATER       =1<<20,    // if objects is on water
    SPECIALTARGETS =1<<21,    // if object has special targets (i.e. own
units)
    EXT_GLOBAL_ID  =1<<22,    // uses special way to find global id
    USES_POSTTHINK=1<<23,    // Nonzero if post_think() method should be
called.

    //the following flags are particularly important in networking
mode
    NEEDS_SYNC     =1<<24,    // Nonzero if the object has thought
something that cannot be guessed

                                // on the remote machine
    LAST_SYNC      =1<<25,    // Has synced last frame (perhaps requires
resending)
    LOST_SYNC      =1<<26,    // Is known to have lost sync.
    NEWER_SYNC     =1<<27,    // This object cannot get synchronized as
the remote has different id.
};

enum { SAVE_FLAGS= SELECTED | THINKING }; //This mask is applied on
object creation from a file only

w32 flags;

```

These functions make it easy to set/reset a flag bit.

```

int get_flag(int x) const { return flags & x; }
void set_flag(int x, int value) { if (value) flags|=x; else
flags&=(~x); }
i4_bool selected() const { return (w8)flags & SELECTED; }
i4_bool out_of_bounds(i4_float x, i4_float y) const;
i4_bool moved() const { return (lx!=x || ly!=y); }
i4_bool valid() const { return get_flag(MAP_OCCUPIED)!=0; }

gl_object_definition_class *get_type();    // inlined below

void mark_as_selected()
{
    set_flag(SELECTED,1);
};
void mark_as_unselected()
{
    set_flag(SELECTED,0);
}

```

Kind of the “size” of the object.

```
virtual i4_float occupancy_radius() const;
```

```
// called when the object is on a map cell that is about to be drawn
//virtual void predraw() {}
```

The following function is called whenever the object needs to be drawn. In most cases, this just renders the corresponding model.

```
// called when the object is on a map cell that is drawn
virtual void draw(gl_draw_context_class *context);
```

```
// called for each object after everything else has been drawn and in
editor mode
```

```
virtual void editor_draw(gl_draw_context_class *context);
```

```
virtual void note_stank_near(gl_player_piece_class *s) { ; }
```

```
// called every game tick
```

The think() method is called every game tick for every object that is in the map's think list. You should use this method to do the AI of the object. One game tick is 1/10th of a second, regardless of the frame rate. For performance reasons, post_think() is only called if the USES_POSTTHINK flag is set. If the object should continue thinking, you must ensure that inside this method request_think() is called at least once. Dumb objects (like trees) don't need to think, but they must not do anything odd if the think method is called on them anyway.

```
virtual void think() = 0;
virtual void post_think() {};
```

The comment says the most important stuff about constructors. All constructors of derivatives of this class must have *exactly* this format. No default constructor must be supplied. If more parameters are required to create some object, a setup method is used (see the different bullet objects for examples about this). When loading from a file, the constructor should check for the version of the data given. See the implementation of this constructor.

```
//a constructor of an object creates an object of the given type
//if fp is nonzero, the object is loaded from the given file
gl_object_class(gl_object_type id, gl_loader_class *fp);
```

```
//a mostly empty function
virtual void validate() {}
```

Write the object to the given file. Should write a version tag.

```
//saves this object to the given file. Can safely assume fp!=0
virtual void save(gl_saver_class *fp);
```

The following method is used for network support. An object will synchronize its state to the given file stream. Below follow all the functions that can be overwritten by subclasses. The names should be self-explaining.

```
//does the same as the constructor, but doesn't reset the other data
//can safely assume fp!=0 and data_version==latest.
//an object that doesn't have a save() doesn't need this either.
virtual void load(gl_loader_class *fp);
```

```
//Skips the corresponding amount of data in the file stream
//used if the given object does not need to be synced
//i.e if we received a packet that contains older data than the last
//one for a particular object.
virtual void skipload(gl_loader_class *fp);
void request_think();
virtual void request_remove();
```

```
// call to add object to a map (adds to cell x,y where object is
standing)
```

```

virtual i4_bool occupy_location();

// call to remove an object from a map
virtual void unoccupy_location();

virtual i4_bool deploy_to(float x, float y, gl_path_handle ph)
{
    if (ph) gl_path_manager.free_path(ph);
    return i4_F;
}

virtual void damage(gl_object_class *who_is_hurting, //we got some
damage
                    int how_much_hurt, i4_3d_vector damage_dir);

virtual i4_bool check_collision(const i4_3d_vector &start,
                               i4_3d_vector &ray);

enum {NOTIFY_DAMAGE_KILLED=1};

virtual void notify_damage(gl_object_class *obj, sw32 hp) //informs
firing object that the target was hit
{};

virtual void calc_world_transform(i4_float ratio, i4_transform_class
*transform=0);
// calculates the transform from object coordinates to the world and
stores it in
// 'transform'. if transform is null, it stores it into the object's
internal storage

i4_transform_class *world_transform; //calculated and linearly
allocated at draw time

gl_model_draw_parameters draw_params;
gl_mini_object *mini_objects;
w16 num_mini_objects;

void allocate_mini_objects(int num, char *reason)
{
    mini_objects = (gl_mini_object
*) I4_MALLOC(sizeof(gl_mini_object)*num, reason);
    memset(mini_objects, 0, sizeof(gl_mini_object) * num);
    num_mini_objects = num;
}

```

This function sets the values for the last position (lx, ly, lh...) to the current ones. Should be called whenever an object is moved unexpectedly far.

```

virtual void grab_old(); // grab info about the current tick for
interpolation

// to be consistant, every gl_object should have an init
virtual void init() {}

virtual short get_max_health(); //use this instead of defaults-
>max_health
virtual short get_min_health() {return 0;} //returns usually 0, but not
always.
// show in editor if mouse cursor stays still on object
virtual i4_str *get_context_string();

virtual ~gl_object_class();

virtual i4_bool can_attack(gl_object_class *who) { return i4_F; }

virtual void stop_thinking(); // Slow, so don't use if you can help
it

float height_above_ground(); // calls map->terrain_height

char *name();

```

This function is used to send messages about nearby objects/special instructions to the object. It is also used to get the list of special commands a given instance

supports from the object. A good example of its usage can be seen in `objs__bomb_truck.cpp`.

```
virtual li_object *message(li_symbol *message_name,
                          li_object *message_params,
                          li_environment *env) { return 0; }

// called for every object when you click 'ok' in the vars edit dialog
for an object
virtual void object_changed_by_editor(gl_object_class *who, li_class
*old_values) {}
virtual int get_chunk_names(char **&list) { return 0; }
```

Moved these here to allow any object to be a factory under some circumstances (needed i.e. for `road_object`).

```
virtual gl_path_object_class *get_start()
{return 0;};
virtual void set_start(gl_path_object_class *start){};
virtual i4_bool build(int type)
{return i4_F;}; //can't build anything
virtual w32 get_path_color()
{return 0xffa00000;};
virtual w32 get_selected_path_color()
{return 0xffff0000;};
};

inline gl_object_chain_class *gl_object_chain_class::next_solid()
{
    if (!next || !next->object->get_flag(gl_object_class::BLOCKING))
        return 0;
    else return next;
};

class gl_object_definition_class;

// this is the call to add a new object_type to the game.
gl_object_type gl_add_object_type(gl_object_definition_class *def);

// this is the call to find the object_type of a specified object
gl_object_type gl_get_object_type(const char *name);
gl_object_type gl_get_object_type(li_symbol *name);

// remove probably doesn't need to be used during a normal game, but can be
useful for
// adding and removing a dll during a running session instead of
restarting the game
void gl_remove_object_type(gl_object_type type);
```

This class defines an object to golgotha, primarily how to create such an object and load it from disk. The object then has virtual functions that allow it to do things like think, save, and draw. Objects can be created at any time in the game, and should be able to be linked in through dlls or defined in lisp code.

```
class gl_object_definition_class
{
protected:
    typedef void (*function_type)(void);
    char *_name;
    function_type init_function, uninit_function;

    gl_damage_map_struct *damage; // loaded from
    scheme/balance.scm

public:
```

Don't mix these flags with the one for the object instances. These are for the instances of the types. Exactly one instance per object type is present in the system at a given time.

```
enum
{
    EDITOR_SELECTABLE = (1<<0),      // object shows up in editor
    DELETE_WITH_LEVEL = (1<<1),      // delete this type with the map
    MOVABLE            = (1<<2),      // objects
    TO_MAP_PIECE       = (1<<3),      // These define groups for casting
    TO_PLAYER_PIECE    = (1<<4),
    TO_DYNAMIC_OBJECT  = (1<<5),
    TO_PATH_OBJECT     = (1<<6),
    TO_DECO_OBJECT     = (1<<7),
    TO_FACTORY         = (1<<8),
    TO_BUILDING        = (1<<9),
    HAS_ALPHA          = (1<<10)      // if object has alpha polys it will
draw after non-alpha objects
};

w32 var_class;    // class type to create for
object's vars
li_class *vars;   // variables specific to the
type
gl_object_defaults_struct *defaults; // loaded from
scheme/balance.scm

gl_damage_map_struct *get_damage_map();

w32 flags;
int get_flag(int x) const { return flags & x; }
void set_flag(int x, int value) { if (value) flags|=x; else
flags&=~x; }

gl_object_type type;

gl_object_definition_class(char *_name,
    w32 type_flags = EDITOR_SELECTABLE,
    function_type _init = 0,
    function_type _uninit = 0);

virtual ~gl_object_definition_class() { gl_remove_object_type(type); }

// create_object should return a new initialized instance of an object,
if fp is null then
// default values should be supplied, otherwise the object should load
itself from the file
virtual gl_object_class *create_object(gl_object_type id,
    gl_loader_class *fp) = 0;
```

The name of an object should be unique, so you might want to be creative, the name is used to match up the object types in save files since object type id's are dynamically created at runtime. This name *must not* change after the object's type is instantiated for the first time. This means that these names should be supplied once and newer changed thereafter.

```
const char *name() { return _name; }

i4_bool editor_selectable() { return (w8)flags & EDITOR_SELECTABLE; }

virtual void init();
virtual void uninit() { if (uninit_function) (*uninit_function)(); }

virtual void save(gl_saver_class *fp) { ; } // save info about type
virtual void load(gl_loader_class *fp) { ; } // load info about type

// this is called when the object is double-clicked in the editor, the
dialog is added to a draggable frame
virtual i4_window_class *create_edit_dialog();
```

```

};

void gl_apply_damage(gl_object_class *kinda_gun_being_used,
                    gl_object_class *who_pulled_the_trigger,
                    gl_object_class *whos_on_the_wrong_side_of_the_gun,
                    const i4_3d_vector
                    &direction_hurten_is_commin_from);

// increase this number if you change the load/save structure of
gl_object_class
#define G1_OBJECT_FORMAT_VERSION 1

// this table has an array of pointers to object definitions
// this is used by the border frame to find object with build info so it
// can add buttons for them
extern gl_object_definition_class
*gl_object_type_array[G1_MAX_OBJECT_TYPES];
extern gl_object_type gl_last_object_type; // largest object number
assigned

// this is preferred way to create new object in the game
inline gl_object_class *gl_create_object(gl_object_type type)
{
    if (gl_object_type_array[type])
    {
        gl_object_class *o=gl_object_type_array[type]-
>create_object(type, 0);
        o->request_think();
        //cannot do this from here as we don't want to sync _every_
object.
        //This particularly applies to explosion objects which will be
//generated separatelly on every system (with different IDS,
unfortunately).
        //o->set_flag(gl_object_class::NEEDS_SYNC,1);
        o->set_flag(gl_object_class::NEWER_SYNC,1); //All objects created
solely with this
        //method will be created automatically by the remote machine
using
        //a different id. (Because we exspect the remote objects to
think
        //exactly like the local ones)
        return o;
    }
    else
        return 0;
}

inline gl_object_definition_class *gl_object_class::get_type() { return
gl_object_type_array[id]; }

void gl_initialize_loaded_objects();
void gl_uninitialize_loaded_objects();

#endif

```

A new object type is usually created using the following statement (in an implementation file, not in the header). The example is the statement that binds the helicopter object into the system.

```

gl_object_definer<gl_helicopter_class>
gl_helicopter_def("helicopter",
    gl_object_definition_class::TO_MAP_PIECE |
    gl_object_definition_class::EDITOR_SELECTABLE |
    gl_object_definition_class::MOVABLE,
    gl_helicopter_init);

```

The “<gl_helicopter_class>” template instantiation refers to the class with the same name. This is a derivative of `gl_object_class` (actually `gl_map_piece_class`, which is itself a child of `gl_object_class`). The “helicopter” string constant is the internal name of the object. As stated above, this string must not change after it

is supplied once. There follow type flags. This object is a derivative of `gl_map_piece_class` (most player unit types are) so it can be cast “TO_MAP_PIECE”. “EDITOR_SELECTABLE” means that the object appears in the “Object types” list inside the editor and can therefore be placed manually. Most objects will have this flag set. “MOVEABLE” means that the object can move, but the semantics of this flag is somewhat vague. As last (and optional) parameter we can add a reference to a function that is called when the object type (not each instance) is about to be initialized.

4.6.2 The Network support functions

After the initialization of the network subsystem (one server, multiple clients), the game does the following: Each time an object knows that its think method has done something that the remote system cannot guess, his data is added to a network packet and sent to the server (or all clients) where it is unpacked.

Look at the following code from the `gl_map_class::think_objects()` method:

```
i = 0; //start_tail;
//h = start_head;
for (; i < h; i++)
{
    o = think_que_dyn[i];

    //always check to make sure the pointer
    //is good. objects might have removed themselves
    //from the map while still in the que, and left
    //a null pointer in their place
    if (o)
    {
        li_this = o->vars;
        o->set_flag(gl_object_class::THINKING, 0);
        o->think();
        if (!o->get_flag(gl_object_class::USES_POSTTHINK))
        {
            think_que_dyn[i] = 0; //because of cache strategies, it's
better to
            //do this here
        }
#ifdef NETWORK_INCLUDED
        if (rtfp)
        {
            if (o->get_flag(gl_object_class::NEEDS_SYNC))
            {
                w32 sizebefore = rtfp->tell();
                rtfp->write_32(o->global_id);
                rtfp->write_16(o->id);
                o->save(rtfp); <<HERE
                if (rtfp->tell() >= MAX_PACKET_SIZE - 4)
                {
                    rtfp->seek(sizebefore);
                    rtfp->write_32(0);
                    i4_network_prepare_command(G1_PK_PROCESS);
                    rtfp->write_32(o->global_id);
                    rtfp->write_16(o->id);
                    o->save(rtfp);
                }
                o->set_flag(gl_object_class::NEEDS_SYNC, 0);
                o->set_flag(gl_object_class::LAST_SYNC, 1);
            }
            //don't propagate the deletion of non-synced objects
            //as this might propose deletion of some object that is
not
            //really the one that is expected.
            //??? Well, no: It should not be possible that an id has
two
            //different meanings on two systems, but that the same
```

```

object
    //on two systems have different ids.
    if (o-
>get_flag(gl_object_class::DELETED&gl_object_class::NEEDS_SYNC))
        { //Be shure that any deletions will be consistent
          rtfp-
>write_32(gl_global_id_manager_class::ID_DELETEPROPAGATE);
          rtfp->write_32(o->global_id);
        };
    }
#endif
}
}

```

If after thinking, the NEEDS_SYNC flag is set, the object is saved to the stream. The supertank is the only object that always sets this flag for the local player. It is automatically set for objects created in a factory. The LAST_SYNC flag could probably be used to make sure that important synchronization packets (i.e. the creation of an object, which is usually only sent once) will propagate through the entire network and the engine can detect lost data.

The biggest problem is that with this simple scheme, all objects that need to be created should be synchronized over all systems. Unfortunately, this is not as easy: All objects on all systems think for themselves, so the same tank will decide to fire on all systems at the same time. But we don't want to have multiple bullets created. So we use a trick: All objects that are created directly or indirectly by the user will be synchronized (created on the remote machine with the same global id), those that are created simultaneously on all machines are not synchronized. The drawback of this method is that some objects will have different ids on different machines and you must therefore be a bit careful about passing references to other objects with network code. Look in the code of the Global ID Manager class to see how this is done. Also take a look at that code if you want to know how to make reliable packet exchanges using udp. Most of this is accomplished in the following function from net_startup.cpp.

```

void process_data_packet(i4_file_class &r,i4_bool server)
{
    r.read_32();//skip empty field
    w32 ticksent=r.read_32();//to which tick does this packet belong?
    //here follows: reading out the objects
    gl_realtime_loader_class *rtloader=new
gl_realtime_loader_class(&r,i4_F,i4_F);

```


The format of the important network packet is as follows:

Offset	Size	Description
0	1	The constant G1_PK_GAME_DATA (already processed at this time)
1	4	A zero
5	4	The game tick (on the remote machine of course) when the packet was sent. Should be used for synchronization.
9	Varying	Object synchronization chunks, each one starts with
0	4	The global id of the object this chunk is for
4	2	The type of the object the data corresponds to. This is used if the given object doesn't exist locally and must be created.
6	Varying	The objects synchronization data
Varying	Varying	More data
Varying	4	A zero to indicate the end of the packet

```
w32 objfor=rtloader->read_32();
w16 typefor=0;
while (objfor!=0)
{
```

A special kind of ID is used to indicate when an object has been deleted remotely.

```
if (objfor==gl_global_id_manager_class::ID_DELETEPROPAGATE)
{
//don't delete if object just doesn't exist locally.
objfor=rtloader->read_32();
gl_object_class *obdel=gl_global_id.checked_get(objfor);
if (obdel)
{
obdel->set_flag(gl_object_class::THINKING,1); //Perhaps was not
thinking locally
obdel->stop_thinking();
obdel->unoccupy_location();
obdel->request_remove();
}
}
else
{
typefor=rtloader->read_16();
gl_object_class *obj=gl_global_id.checked_get(objfor);
if (!obj)
{
//Object seems to be new
//obj=gl_create_object(typefor);
if (typefor>=0 && typefor<=gl_last_object_type)
{
```

 It is not yet sure that there are no errors in the global id management code for synchronization.

We create the object if it didn't exist locally. The given global id should be free locally. The global id manager is responsible for this. The server assigns each client a range of global ids that he can use, this ensures that two different systems cannot generate two different objects with the same

global id. (The opposite is possible as stated above). The client stops all actions and pauses the game if the number of available ids for him drops below a certain limit. The correct values for this limit need to be checked experimentally.

```
obj=gl_object_type_array[typefor]-
>create_object(typefor,rtloader);
//the object will be in sync with the load
obj->occupy_location();
obj->grab_old();
```

```

        i4_warning("Remotelly creating object ID %i as %s.",obj-
>global_id,gl_object_type_array[typefor]->name());
        if (obj->get_flag(gl_object_class::THINKING))
        {
            //cannot request_think() without resetting this flag.
            obj->set_flag(gl_object_class::THINKING,0);
            obj->request_think();
        }

        gl_player_man.get(obj->player_num)->add_object(obj-
>global_id);
        if (typefor==gl_get_object_type("stank"))
        {
            // a new enemy supertank. Needs a few special updates
            gl_player_man.get(obj->player_num)->num_stank_lives()--;
            gl_player_man.get(obj->player_num)->calc_upgrade_level();
            gl_player_man.get(obj->player_num)-
>set_commander(gl_player_piece_class::cast(obj));
            gl_player_man.get(obj->player_num)->continue_wait=i4_F;
        }
        gl_network_time_man.updatecomplete(obj->global_id,ticksent);
    }
    else
    {
        i4_error("ERROR: Unknown object type requested.");
        delete rtloader;
        return;
    }
    //gl_global_id.assign(objfor,obj);
}
else
{

```

If the object already exists locally, we just load the data from the stream, which synchronizes the data. But first we check (experimental code!) whether we got an aged packet and we already got newer synchronization information on this instance earlier. We newer synchronize the local super tank to something received from a remote system. This code currently contains lots of debug output that can help find synchronization bugs.

```

        if (gl_network_time_man.shouldupdate(obj->global_id,ticksent))
        {
            obj->load(rtloader);
            gl_network_time_man.updatecomplete(obj->global_id,ticksent);
        }
        else
        {
            //i4_warning("Skipping update of object %i, data is
outdated.",obj->global_id);
            obj->skipload(rtloader);
        }
    }
}

```

If we are the server, we set the NEEDS_SYNC flag of this object again (to send the update to all the other clients), otherwise we reset it.

```

        if (server)
        {
            obj->set_flag(gl_object_class::NEEDS_SYNC,1);
        }
        else
        {
            obj->set_flag(gl_object_class::NEEDS_SYNC,0);
        }
        objfor=rtloader->read_32();
    }

    delete rtloader;
}

```


Trouble Shooting

Read on if something happens with your copy of Golgotha that you did not expect

5.1 Frequently asked questions and frequent problems

5.1.1 Problems compiling the game

Help! Golgotha does not compile.

That's a bit too vague to help you. Please state your question a bit more precise.

I get compiler errors saying that my platform is not implemented. (errors in arch.h)

If you are using anything but windows or unix, this is just normal. Add the definitions for your platform to arch.h in a way that avoids that anything changes for the already mentioned platforms (perhaps someone want to write an unix configure script?) If you are using windows, check that you project settings are correct (see my version of golgotha.dsp). One of the most important points is that the preprocessor symbols are set correctly.

I get some strange errors and/or internal compiler errors I've newer seen before when compiling the source using MSVC.

There seems to be a problem with your version of the compiler. Be sure to have the latest service pack for MSVC installed. It seems that golg won't compile successfully below MSVC6 SP4. It is highly recommended to install Service Pack 5 (SP5) for Visual Studio 6. Golgotha also requires the DirectX SDK Version 9 or above and – under some yet to be checked conditions – you need to update your Platform SDK files to at least the version from July 2000. Check the Microsoft Web Page for Platform SDK updates.

The compiler cannot find some files

If he complains about OS header files (i.e. windows.h), check that you correctly installed the platform SDK and your paths are set correctly. Be sure to also have the DirectX SDK (at least version 9) installed (compile another project and see whether it runs). If you get errors about golgotha include files, be sure that you add your main golgotha directory to the search path of the compiler.

I get linker errors

Be sure that all required components are bound, the required libraries can be found and you did not omit an implementation of some function if you wrote code on your own. The Golgotha engine requires quite a lot of libraries for successful linking, including the complete MFC package for windows. If you get linker errors for strange symbols (starting with many underscores for instance), something with your compiler files is wrong. You should reinstall the latest service pack or the entire compiler.

5.1.2 Problems launching the game

When I start the game, I get the following message: “There’s a problem with the DirectSound device (cannot lock buffer), try restarting your system.”

This is an internal problem of DirectSound that happens sometimes (not sure why and when). You can safely ignore this message, but you won’t have sound then. You’ll notice that most other games cannot play sound any more, too. To get back to normal, you must restart your system.

When I start the game, I get a message saying that “render.res” or “resource.res” or some entry within them is missing.

The two files need to be in the same directory as Golgotha.exe. Check that they are not corrupted and contain the requested info. This error message may also be displayed if the game is not started from the correct directory. You must make sure that the current directory is the main golgotha directory before launching the game. (When running under the debugger, check the working directory in the project settings)

When I start the game, I get an empty error message or something that just says “abort”, “retry”, “ignore”.

Check that you correctly compiled the windows resource file “golgotha.rc”. Depending on your language setting, you must manually edit the file such that the error dialog IDD_ERRORDIALOG is included. This problem seems to be common under Windows 2000/XP for all Golgotha versions bellow 1.0.5.5.

5.1.3 Problems running the game

The Error Log is full of messages saying that some file(s) is (are) missing.

Try to find out what this file is for and place a matching file there. You can ignore messages that say “Golgotha.cd missing” or “some sfx missing” (if the sound files are not installed).

If I press key X, the game reacts as if Alt+X was pressed (or similar problems)

Actually, I’ve never seen this bug unless I was debugging, but who knows? To solve this problem, just press the modifier keys alone one after the other (shift, alt, ctrl).

The game is running very slowly. (the profile tells me that “decompress_jpg” takes much time).

This has to do with the current implementation of the texture loader and will be optimized soon. To decrease load times, set the texture quality to minimum.

I cannot win the game!

I know; beside the fact that the test level is very difficult, there’s currently no method implemented that checks some winning condition.

Distant terrain looks very screwed up

This might have several reasons. The most probable is that you are using a video mode that Golgotha doesn’t understand properly. Try to change the screen bitdepth or set the texture resolution to “Automatic”. If only the colors of dis-

tant terrain is wrong (i.e. Water becomes red, grass becomes blue...) contact me, and I'll fix that (I know what's wrong then, but I cannot test it, since I do not know which hardware supports which texture formats).

Appendixes

This part contains some additional information that could not reasonably be placed elsewhere

6.1 References

- [1] <http://www.jonathanc Clark.com> The website where golgotha originally was released.

6.2 Index

A

ANSI 24
arch.h 24
army 22

B

browse 26

C

CD-ROM
 Installation 9
compiler 49

D

device
 time 26
DirectSound 50
DirectX 5 8

F

FAQ 49

G

Golgotha
 Name 5
 origin 5
 purpose 5
 starting 17
golgotha.rc 50

I

i4
 core 24
i4_array<T> 26
i4_const_str 27
i4_dynamic_que<T> 27
i4_fixed_que<T> 27
i4_init_class 24
i4_os_string 27
i4_str 27
input
 class 26
 keyboard 26
 mouse 26

K

Keys
 Enter 21
 Tab 22

L

Level
 Editor 20
Load 18

M

Main Menu 19
Maxtool
 Editor 20
Microsoft Visual C++ 8

N

Network 18

New Game 18

O

OpenGL 8
Options 18

P

platform 24

Q

questions 49
Quit 18

R

resource.res 50

S

Save 18
Screen Shoot 19
Setup 9
STL 24
String 27
supergun 21
supertank 21

X

X server 8