RenderWare Graphics

White Paper

Building RenderWare Graphics

Contact Us

Criterion Software Ltd.

For general information about RenderWare Graphics e-mail info@csl.com.

Contributors

RenderWare Graphics development and documentation teams.

Table of Contents

1. Introduction 4
2. Overview 5
3. The options.mak File 6
   - RWTARGET=<target> 7
   - RWOS=<os> 8
   - RWCOMPILER=<compiler> 8
   - RWGSDK=<path> 9
   - RWDEBUG=<rwdebug flag> 9
   - RWMETRICS=<rwmetrics flag> 10
   - PLUGINS=<plugin list> 10
   - TOOLKITS=<toolkit list> 10
   - CDEBUG=<c debug flag> 10
   - COPTIMIZE=<c optimize flag> 11
   - RWLOGO=<logo flag> 11
   - RWSPLASH=<splash flag> 11
   - RWMOUSE=<mouse flag> 11
   - RWDLL=<dll generation flag> 12
   - DXSDK=<path> 12
   - OGLLIBPATH=<path> 12
   - PS2_DRIVE=<driver letter> 12
   - IOPPATH=<path> 13
   - XBOXSDK=<path> 13
4. Makefile Hierarchy 14
5. Tools needed to Build RenderWare Graphics 19
   - Compiler, Linker and Assembler 19
   - Platform Specific Libraries & headers 19
   - Building the Playstation 2 SDK on Linux 19
6. GUI tool for Building RenderWare Graphics 20
7. Building components of the RenderWare Graphics SDK 21
   - Use of .mcp files 22
8. Building a complete SDK 23
9. Common Build Problems 24
   - General build problems 24
   - cygwin version conflicts 24
   - Platform specific SDK versions and updates 24
   - Unable to find platform SDK's 24
   - PC SDK build is looking for the CodeWarrior compiler – however this isn't being used 25
10. Advanced features 26
   - Rebuilding a single plugin 26
   - .ipp files 26

1. Introduction

This document describes the process used to build a RenderWare Graphics SDK from the source tree.

In addition, some hints and tips are provided for the reader on how to perform some of the more advanced construction techniques used by the development teams at CSL.

The process is described from the inside out – starting with a description of the tools and processes used to build a single set of RenderWare Graphics libraries – then expanding this into the higher level process used to build an entire SDK – including multiple library sets, precompiled examples and documentation. A brief outline of the use of a GUI Windows tool to manage the RenderWare Graphics build process is also provided.

2. Overview

Since RenderWare Graphics is a multi-platform solution, we have adopted a process and tool-chain for building the SDK that allows the same system to be utilized on all platforms.

Where practical, we make no assumption about the platform used to build the SDK. At CSL we use a PC hosted system to build and maintain the SDK – however internal and external developers are also using Linux hosted development environments (for PlayStation 2 at least). See *5 Building the Playstation 2 SDK on Linux* for more information on using Linux as the build environment for Playstation 2.

All programming projects require a method for compiling and linking the various individual files that make up a library or executable. Modern compilers are typically shipped with an IDE (Integrated Development Environment) that serves this purpose. (Examples of an IDE are Microsoft's Visual Studio or Metrowerks CodeWarrior.) However most IDE's are tightly coupled to the underlying compiler and it is difficult (or impossible in some cases) to use a competitor's product within this framework.

We have elected to use the freely available gnumake command line utility as the heart of the RenderWare Graphics build process; for details, see

http://www.fsf.org/software/make/make.html

In addition, we also utilize a number of Unix command line functions (cp, mv, rm etc.) in the process. All of the tools needed to build RenderWare Graphics are either part of the host operating system or ship with the SDK (in the `rwsdk/bin` folder).

✎ On the PC, it is important that the `rwsdk/bin` folder is on your PATH for the build to be successful.

In theory, the use of these build tools allow us to build RenderWare Graphics using any compiler and on any platform.

In keeping with the modular nature of RenderWare Graphics, the build process is also modular and is constructed in a hierarchical fashion. Each component has it's own makefile which, in turn, is called by higher level makefiles as more and more components are included in the build process. The makefile hierarchy is described in more detail later.

Without wanting to point out the obvious, if you have a previous version of RenderWare Graphics and have altered any of the RenderWare Graphics folders contents and wish to keep the changed files, it may be a good idea to move or rename them before proceeding with any of the commands in this document, as the build process may overwrite these files.

3. The options.mak File

There are a number of ways in which the whole RenderWare Graphics SDK or specific components thereof can be built. The most common form is where all of the user configurable build options for RenderWare Graphics are contained within a single control file. This control file can be

- either specified on the command line to the build process:

```
gnumake RWOPTIONS=c:/rw/graphics/rwsdk/options.mak
```

- or, alternatively, be explicitly specified in the environment:

```
set RWOPTIONS=c:/rw/graphics/rwsdk/options.mak

gnumake
```

In either case, the fully qualified pathname of the control file must be specified, using forward slashes to separate paths.

The selection of which method to use depends entirely on your own development practices. You can choose to:

- either maintain different control files for different builds and specify them on the command line.

```
gnumake RWOPTIONS=c:/rw/graphics/rwsdk/options.d3d8

gnumake RWOPTIONS=c:/rw/graphics/rwsdk/options.sky

etc.
```

- or simply use a single control file and edit as necessary.

The name of the file is not significant. At CSL we typically use `options.mak` for generic versions and options.d3d8 etc. for specific ones.

A Windows GUI tool is also available to automate the generation and maintenance of options.mak files. This is described further in GUI tool for Building RenderWare Graphics.

A sample `options.mak` file is shown below. This example is for the release D3D8 libraries

```
RWTARGET=d3d8

RWOS=win

RWGSDK=c:/rw/graphics/rwsdk

```

```
PLUGINS=adc anisot collis crowd dmorph dpvs hanim lodatm logo
ltmap matfx mipkl morph patch prtstd prtadv ptank pvs random
skin2 spline team toon userdata

TOOLKITS=2d 2danim anim bezpatch bmp brycntrc charse cmpkey
gencpipe geomcond import intsec ltmap mipmapk pick pitexd png
quat ras ray sknsplit slerp splpvs tiff tilerd toc vcat wing
world

CDEBUG=0

RWCOMPILER=visualc

RWLOGO=1

RWSPLASH=0

RWMOUSE=0

DXSDK=c:/mssdk

OGLLIBPATH=c:/ogllib

RWDEBUG=0

RWDLL=0
```

Each of these options is now described:

RWTARGET=<target>

This defines the target driver for which of RenderWare Graphics is to be built. This option selects platform specific driver files and compiler options.

Valid values for <*target*> are:

- sky2 – for PlayStation 2 native libraries
- xbox – for Xbox native libraries
- opengl – for OpenGL libraries on many platforms (use RWOS to select variants)
- d3d8 – for Direct3D8 libraries on the PC
- gcn – for GameCube native libraries
- null – for the NULL driver on the PC
- nullsky – for the NULL sky driver on the PC

- nullxbox – for the NULL Xbox driver on the PC
- nullgcn – for the NULL GameCube driver on the PC

A complete list of all possible options can be derived from the contents of the `rwsdk/makeincl/rwtarget` folder in the source tree.

RWOS=<os>

This defines the target operating system. It is used to specify OS specific files in the SDK, and is also a parameter in the compiler selection process.

Valid values for <*os*> are:

- win – for PC builds (d3d8, opengl, null, etc)
- xbox – for Xbox builds
- sky – for PlayStation 2 builds
- gcn – for GameCube builds

A complete list of all possible options can be derived from the contents of the `rwsdk/makeincl/rwos` folder in the source tree.

RWCOMPILER=<compiler>

This defines the compiler used to build the SDK.

- Note: there can be more than one supported compiler for any given platform.

Valid values for <*compiler*> are:

for RWOS=win

- cwpc – CodeWarrior for Windows
- intel – Intel 4 compiler
- visualc – Microsoft Visual C 6 compiler
- net – Microsoft Visual Studio .NET compiler

for RWOS=sky

- cwsky – CodeWarrior for PlayStation 2

- skygcc – ee-gcc compiler on PC or Linux (V1.6b & V2.96)
- sky295 – gcc v2.95 on PC
- sky2953 – gcc v2.953 on PC
- skyprodg – SN ps2cc and ps2ld

for RWOS=xbox

- xbox – VisualC 7 for Xbox
- net – Microsoft Visual Studio .NET compiler

for RWOS=gcn

- cwdfin – CodeWarrior for GameCube

A complete list of all possible options can be derived from the contents of the `rwsdk/makeincl/rwos/<os>/rwcmplr` folder in the source tree.

RWGSDK=<path>

Specifies the location of the RenderWare Graphics `rwsdk` directory.

✎ This option is not needed in order to build the SDK but is required to build the examples and other sample code.

The RenderWare Graphics installer will create environment variables `RWGDIR`, containing the installed location of RenderWare Graphics (e.g. `c:/rw/graphics`), and `RWGSDK`, containing `<RWGDIR>/rwsdk`.

If the `RWGSDK` environment variable does not exist for any reason, and it is not explicitly specified in the options.mak file, then its path is constructed (local to a gnumake build) from `<RWGDIR>/rwsdk`.

RWDEBUG=<rwdebug flag>

Specifies whether or not to build a debug set of RenderWare Graphics libraries. The RenderWare Graphics debug libraries provide additional parameter validation and other debug checks such as assertions to assist in the elimination of coding errors.

Valid options for `<RWDEBUG` *flag*> are:

- 1 – build a debug set of libraries

- 0 – build a release set of libraries

✎ This option does *not* enable C debugging information in the libraries. To generate C debugging information in order to step through the code with a debugger, you will need to specify the CDEBUG option (see below).

RWMETRICS=<rwmetrics flag>

Specifies whether or not to build a metrics set of RenderWare Graphics libraries. The RenderWare Graphics metrics libraries provide additional performance reporting metrics to the application. Metrics can be used to assist in the location and hence elimination of performance bottlenecks in the application.

Valid options for `<RWMETRICS` *flag*> are:

- 1 – build a metrics set of libraries
- 0 – build a release set of libraries

✎ `RWMETRICS` is only really useful with release builds of the library. It should be possible to build an `RWDEBUG+RWMETRICS` library but this will be of limited practical use.

PLUGINS=<plugin list>

Specifies the plugins to be built in the SDK. This is a list of the plugin directory names (from `rwsdk/plugin`) separated by spaces.

✎ If this list is not specified then all plugins in the `rwsdk/plugin` directory will be built.

TOOLKITS=<toolkit list>

Specify the toolkits that are included in the SDK. This is a list of the toolkit directory names (from `rwsdk/tool`) separated by spaces.

✎ If this list is empty then by default, all tools in the `rwsdk/tool` directory will be built.

CDEBUG=<c debug flag>

Specifies the generation of C debugging information in the build. Valid options for *<c debug flag>* are:

- 1 – enable C debugging

- 0 – disable C debugging

COPTIMIZE=<c optimize flag>

Specifies compiler optimization. Valid options for *<c optimize flag>* are:

- 1 – enable Compiler Optimizations
- 0 – disable Compiler Optimizations

✎ If unspecified, COPTIMIZE will default to the opposite of CDEBUG.

RWLOGO=<logo flag>

Specifies whether or not the RenderWare Platform logo is to be displayed on the screen when running the SDK examples and viewers. This is used in tandem with the logo plugin to provide a quick and easy way of incorporating the logo into any example or demonstration application.

Valid options for *<logo flag>* are:

- 1 – Include the RenderWare Platform logo in applications
- 0 – Don't include the RenderWare Platform logo in applications

RWSPLASH=<splash flag>

Used when building the examples to control whether or not the RenderWare Graphics splash screen is displayed on the screen at startup.

Valid options for *<splash flag>* are:

- 1 – Include the RenderWare Graphics splash screen in examples
- 0 – Don't include the RenderWare Graphics splash screen in examples

✎ This option is currently disabled by default as the splash screen artwork is out of date with the latest logo.

RWMOUSE=<mouse flag>

For the example code that has been written to require a mouse pointer, we provide a simple emulation for platforms lacking native mouse support. This option specifies whether or not this mouse emulation is needed for the platform.

Valid options for *<mouse flag>* are:

- 1 – platform doesn't have a mouse – emulate where necessary
- 0 – platform supports a mouse natively – emulation not required

RWDLL=<dll generation flag>

The default RenderWare Graphics build process generates a set of libraries that applications can then link against. In some cases, for example the RenderWare Graphics exporters, a DLL containing the RenderWare Graphics functionality is of more use. Therefore, for *NULL* targets only, the additional make option RWDLL is available to generate the usual RenderWare Graphics libraries but also a DLL containing most of those libraries. This DLL is called `rwg.dll` and is placed in the `rwsdk/dll/null/debug|release|metrics` folder.

Note that a full clean of the NULL libraries is recommended before running a DLL build as the RenderWare Graphics source is built using multithreaded DLL compiler setting.

Valid options for *<dll generation flag>* are:

- 1 – `rwg.dll` is generated after the regular libraries are created.
- 0 – only the regular libraries are created.

DXSDK=<path>

For Direct3D 8 PC builds, this specifies the directory where the Microsoft DirectX SDK can be found. This will typically be:

```
DXSDK=c:/mssdk
```

OGLLIBPATH=<path>

For OpenGL PC builds, this option specifies the directory in which the opengl32.lib file may be found.

PS2_DRIVE=<driver letter>

For PlayStation 2 builds, if defined this will be prepended, with a colon appended, to references to Sony libraries and toolchain files in \usr. (Linux hosted builds should not define this). This might be:

```
PS2_DRIVE=c
```

IOPPATH=<path>

For PlayStation 2 builds, this will be prepended in examples built using the skeleton and the host filesystem, to references to modules/*.{img,irx} This will typically be:

```
  IOPPATH=/usr/local/sce/iop
```

XBOXSDK=<path>

For Xbox builds, this specifies the path to the Microsoft Xbox SDK.

The path for this option must be specified in short name form. i.e.

```
XBOXSDK=c:/progra~1/micros~4/
```

4. Makefile Hierarchy

The makefiles used in the build process are described in the following tree diagram. Please refer to each individual makefile for more specific details on how they operate. Note: a familiarity with gnumake is assumed in order to understand these makefiles.

```
|   makefile                     Top level build process makefile. This
                                 makefile controls the components
                                 included in the SDK. The plugins,
                                 tools, demos, examples and viewers
                                 that make up the SDK are all defined
                                 here.  Typically this makefile is used
                                 in conjunction with one of the master
                                 control files below to create a
                                 complete SDK tree.  It operates by
                                 creating an options.xxx file for the
                                 build and then calling the lower level
                                 makefiles for each of the components
                                 in turn with this set of options.

|   makefile.gcn                 Master makefile for GameCube SDK
|
|   makefile.sky                 Master makefile for PlayStation 2 SDK
|
|   makefile.win                 Master makefile for PC SDK
|
|   makefile.xbox                Master makefile for Xbox SDK

+---examples                     Each SDK example has its own makefile
                                 which defines how the example should
                                 be built. Each of the examples can be
                                 built in isolation.
|   |
|   \---oneoftheexamples
|       |   makefile
```

```
+---rwsdk
|   |   makefile                 This is the master makefile used to
                                 build the SDK. For developers who are
                                 regularly rebuilding a single version
                                 of RenderWare Graphics, this will be
                                 the makefile that is usually used.
                                 This calls sub-directory makefiles to
                                 build the core, plugins and tools.

|   +---makeincl                 This directory contains all files
                                 included from rwsdk makefiles.

|   |   |   makeaplg             This file is included by all the
                                 archive plugin makefiles. It contains
                                 a set of options that are common to
                                 all archive plugins.

|   |   |   makecore             This file is included by the core
                                 makefile.

|   |   |   makedll              This file contains options to build a
                                 combined DLL containing most of the
                                 RenderWare Graphics libraries. This is
                                 only possible for NULL targets
                                 currently.

|   |   |   makeopt              Common makefile options shared by all
                                 SDK components.

|   |   |   makeplug             This file is included by all the
                                 plugin makefiles. It contains a set of
                                 options that are common to all
                                 plugins.

|   |   |   maketool             This file is included by all the
                                 toolkit makefiles. It contains a set
                                 of options that are common to all
                                 toolkits.

|   |   |   makewrld             This file is included by the world
                                 plugin makefile.

|   |   +---rwos                 This directory contains O/S specific
                                 build options.
|   |   |   +---gcn
|   |   |   |   +---rwcmplr      This directory contains compiler
                                 specific build options.
|   |   |   |   |   +---cwdfin
|   |   |   +---sky
|   |   |   |   +---rwcmplr      This directory contains compiler
                                 specific build options.
|   |   |   |   |   +---cwsky
|   |   |   |   |   +---skygcc
|   |   |   +---win
|   |   |   |   +---rwcmplr     This directory contains compiler
                                specific build options.
|   |   |   |   |   +---intel
|   |   |   |   |   +---visualc
|   |   |   +---xbox            This directory contains compiler
                                specific build options.
|   |   |   |   +---rwcmplr
|   |   |   |   |   +---xbox
|   |   +---rwtarget            This directory contains target
                                specific build options. Files herein
                                typically define the additional driver
                                files needed in the core library over
                                the shared set.
|   |   |   +---d3d8
|   |   |   +---default         Options common to all targets.
|   |   |   +---gcn
|   |   |   +---null
|   |   |   +---nullgcn
|   |   |   +---nullsky
|   |   |   +---nullxbox
|   |   |   +---opengl
|   |   |   +---pipe            A variation on rwtarget for pipeline
                                specific files.
|   |   |   |   +---p2
|   |   |   +---sky2
|   |   |   +---xbox

|   +---plugin                  The plugin directory consists of a
                                number of subdirectories - one per
                                plugin. Each plugin has its own
                                makefile that can be used to build the
                                plugin in isolation.
|   |   \---oneoftheplugins
|   |       |   makefile        The makefile for a plugin.


|   +---src                     The RenderWare Graphics core files.
|       |   makefile            Makefile for RenderWare Graphics core
                                files.
|
|   +---tool                    The toolkit directory consists of a
                                number of subdirectories - one per
                                toolkit. Each toolkit has its own
                                makefile that can be used to build the
                                toolkit in isolation.
|   |   |
|   |   \---oneofthetoolkits
|   |       |   makefile        The makefile for a toolkit.

|   \---world                   The world plugin is a special case,
                                and has a separate makefile.
|       |   makefile            Small wrapper makefile that
                                encapsulates makewrld.
```

```
|
+---shared                      The shared tree contains files shared
                                between the demos, examples, tools and
                                viewers.

|   |   makeopt                 Common makefile options for
                                application tree. This file is
                                included by all application makefiles.

|   |   maketarg                defines the build options for the
                                                              target
                                d3d8, opengl, sky2 etc

|   +---cwcommon                Shared CodeWarrior files.
|   +---democom                 Shared files for demonstrations.
|   +---makeincl                Shared files included from application
                                makefiles.
|   |   +---rwos                Shared files per O/S.
|   |   |   +---gcn
|   |   |   |   +---rwcmplr     Shared files per compiler.
|   |   |   |   |   +---cwdfin
|   |   |   +---sky
|   |   |   |   +---rwcmplr     Shared files per compiler.
|   |   |   |   |   +---cwsky
|   |   |   |   |   +---skygcc
|   |   |   |   |   +---
skyprodg
|   |   |   +---win
|   |   |   |   +---rwcmplr     Shared files per compiler.
|   |   |   |   |   +---intel
|   |   |   |   |   +---visualc
|   |   |   \---xbox
|   |   |       +---rwcmplr     Shared files per compiler.
|   |   |           \---xbox
|   +---skel                    Source files of skeleton framework for
                                demos, examples, tools and viewers.
|   |   +---sky
|   |   |   +---mpeg
|   |   +---win
|   |   \---xbox
|   |
|   \---sncommon                Shared SN-Systems files.

+---tools                       The SDK tools directory contains PC
                                hosted tools. These are built with one
                                of the PC SDK libraries.
|   |
|   \---oneofthetools
|       |   makefile            Each tool has it's own makefile.
|   |   +---gcn


\---viewers                     Each SDK viewer has its own makefile
                                which defines how the viewer should be
                                built. Each of the viewers can be
```

```
                                    built in isolation
    |
    \---oneoftheviewers
        |   makefile
```

5. Tools needed to Build RenderWare Graphics

RenderWare Graphics ships with many of the build tools required – these can be found in the `rwsdk/bin` directory.

For the purposes of building RenderWare Graphics, it is assumed that the following additional tools are present on the build machine.

Compiler, Linker and Assembler

These should be configured to support command line build options. For a list of supported compilers see the '`makeincl/rwos/<os>/rwcmplr`' directory in the tree structure above. Typically, your `PATH`, `INCLUDE` and `LIB` environment variables should be set appropriately for these tools.

Platform Specific Libraries & headers

These are the SDK components shipped by the platform manufacturer. It is assumed that these are installed using their default configuration.

Building the Playstation 2 SDK on Linux

Since the Windows installer that supplies the RenderWare Graphics distribution does not preserve Unix permissions, several of the tools need to have their permissions restored before the SDK is built.

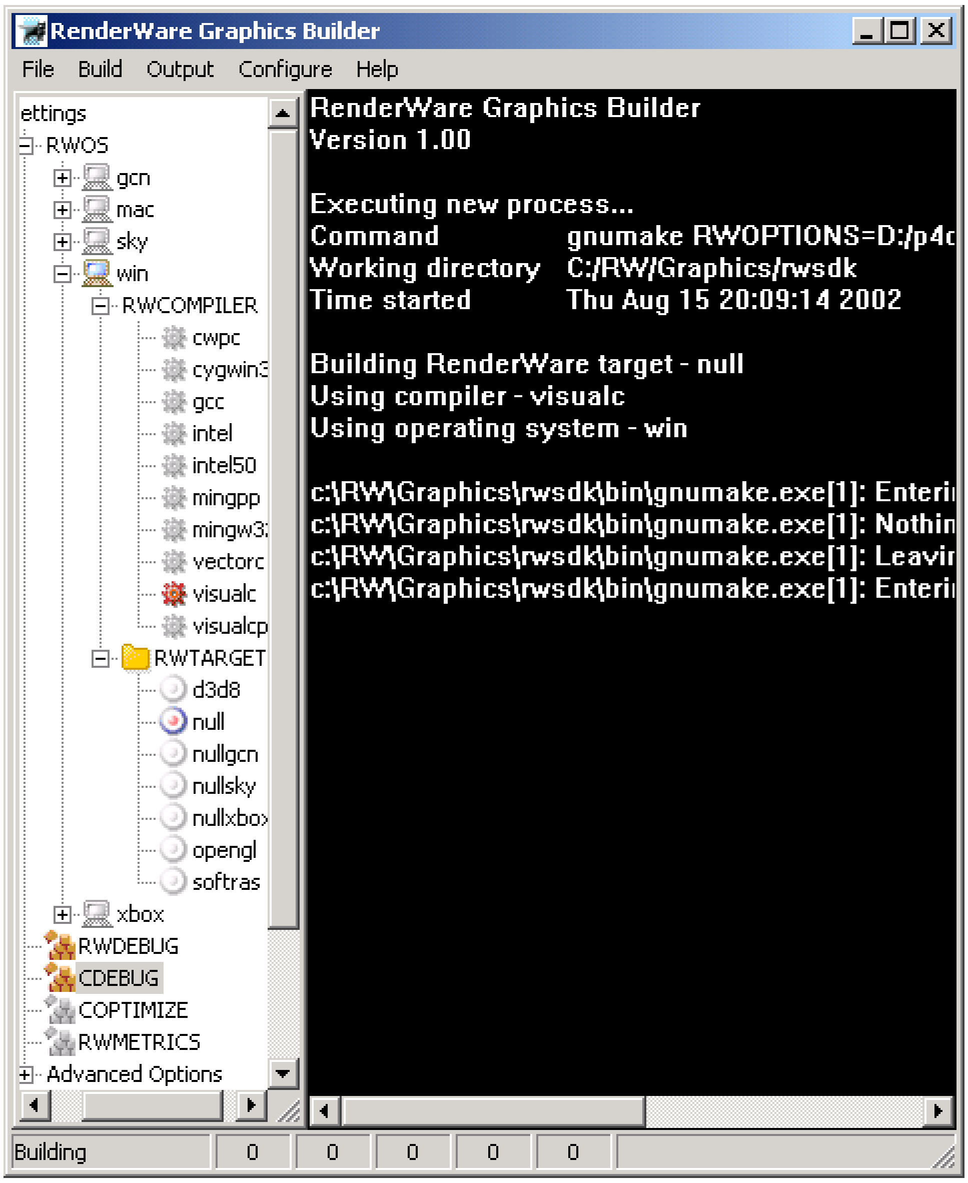
The following commands, issued in the rwsdk directory, will restore the permissions of the tools:

```
chmod a+x bin/sed

chmod a+x bin/mkdir

chmod a+x buildtools/findsyms/findsyms

chmod a+x buildtools/incgen/incgen

chmod a+x buildtools/inline/inline
```

6. GUI tool for Building RenderWare Graphics

A Windows (compatible with Windows 2000 and above) GUI tool is supplied with the RenderWare Graphics source distribution. This is a graphical front-end to managing options.mak files and building the RenderWare Graphics SDK.

This tool allows you to build the SDK, documentation, individual plugins or toolkits, and examples and viewers using the same build options at a touch of a button. The source for examples and viewers are supplied with the RenderWare Graphics binary distribution.



```
RenderWare Graphics Builder
Version 1.00

Executing new process...
Command              gnumake RWOPTIONS=D:/p4c
Working directory    C:/RW/Graphics/rwsdk
Time started         Thu Aug 15 20:09:14 2002

Building RenderWare target - null
Using compiler - visualc
Using operating system - win

c:\RW\Graphics\rwsdk\bin\gnumake.exe[1]: Enteri
c:\RW\Graphics\rwsdk\bin\gnumake.exe[1]: Nothin
c:\RW\Graphics\rwsdk\bin\gnumake.exe[1]: Leavir
c:\RW\Graphics\rwsdk\bin\gnumake.exe[1]: Enteri
```

For more information, you are referred to the build tool user guide, `buildtool.pdf`, in `docs/tools` in the RenderWare Graphics directory.

7. Building components of the RenderWare Graphics SDK

There are alternative methods for building parts of, or the whole of, the RenderWare Graphics SDK.

✎ Tip: a quick method to generate just the SDK libraries and headers (whilst avoiding building any examples, tools or viewers) is to invoke the command line

```
gnumake RWOS=win RWCOMPILER=visualc RWTARGET=d3d8 sdk-libs
```

Take a look at makefile.xbox, makefile.win, makefile.gcn and makefile.sky (PS2) for extra information and examples.

NOTE: there are dependencies between SDK components, for example, the SDK examples require the latest version of the SDK libraries to build successfully. It is recommended that the user gains extended experience of building RenderWare Graphics in this fashion as build dependency problems may occur. However, any problems should be resolved if the user performs the process explained in The options.mak File section, or the shortcut explained below.

Typically, developers are only working on one "flavor" of the libraries at any time. These instructions describe how to build a single library set.

1. Construct an `options.mak` file that contains the options for the platform required (see above).

✎ Tip: a quick method to generate an `options.mak` file that contains the standard options is to use the top-level makefile as follows

```
gnumake RWOS=win RWCOMPILER=visualc RWTARGET=d3d8 options.d3d8
```

Substitute any of the supported values for the platform in place of win, visualc and d3d8 above. The resultant file can now be renamed as `options.mak` and used in this process.

2. Specify the path to this `options.mak` file in the environment using `RWOPTIONS=c:/mypath/options.mak`.
3. Ensure that `rwsdk/bin` tools directory is included in your `PATH` environment variable.
4. Reboot to ensure the environment variables are picked up.
5. cd `/rwdistribution/rwsdk`.

6. Type '`gnumake`'.

The `options.mak` file can be manually edited to build variations on a library or to build a separate library.

Use of .mcp files

CodeWarrior project (MCP) files are provided with the RenderWare Graphics demos, examples and viewers. This allows users of the CodeWarrior IDE to build RenderWare Graphics applications from within their favored development environment, and illustrates a suitable structure for generating new RenderWare Graphics applications using CodeWarrior. Targets are provided for the standard debug, release and metrics configurations in a similar manner to the other build methods available.

It is recommended that users have the most recent version of the MetroWerks CodeWarrior IDE for their development platforms.

8. Building a complete SDK

If you wish to reproduce the entire SDK build process, the simplest method is to use one of the top-level master build control files.

e.g.

```
gnumake -i -f makefile.sky
```

will build the PlayStation 2 SDK. The `-i` option will ignore any errors that occur, allowing the build process to run to completion.

Examination of these master build files will provide more details on what is taking place.

9. Common Build Problems

General build problems

General build problems can be avoided by

- Cleaning out the folders before rebuilding (removing unused or old source and libs)
- Updating your paths so that all the newest platform specific SDKs are found
- Making sure that the most up-to-date tools are included in the paths

cygwin version conflicts

The build process uses the freely available `cygwin` suite of command line utilities. If these are already installed on the build machine then it is possible that there will be a version number conflict between the installed version and that shipped with the SDK.

These conflicts can usually be resolved by renaming or deleting the file `rwsdk/bin/cygwin1.dll` from the distribution.

Platform specific SDK versions and updates

The versions of the software development kits provided for the development hardware should be as up-to-date as possible to avoid compatibility problems when building with RenderWare Graphics. The current versions of these platform specific SDKs, on which the RenderWare Graphics development tools have been built, may be found in the `changelog.txt` file in the RenderWare Graphics root directory.

Unable to find platform SDK's

If the platform specific SDK components have not been installed in the default location, then the build process will be unable to find them. This can be resolved by:

- putting the correct path in your `options.mak` file
- if building the entire SDK, editing the top level makefile to indicate the correct location

PC SDK build is looking for the CodeWarrior compiler – however this isn't being used

In order to provide a set of libraries that work with all PC compilers, the RW SDK is built with the Microsoft Visual C/C++ compiler. This generates libraries compatible with all compilers - with one exception. The PNG toolkit, which uses the publicly available code, needs to be rebuilt with CodeWarrior in order to provide a library that is compatible the CodeWarrior compiler. If CodeWarrior support is not required, then any such errors can be safely ignored.

10. Advanced features

This section describes some of the more advance techniques that are used by the development team at CSL. These are provided for information only and may not be supported under all configurations.

Rebuilding a single plugin

As mentioned briefly above, it is possible to rebuild a single plugin or tool in isolation from the main SDK build. In order to achieve this you need to do the following:

1. Build the flavor of SDK currently being used for development.
2. cd `rwsdk/plugin/anyplugin`.
3. Type `gnumake`.

✎ This will require that the `options.mak` file is specified using the `RWOPTIONS` environment variable.

.ipp files

There is an additional rule that is present in the SDK makefiles that supports the generation of files with the `.ipp` extension. These files contain the result of the pre-processor stage of the compiler.

It can sometimes be useful to inspect a preprocessed/expanded version of a file to allow debugging of code sections consisting of macros.

A simple example of generating a `.ipp` file is shown below:

```
cd rwsdk/plugin/spline
gnumake rpspline.ipp
```

The file `rpspline.ipp` contains the preprocessed version of `rpspline.c`