

RenderWare Graphics

Tool

SDK Build Tool

Copyright © 2003 – Criterion Software Ltd.

Contact Us

Criterion Software Ltd.

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

Developer Relations

For information regarding Support please email devrels@csl.com.

Sales

For sales information contact: rw-sales@csl.com

Contributors

RenderWare Graphics development and documentation teams.

The information in this document is subject to change without notice and does not represent a commitment on the part of Criterion Software Ltd. The software described in this document is furnished under a license agreement or a non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or non-disclosure agreement. No part of this manual may be reproduced or transmitted in any form or by any means for any purpose without the express written permission of Criterion Software Ltd.

Copyright © 1993 - 2003 Criterion Software Ltd. All rights reserved.

Canon and RenderWare are registered trademarks of Canon Inc. Nintendo is a registered trademark and NINTENDO GAMECUBE a trademark of Nintendo Co., Ltd. Microsoft is a registered trademark and Xbox is a trademark of Microsoft Corporation. PlayStation is a registered trademark of Sony Computer Entertainment Inc. All other trademark mentioned herein are the property of their respective companies.

Table of Contents

1.1	Introduction	4
1.1.1	What you need to use RWB	4
1.1.2	Compiler environment sanity checks	4
1.1.3	Advantages of using RWB	5
1.1.4	Glossary of terms	5
1.1.5	Further reading	6
1.2	Getting started	7
1.2.1	Introducing the RWB window	7
1.2.2	Configuring a source tree	8
1.2.3	Browsing the tree-view	9
1.2.4	Running a basic SDK build	13
1.3	Menus in detail	17
1.3.1	File menu	17
1.3.2	Build menu	18
1.3.3	Output menu	19
1.3.4	Configure menu	20
1.3.5	Help menu.....	21
1.4	Tree lists in detail	22
1.4.1	Settings tree.....	22
1.4.2	SDK components tree	23
1.4.3	Applications Tree.....	28
1.5	Keyboard shortcuts.....	30
1.5.1	Fixed shortcuts	30
1.5.2	Customizable shortcuts	30
1.6	Resources file.....	31
1.6.1	Root paths.....	31
1.6.2	RenderWare Internal Paths	31
1.6.3	RenderWare Extension Paths	31
1.6.4	RenderWare Application Paths.....	32
1.6.5	Target bindings	33
1.6.6	Advanced options	33
1.6.7	Operating System Build Paths	34
1.6.8	Compiler context strings.....	34
1.6.9	Recent MAK files	35
1.6.10	Keyboard shortcuts.....	35
1.6.11	Compiler setup.....	36
1.6.12	Output options	37

1.1 Introduction

The RenderWare Builder (RWB) is a graphical user interface front-end to the **gnumake** process used to build the RenderWare Graphics SDK. Its limitations are therefore the same as **gnumake**. It is intended as a learning device for those new to building RenderWare Graphics and also as an aid to those experienced in that art.

1.1.1 What you need to use RWB

You will need the following in order to use RWB on your computer:

- Windows 2000 and above.
- A RenderWare Graphics 3 (release 3.2 and above) source tree.
- The RWB executable, **rwbuilder.exe**. Using the RenderWare Graphics installer to install to the default location, this executable will be found in `c:/rw/graphics/tools/rwbuilder`.
- The RWB resource file, **rwbuildres.cfg**, in the same folder as **rwbuilder.exe**. This file must have write-access.
- Compilers set up correctly to execute from the command line, including the location of necessary libraries and headers. If RWB reports any problems about not being able to create a "process" then please check that you can use the selected compiler from the command line. If not, refer to your compiler's documentation as to how to set it up for command line use.

gnumake is provided with the RenderWare Graphics source, and RWB sets the PATH environment variable appropriately. Note that this change to PATH is local to RWB.

RWB is, essentially, a graphical wrapper on top of **gnumake**. Hence, if compilers are set up to build from the command line, RWB should work as well.

1.1.2 Compiler environment sanity checks

It is a requirement of using RWB that the compilers used are set up so that they can be executed, and can find their associated headers and libraries, from the command line. This is because RWB is just a wrapper on top of a command line build process.

However, to aid those that are unfamiliar with setting up compilers to function from the command line, RWB provides sanity checks for the environment. These include:

1. Checks that a compiler's entire tool chain (compiler, linker, archiver, etc.) is visible from the command line

2. Checks that a compiler's required header and library path environment variables are present.

It is more difficult to ascertain whether header and library paths set up in the environment variables above are correct since it is entirely possible to install compilers to non-default locations. Hence only the existence of the environment variables are tested. Some environment variables are declared as mandatory that must exist for a build to run, and some are optional which will allow a build to run with the possibility of build issues occurring.

These sanity checks are not guaranteed to be full proof.

1.1.3 Advantages of using RWB

The main advantages of using RWB over command line based builds are:

- Easier learning curve to the setup and maintenance of building the RenderWare Graphics SDK.
- Point-and-click interface.
- Summary of warnings/errors encountered during a build, and context highlighting to easily locate them in the build output.
- Build output can be saved for inspection or future reference, or sent directly to a text editor.
- Maintainability of RenderWare Graphics `options.mak` files.
- Sanity checks of the command line build environment.

1.1.4 Glossary of terms

Here is a glossary of terms that you will encounter in this user guide.

- RWB

An acronym for RenderWare Graphics Builder.

- `options.mak`

This file is used by the RenderWare Graphics build process to describe all of the build options, SDK locations, the plugins/toolkits to build. RWB automatically generates the `options.mak` files according to the settings you select.

- `rwbuildres.cfg`

This file is used internally by RWB to configure itself to your preferences, e.g. the default RenderWare Graphics source path, the locations of external SDKs, etc. See *1.6 Resources file* for more information on the contents of this file.

- RWOS

This is the platform for which the RenderWare Graphics SDK is to be built for. For example, *win*, *gcn*, *xbox*.

- RWCOMPILER

This is the compiler used for the selected RWOS to build the RenderWare Graphics SDK. For example, *visualc* on *win*, *intel* on *win*, *skygcc* on *sky*.

- RWTARGET

This is the platform-variation for the selected RWOS to build the RenderWare Graphics SDK for. For example, *opengl* on *win*, *d3d8* on *win*.

1.1.5 Further reading

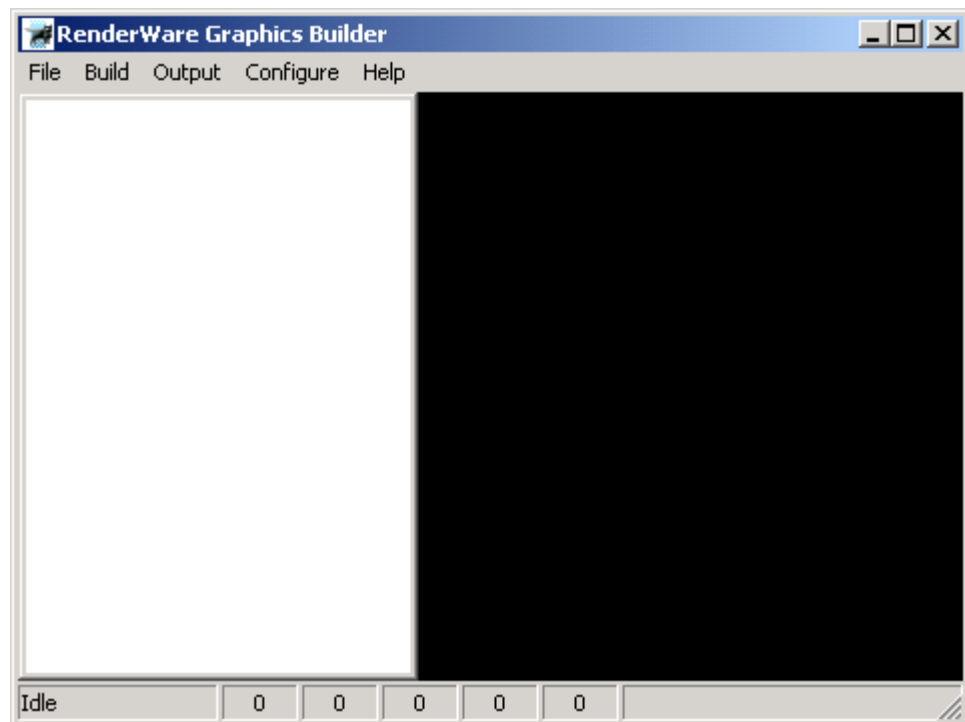
You are recommended to read the "Building RenderWare Graphics" whitepaper supplied with the RenderWare Graphics source distribution for more information on building the SDK.

The gnumake website <http://www.gnu.org/software/make/> may also be of interest.

1.2 Getting started

1.2.1 Introducing the RWB window

Launching RWB for the first time will display a window that looks like this:



The white area to the left of the window is the *tree-view window* that will display your session settings, such as the platforms you want to build a RenderWare Graphics SDK for.

The black area on the right is the *output window* that will display the build output during and after an SDK is built. Dragging the vertical bar separating it from the tree-view window will alter the width of the output window.

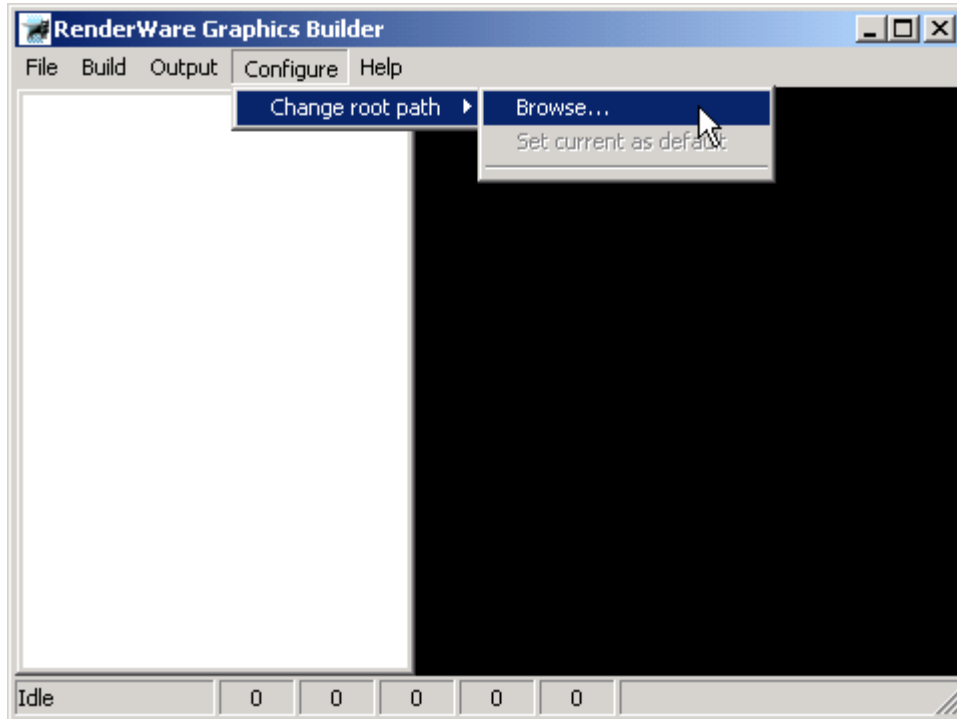
The status bar along the bottom indicates the current build status and five counters. Currently, the system is "Idle" and all the counters are zero. The meaning of the counters will be explained shortly.

The status area on the taskbar also displays an icon (outline of a cube) when RWB is loaded.



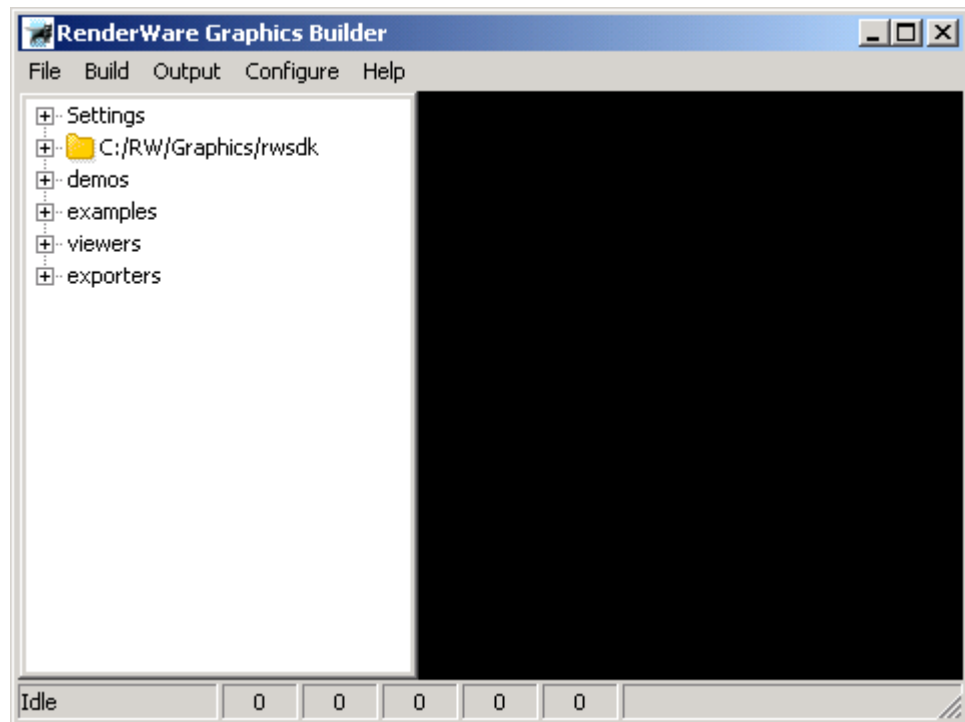
1.2.2 Configuring a source tree

The first thing you must do is set the location of your RenderWare Graphics source tree. Do this by choosing the *Configure* → *Change root path* → *Browse...* menu option.



You will then be presented with an Explorer style dialog that allows you to browse to any folder on your computer/local network. The folder that RWB requires here is the parent of the **rwsdk** directory of the source tree you want to configure. RWB automatically performs a validation whether the selected folder contains a RenderWare Graphics source tree and will inform you if it does not.

Once a valid path has been selected, you will see that the tree-view window configures itself for that source tree.



For example, in the above screenshot, the source tree found at **c:/RW/Graphics** was selected, as the path to the source tree is used as the text for the second item in the tree list.

When a source tree is configured, RWB will automatically search through that source tree determining which platforms are available. For example, if you have the Xbox source distribution then RWB will only show what is available from the Xbox distribution. The screenshots shown in this user guide will therefore not necessarily match what you will see.

It will also search for SDK examples, viewers, and demos. If these are not found, they will not be displayed.

1.2.3 Browsing the tree-view

Generally there will be at least two top-level items on the root of the tree list:

1. *Settings* containing current build options
2. *.../rwsdk* containing the mandatory and extensions of the RenderWare Graphics SDK that can be built

Extra folders will appear on the root of the tree list if the user has configured them. For example, the screenshot above shows four other folders called *demos*, *examples*, and *viewers*. We will explore these, and how to configure them, in more detail later.

Build settings

The *Settings* item on the tree list contains a list of build settings in the order:

- RWOSs
- Build options
- Advanced build options

Each RWOS (or platform) that RWB has detected in your configured source tree contains sub-lists of RWCOMPILERS (what compiler you build the RenderWare Graphics SDK with) and RWTARGETS (which variation of the selected RWOS, e.g. OpenGL or D3D8 for Win).

Expanding the Settings item will show something like this:



This screenshot shows that there are *five* RWOS' configured under the source tree. The *win* RWOS is expanded further to show *ten* possible RWCOMPILERS and *seven* possible RWTARGETS. There can only ever be one compiler selected for each RWOS, but any number of targets may be selected.

There are also other options *RWDEBUG*, *CDEBUG*, *COPTIMIZE*, *RWMETRICS* below the list of RWOSs that provide further specification of what type of binary files for the SDK should be built. These are explained further in the *Building RenderWare Graphics* whitepaper referred to in *1.1.5 Further reading*.

Advanced Options provides an additional list of options that are considered non-standard and so may only be useful to those developers well-versed in RenderWare Graphics.

Selecting RWOS', RWCOMPILERS and RWTARGETS is just a case of clicking on the appropriate item using the mouse.

What the icons mean

Each item has its own state icons, to indicate whether it is selected or not.

A selected RWOS has a "computer" icon. For example, the *win* RWOS is selected here:



A selected RWCOMPILER has a "cog" icon. For example, the *visualc* RWCOMPILER is selected here:



A selected RWTARGET has a "target" icon. For example, the *opengl* RWTARGET is selected here:









A selected build option has a "blocks" icon. For example, the *CDEBUG* option is selected here:



A selected SDK component has a "jigsaw" icon. For example, the *anisot* plugin is selected here:



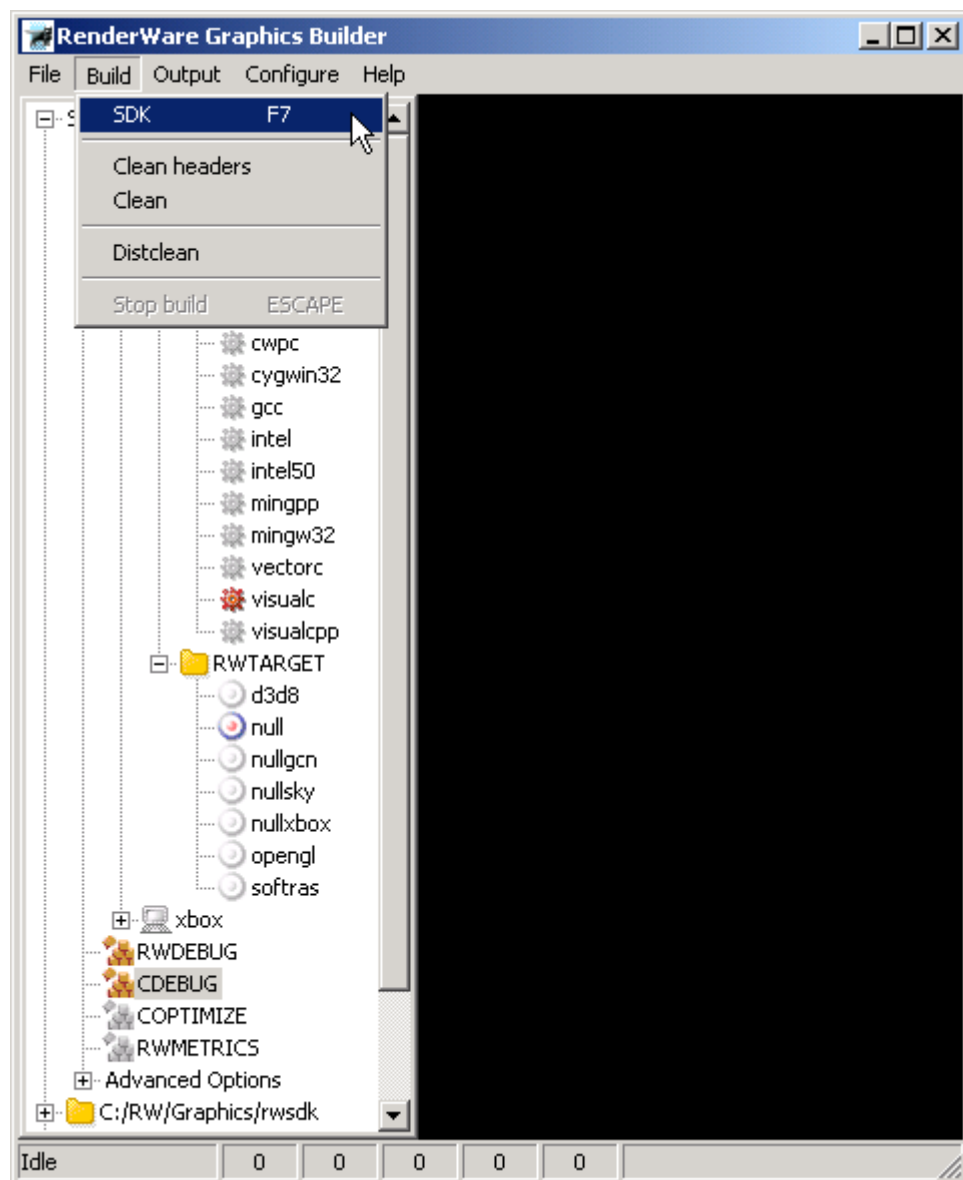
Finally, a folder has three states depending on the states of the items it contains (it's children):

- *no* children selected:   toolkits
- *some* children selected:   C:/RW/Graphics/rwsdk
- *all* children selected:   RWTARGET

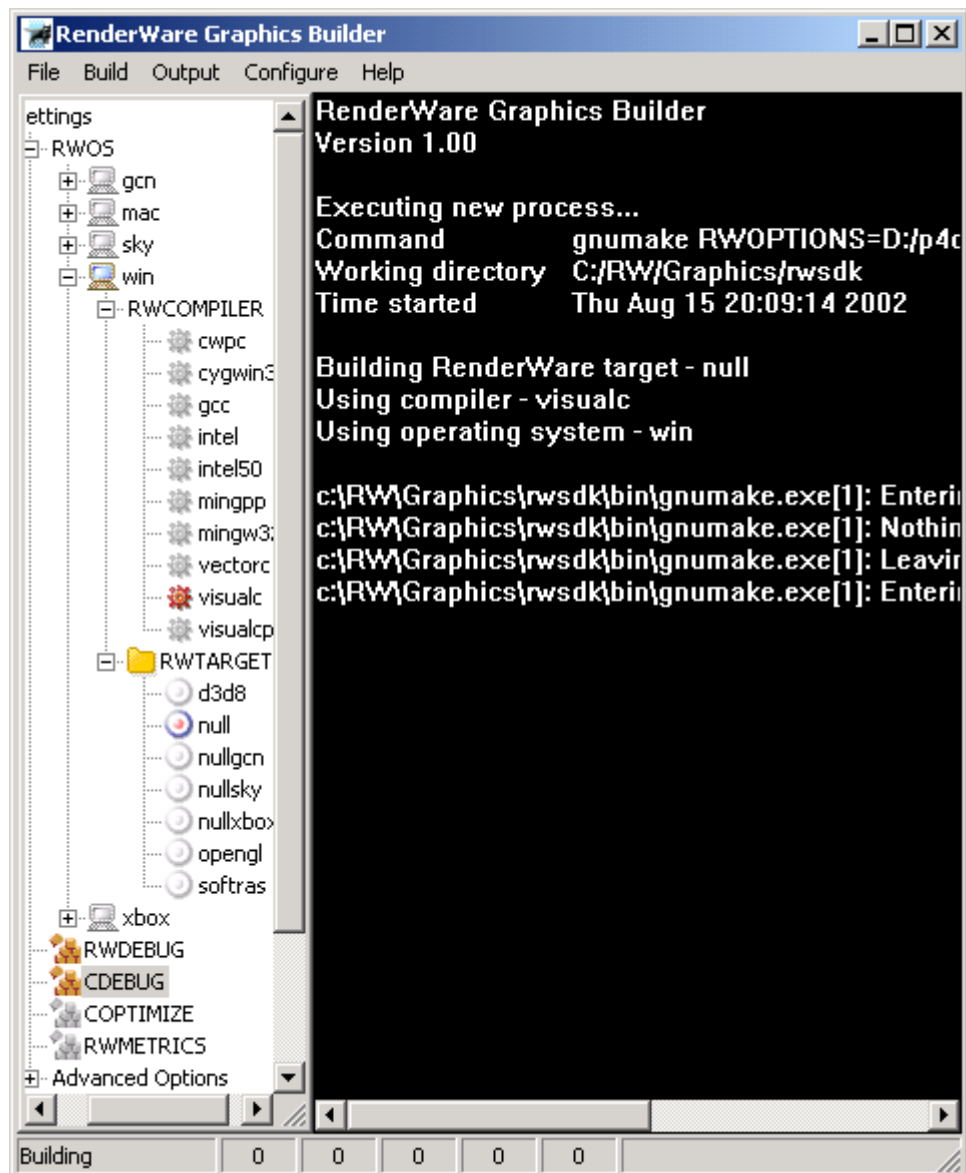
1.2.4 Running a basic SDK build

At this stage, you are nearly able to build a minimal SDK if you so wish, or continue to edit your settings further. If you wish to build then you must select the RWOS, RWCOMPILER and at least one RWTARGET you want to build. These are the minimum requirements for initiating a build. If these are not met, RWB will issue a warning message.

Now select the *Build* → *SDK* menu option to start the build. In the screenshot below, the *visualc* RWCOMPILER is chosen for the *null* RWTARGET, on the *win* RWOS.



Once selected, the build process starts and it's output shown in the output window:



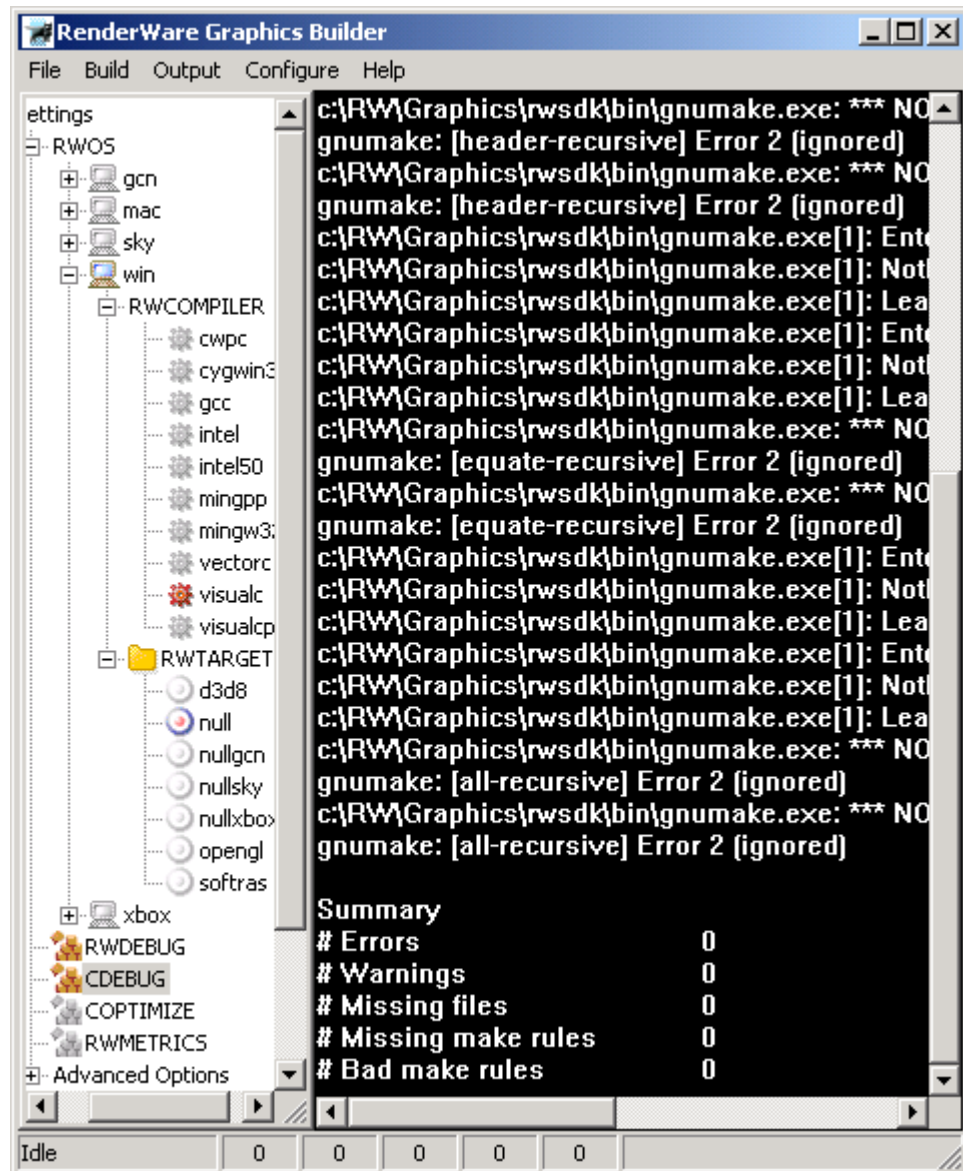
Notice how the text in RWB's status window changes. The mouse pointer will also change to an hourglass to indicate that the build is taking place.

In addition, the status area icon (displayed on the taskbar) will animate during a build. If the RWB main window is minimized or pushed back in the window stacking order, you will still be able to know the status of the build progress. Double clicking on the status area icon will always restore the RWB main window to the top of the window stack.

Before the output of each build is shown, RWB will output what the command being issued is, what directory the command was called in and the time at which the command was executed. This is useful for logging sessions.

As the build progresses, the output window will scroll automatically to show the most recent output. This feature can be disabled, as will be described later on.

During the build, the status bar is updated with the summary of the build in real time. The five numbers to the right of "Building..." or "Idle" text indicate the summary. Once the build has completed, a more descriptive textual summary is shown in the output window, containing the various statistics that RWB gathers as the build progresses:



The summary contains counts of the *errors*, *warnings*, *missing files*, *missing make rules* and *bad make rules* from the most recent build. These are gathered by comparing the output against context strings for each compiler. (See [1.6.8 Compiler context strings](#) for more on where and how these context strings are defined.)

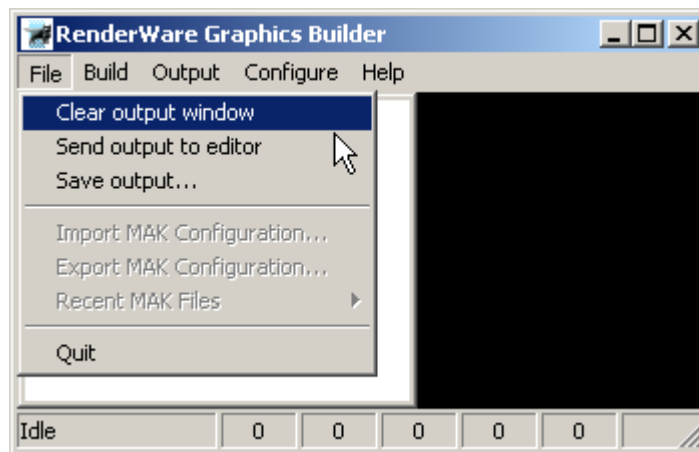
When the compiler context strings are identified in the build, the output text changes color to indicate this. The default colors are:

- Errors red
- Warnings yellow
- Missing files cyan
- Missing make rules magenta
- Bad make rules green

The five numbers, from left to right, on the status bar are in the same order as the summary contexts shown in the output window.

1.3 Menus in detail

1.3.1 File menu



Clear output window allows you to clear the contents of the output.

Send output to editor is for passing the contents of the output window to a text editor. This is achieved through a temporary text file, and uses the Windows file type associations to open the appropriate editor.

Save output... is for saving the contents of the output window to a file. This option will present an Explorer-style dialog box allowing you to browse to a file that will subsequently be written with the entire contents of the output window.

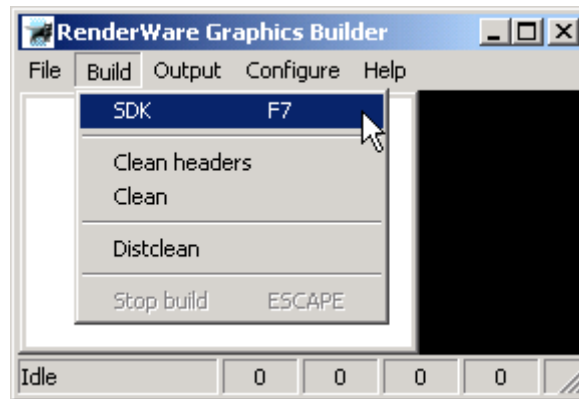
Import MAK configuration... is used to import **.mak** (see 1.1.4 Glossary of terms) files that specify RenderWare Graphics build options, such as those seen in the "Settings" tree list. Loading these files will automatically clear all selections on your tree list and then re-select them as defined in the specified **.mak** file. Note that importing arbitrarily generated **.mak** files is not supported. Use only those **.mak** files that have been exported from RWB first.

Export MAK configuration... is used to export **.mak** files so that the output from RWB can be used from the command line, or restored in RWB at a later date. The user is presented with a dialog that requests the location of each **.mak** file. Each **.mak** file contains one RWTARGET, so if you have three RWTARGETs selected in your tree view, three **.mak** files will be exported, each requiring a different filename.

Recent MAK Files leads to a submenu providing a list of recently imported **.mak** files. This list is refreshed upon each import, and the contents of the list are stored persistently in the resources. Clicking on a filename in this list will load the settings it contains. (See 1.6.9 Recent MAK files for more details on how these files are stored in the RWB resources.)

Quit is used to end the current RWB session. If any modifications to the resources have been made in this session then you will be prompted to save the modified resources.

1.3.2 Build menu



If more than one RWTARGET is selected, then most of the options on the build menu queue a sequence of build jobs (in a top-down sequence as defined in the tree-list) and are built in that order. The first build will fail to begin if there are any mandatory options missing for any target selected.

SDK, as shown previously, is used to build the SDK. This only affects the RWTARGETs selected. Pressing the key bound to BUILDSDK (default F7) will also build the SDK. (See *1.5.2 Customizable shortcuts*.)

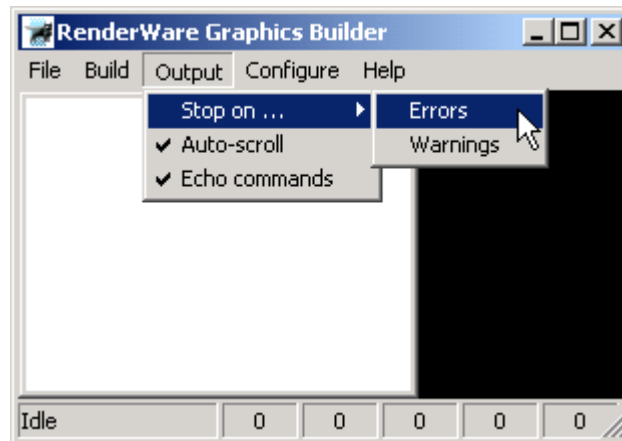
Clean headers is used to erase all generated header files from an SDK. This only affects the RWTARGETs selected.

Clean is used to erase all intermediate files from an SDK. This only affects the RWTARGETs selected.

Distclean is used to erase all intermediate files, generated headers and libraries from an SDK for ALL RWTARGETs. Use this with caution as it removes all generated files. You will be prompted to confirm this action. Note that this requires the NULL win target to be available as this is used to generate a minimal make options file to satisfy the build process.

Stop build is usually grayed out. This option becomes available when a build has started when the other build options become grayed out. Selecting this option will stop the build and all queued builds. Pressing the key bound to STOPBUILD (default Escape) at any time during the build will also stop it. (See *1.5.2 Customizable shortcuts*.)

1.3.3 Output menu



Stop on... leads to a submenu that contains

- *Errors*
- *Warnings*

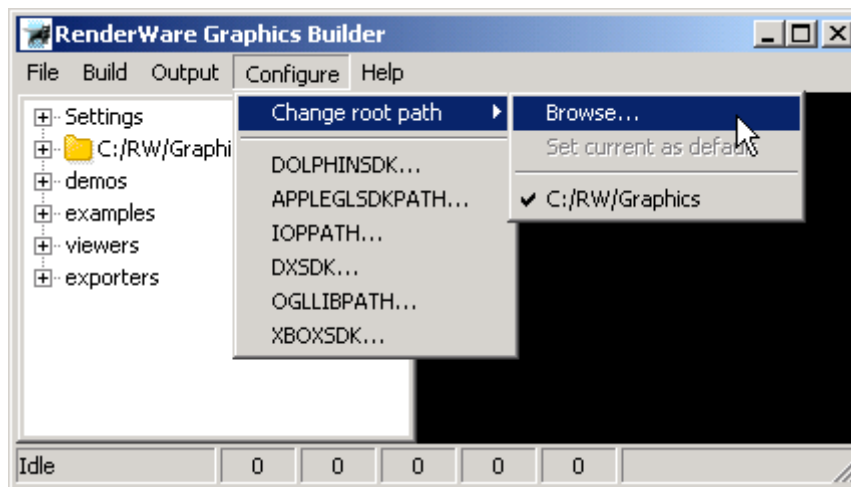
Clicking on one of these toggles between stopping the build on errors/warnings and ignoring them. The output will continue for a few lines after the first stop-instance in case the chosen compiler outputs warnings/errors on multiple lines.

Auto-scroll can be toggled on (checked) or off (unchecked). If this option is on (the default case), the output window will automatically scroll downwards to follow the progress of the build. If it is unchecked then the user has control over the scroll bar location, in order to view any part of the build output (see [1.5 Keyboard shortcuts](#)). In both cases, the scroll bar moves/resizes appropriately to show the location of the current page shown in relation to the whole output.

Echo commands can be toggled on (checked) or off (unchecked). If this option is on (the default case), all commands from **gnumake** are passed to RWB and displayed in the output window. If this option is off, no commands from **gnumake** are passed to RWB. However, any output from any child processed called by **gnumake**, such as compilers and linkers, will still be passed to RWB. This option only has an effect *before* a build is started, so during a build it is grayed out.

These options are restored persistently in the resources file. (See [1.6.12 Output options](#) for more details about how these paths are stored in the RWB resources.)

1.3.4 Configure menu



Change root path leads to a submenu containing at least two items:

- *Browse...* that allows you to browse for a new root path via an Explorer-style dialog, and RWB will determine what it displays on it's tree view according to the contents of that root path.
- *Set current as default* that is grayed out until two or more root paths can be toggled between. When available, this option allows the user to choose the default root path that RWB initializes to next time the application is started.

The remaining items in the change root path submenu, below the separator line, consist of a list of root paths that have been used previously, or browsed to in this RWB session. The root path that RWB is currently synchronized to (and hence should match that shown in the tree-list) will be ticked. The first root path visible in the list is the default root path. The list of root paths is stored in the resources file. (See *1.6.1 Root paths* for more details on how these paths are stored in the RWB resources.)

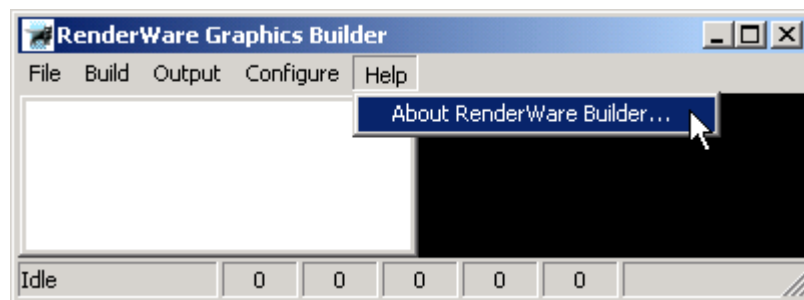
RWOS build paths may or may not appear below *Configure → Change root path* depending on what RWOS' are in your source tree. Currently the following are supported:

<u>RWOS</u>	<u>Build paths</u>
Win	DXSDK
Win	OGLLIBPATH
Sky	IOPATH
GCN	DOLPHINSDK
GCN	GDEV
GCN	GAMEOPTIX

Mac	APPLEGLSDKPATH
Xbox	XBOXSDK

Selecting any of these allows you to browse to the paths containing the necessary SDK/APIs using an Explorer-style dialog. These are stored in the resources file. (See *1.6.7 Operating System Build Paths* for more details about how these paths are stored in the RWB resources.) Initially these paths are not set up so you must configure the appropriate paths for the platform you wish to build the RenderWare Graphics SDK for.

1.3.5 Help menu

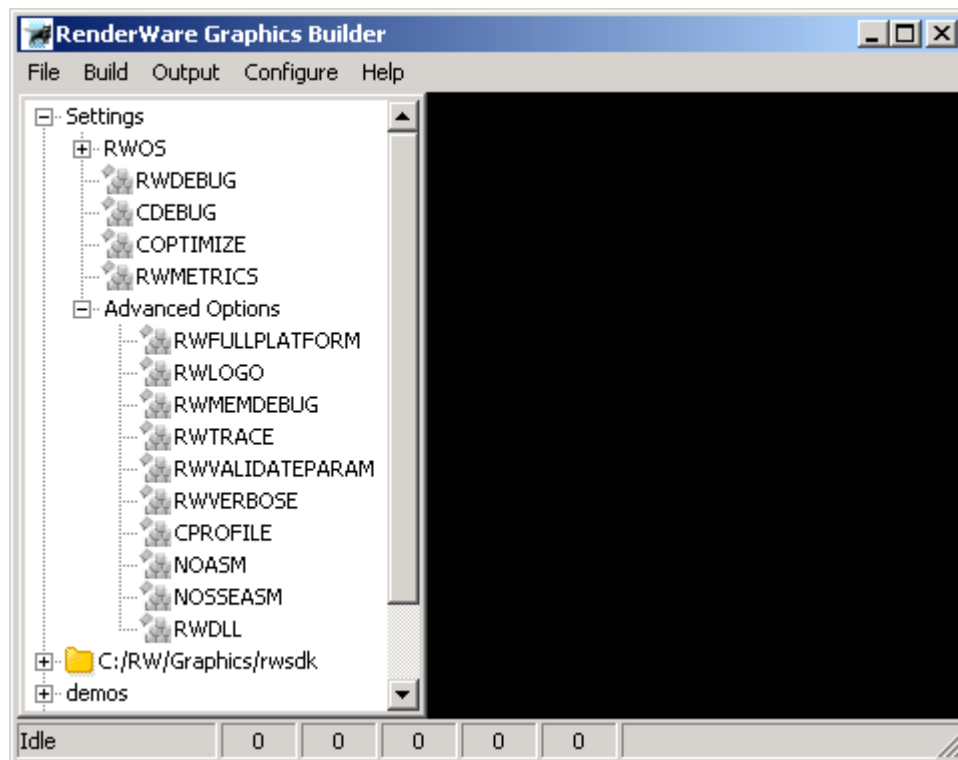


The help menu can be used to determine version information about RWB.

1.4 Tree lists in detail

1.4.1 Settings tree

The *Settings* tree generally appears in the following form:



The *RWOS* expandable tree item has been illustrated earlier, and contains a list of *RWOS*'s, *RWCOMPILERS* and *RWTARGETS*.

The settings immediately below *RWOS* define the various options that can be applied to the build process in order to obtain different types of libraries. These options all have tool tip help associated with them to describe their uses. Each setting can be toggled on or off. Those that are on will be used in subsequent builds.

1.4.2 SDK components tree

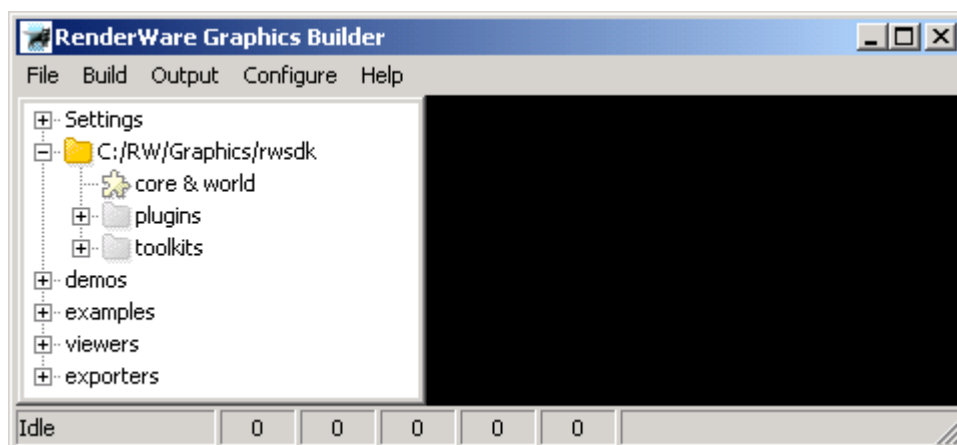
The SDK components tree list is identified by the pathname to the currently configured source tree. This is further divided into

- *core & world*
- *extensions*

The core and world plugin are currently always built. The extensions on the SDK components tree are completely optional, and should be selected in order to build them, although we will describe an alternative approach to building them separately.

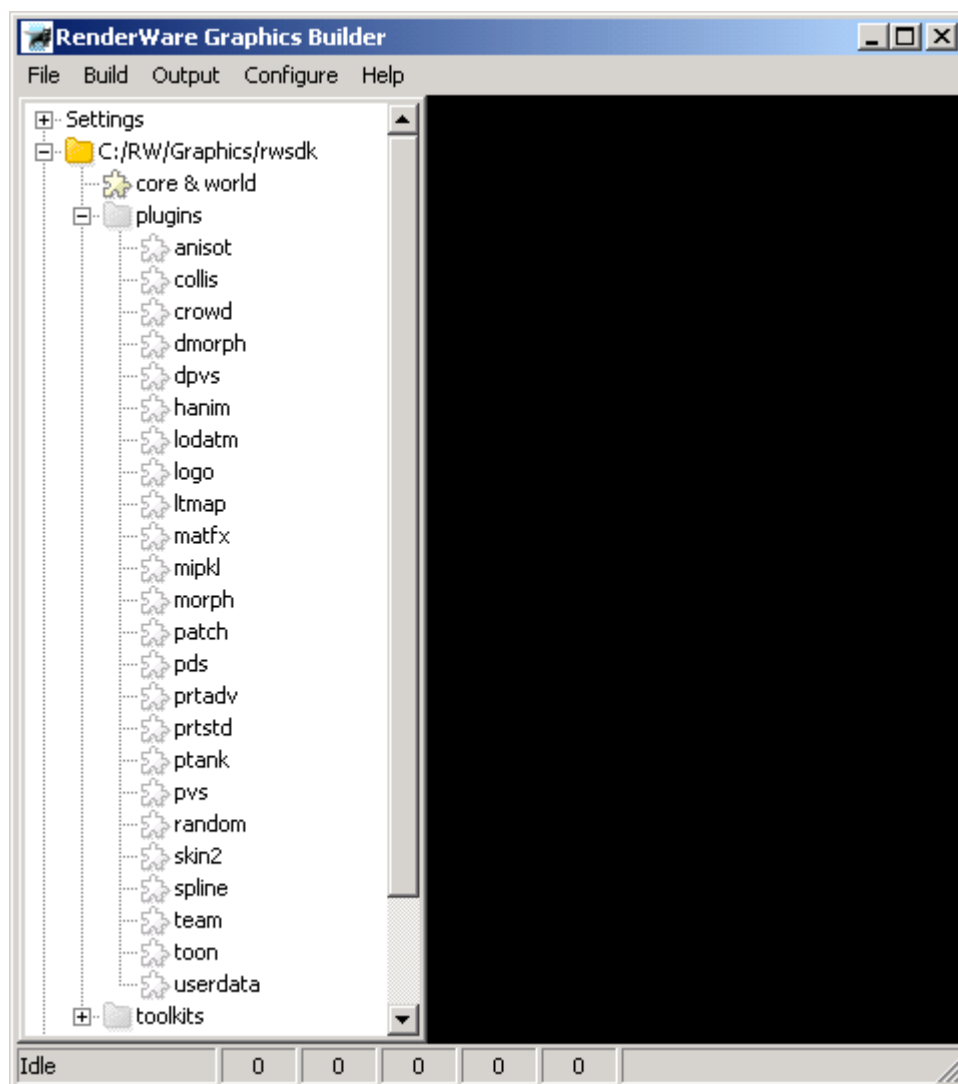
The extensions are defined in the resources, and are discussed in more detail later on (see *1.6.3 RenderWare Extension Paths*).

The screenshot below shows just the core and world plugin selected for building. Here there are three extension paths defined and available in the selected source tree: *plugins*, *development plugins*, and *toolkits*.

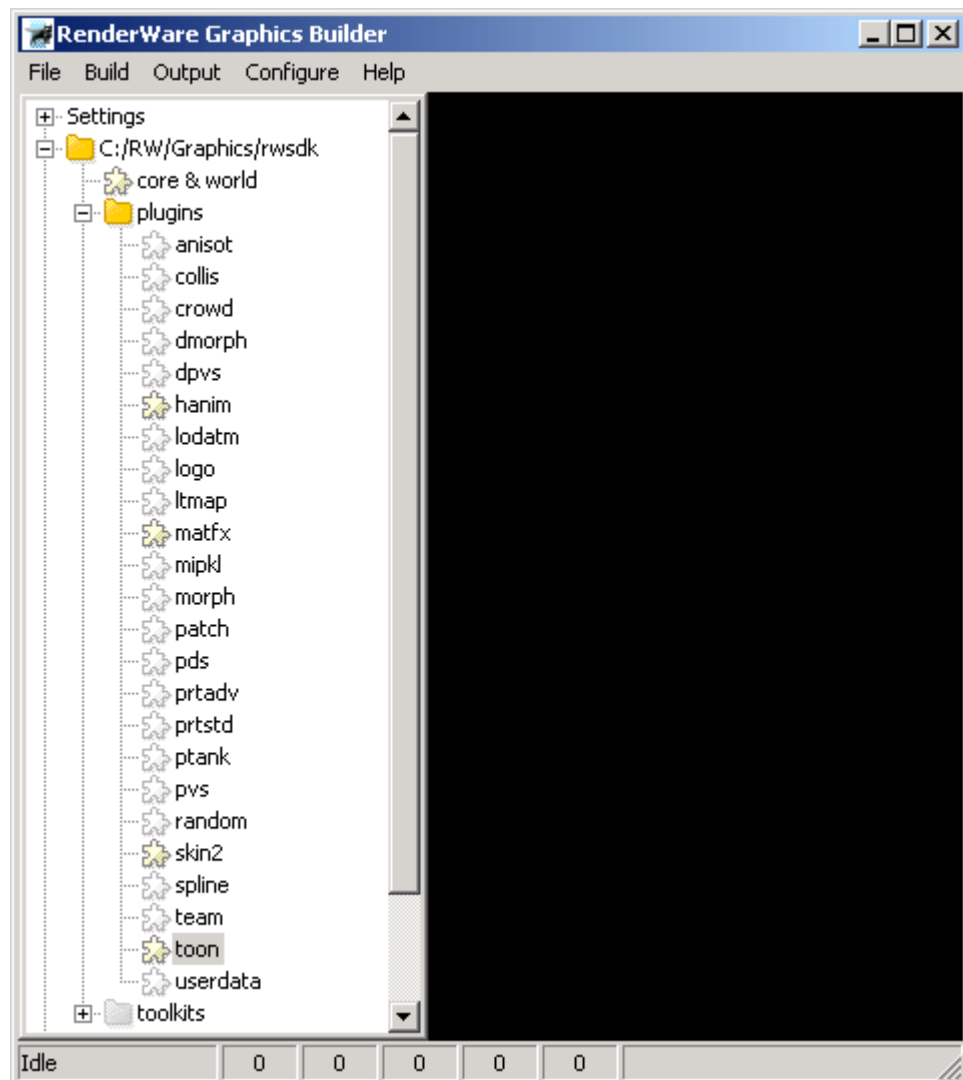


Each of the extension paths on the SDK components tree are treated in the same way. Each provides a list of extensions that they group together, e.g. plugins, and may either be *all selected*, *some selected*, or *none selected* for the next build. Clicking on the parent folder of an extension group toggles between all and none selected, whereas clicking on individual extensions toggles the parent folder between all three states.

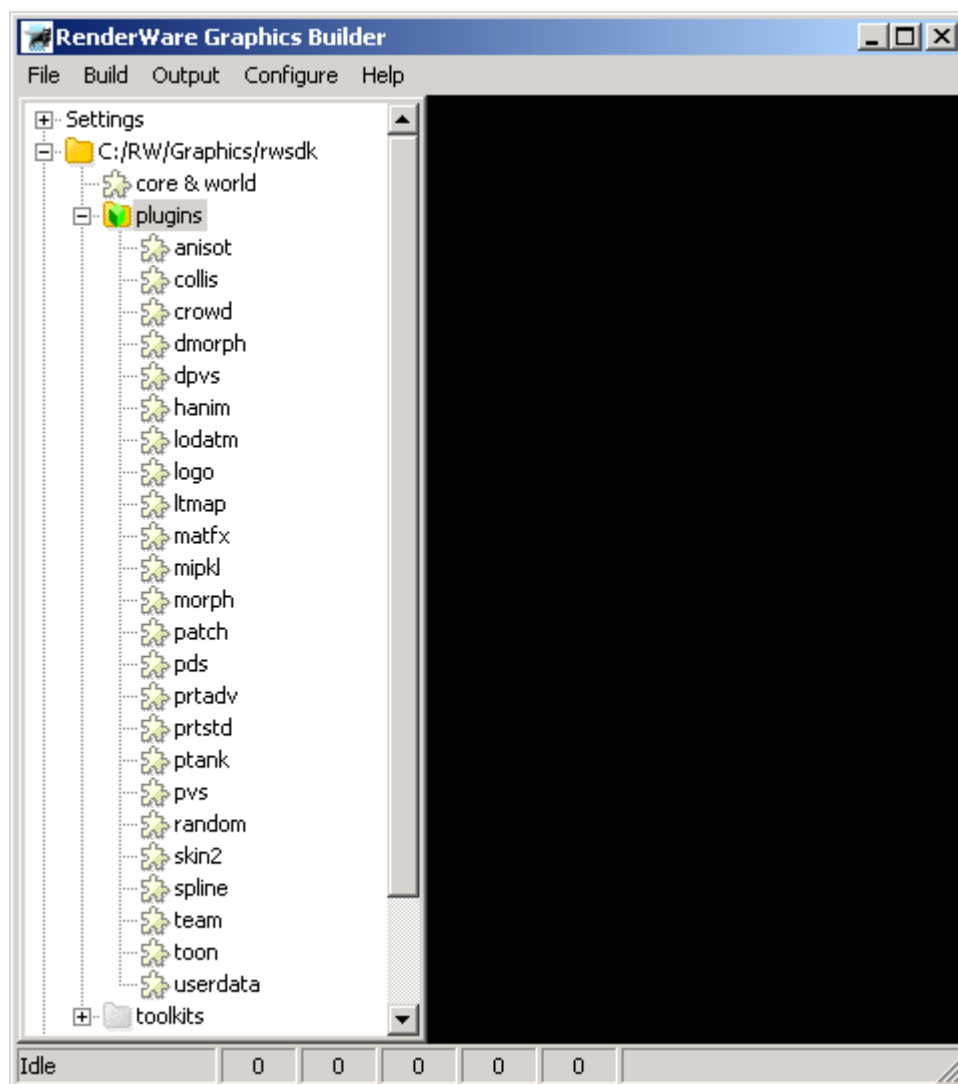
The screenshot below shows no plugins selected for building.



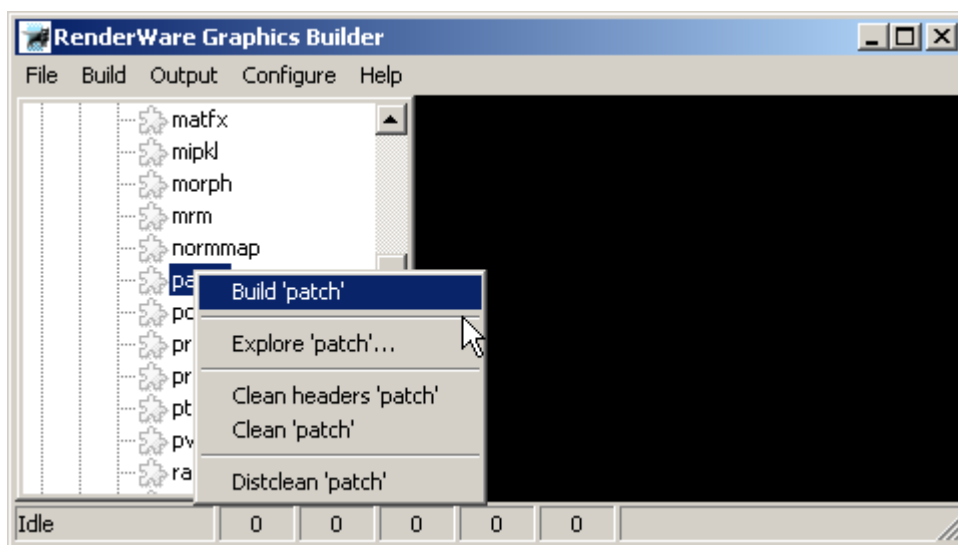
The screenshot below shows four plugins selected for building: *hanim*, *matfx*, *skin2*, and *toon*.



The screenshot below shows all the plugins selected for building.



It is also possible to build extensions on an individual basis, away from the main RWB build system. By right clicking over a plugin (for instance), opens a build context menu, such as shown below for the '*patch*' plugin:



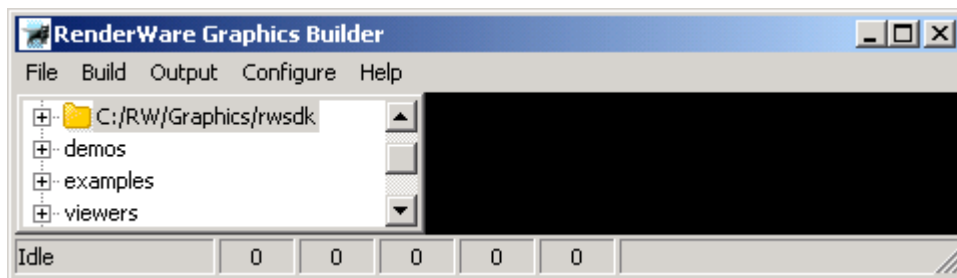
The menu options are similar to those from the build menu, except *Stop* build is not present, and an *Explore* option is available. All options, except for *Explore*, like for the main build menu, will be grayed out once a build begins.

This build method is useful for debugging local changes to a particular SDK component.

The *Explore* option can be used to open Windows Explorer to the appropriate location on your hard drive. For example, for the patch plugin, it would open `c:/rw/graphics/rwsdk/plugin/patch` if you installed your RenderWare Graphics SDK to that location.

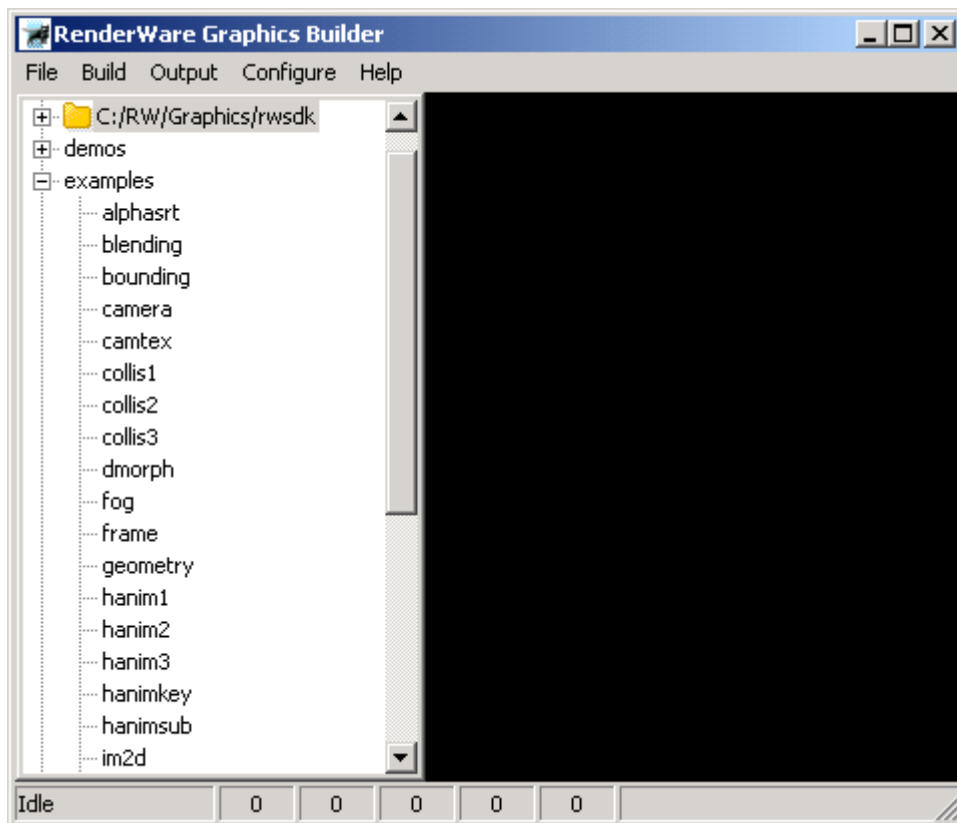
1.4.3 Applications Tree

Below the SDK components tree, application trees are placed off of the root. Which applications trees are visible depends on what is available in your selected source tree and what application paths have been defined in the RWB resources (see *1.6.4 RenderWare Application Paths*). In the screenshot below, there are three application trees: *demos*, *examples*, and *viewers*.

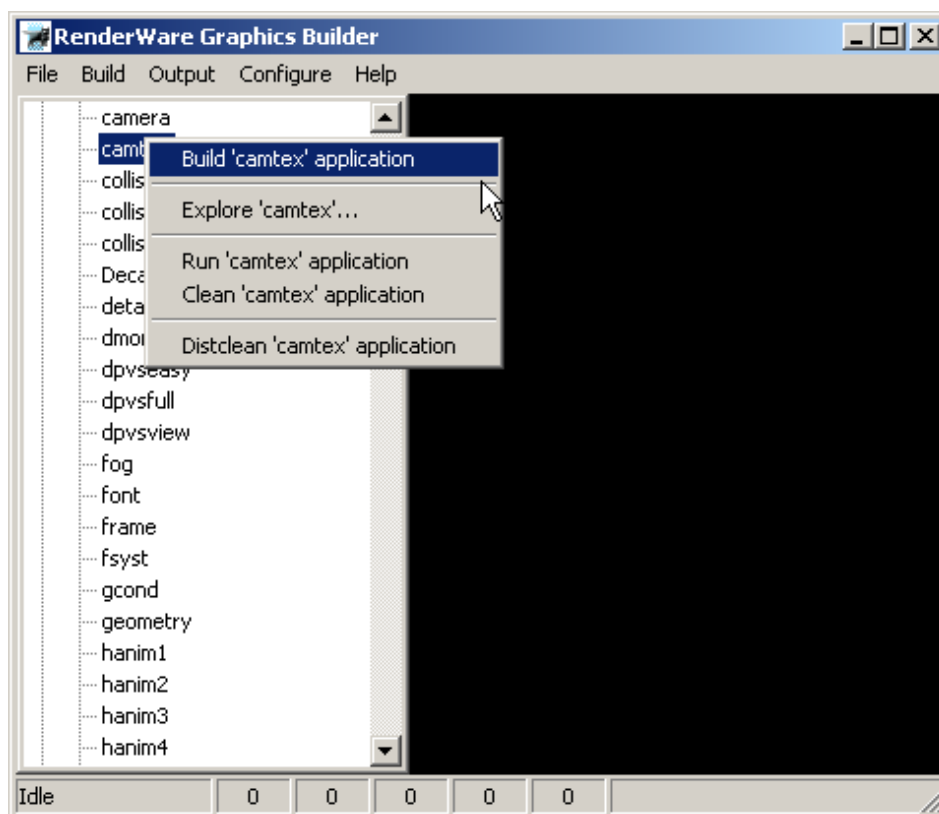


Building an application

The screen shot below shows a possible list of examples:



In order to build applications using the same options as the SDK you must make use of the context menus again. For example, right clicking over the 'camtex' example to display its build context menu:



Note that this context menu is subtly different from that displayed over an SDK component. This is because the *Clean headers* option is not applicable to an application. Hence that option has been replaced by *Run* to execute the example (provided it was built successfully).

Note that RWB uses the application's **makefile** to perform the execution.

1.5 Keyboard shortcuts

There are two sets of keyboard shortcuts: those that are customizable and those that are not. How to customize the keyboard shortcuts are described in *1.6.10 Keyboard shortcuts*.

1.5.1 Fixed shortcuts

When there is text shown in the output window there are a number of keyboard shortcuts that can be used to navigate. (Remember to disable auto-scrolling while the build is in progress to make effect use of the navigation shortcuts.)

- Cursor up move up a line
- Cursor down move down a line
- Page up move up a page
- Page down move down a page
- Home move to the beginning of the output
- End move to the end of the output

1.5.2 Customizable shortcuts

More general keyboard shortcuts are available through the resources. These define the following commands:

- Build the SDK for the currently selected targets (default F7).
- Stop any build in progress (default ESCAPE).
- Display the previous context line in the output window (default SHIFT+F4).
- Display the next context line in the output window (default F4).

The latter two commands enable the user to browse the build output for lines of particular interest, e.g. warnings, errors etc, that RWB has detected.

The key bindings for the shortcuts may be edited in the resources.

1.6 Resources file

The resources file, `rwbuildres.cfg`, may be edited by the user if they so wish. Note, however, that the internal format may be subject to change. If possible, and if the interface exists, any changes should be made through the GUI. Changes to the resources file should only be made when RWB is not running as RWB maintains an internal copy of the resources that are used to write back unmodified sections upon shutdown. Thus, any local changes you make may be lost.

1.6.1 Root paths

This resource stores a list of root paths that are visible in RWB from the *Configure* menu. The resource takes the form:

```
[ROOTPATHS]
{
    C:/rw/graphics
    C:/rwg34
}
```

Here there are two root paths defined. The top-most root path is always the default root path, i.e. that path that RWB initializes to upon startup. All root paths are validity tested upon resource load and if any are not found to be valid a message will be presented to the user. Any invalid paths are not added to the Configure menu and will not be written back if the resources are modified.

It is highly advised to use the GUI interface in RWB to manipulate the root paths.

1.6.2 RenderWare Internal Paths

This resource is for internal use to RWB and should not be edited.

1.6.3 RenderWare Extension Paths

This resource defines the extensions to the RenderWare Graphics SDK that RWB should search for and provide access to on its tree list. This resource takes the form:

```
[RENDERWAREEXTENSIONPATHS]
{
    [PLUGINS]
    {
        plugins
        /rwsdk/plugin
        Select any plugins within this tree to build.
    }
}
```

Within the `RENDERWAREEXTENSIONPATHS` group, you can define any number of subgroups that represent each extension to the SDK. For example, the `PLUGINS` group is defined above.

The format for each extension subgroup is:

```
[<NAME>]
{
    <tree list header>
    <relative path>
    <tool tip text>
}
```

`<NAME>` is the text that is written to the `options.mak` file when a build occurs.

`<tree list header>` appears in the tree list to identify the extension.

`<relative path>` is used by RWB to enumerate all the associated extensions, and is relative to the root path.

`<tool tip text>` appears in a tool tip dialog when the cursor moves over the extension in the tree list.

1.6.4 RenderWare Application Paths

This resource defines the applications that use the RenderWare Graphics SDK that RWB should search for and provide access to on its tree list. This resource takes the form:

```
[RENDERWAREAPPLICATIONPATHS]
{
    [DEMOS]
    {
        demos
        /demos
        These are RenderWare Graphics demos.
    }
}
```

The format of each application sub-group, e.g. `DEMOS` in the example above, is the same as that for the RenderWare Graphics extension paths.

1.6.5 Target bindings

This resource defines how RWTARGETs become bound to RWOSs. The format of this resource is:

```
[TARGETBINDING]
{
    [gcn]
    {
        gcn
    }
}
```

Each RWOS defines a subgroup, and the data within these sub-groups are the viable RWTARGET names. If RWB does not recognize an RWTARGET then it will ignore it.

1.6.6 Advanced options

This resource defines the advanced options available on the Settings tree. The format of this resource is:

```
[ADVANCEDOPTIONS]
{
    [OPTIONS]
    {
        NOASM
        NOSSEASM
    }
    [OPTIONSTOOLTIPS]
    {
        This option disables all assembly compilation.
        This option disables all SSE assembly compilation.
    }
}
```

This resource has two mandatory sub-groups, `OPTIONS` and `OPTIONSTOOLTIPS`.

The former sub-group contains a list of the options that will be written out to the `options.mak` file during a build. These options can only take the form

```
OPTXXX=0 | 1
```

Advanced options with other `.mak` file formats are not supported.

The latter sub-group provides a list of tool tip help to display over the corresponding option in the tree list.

1.6.7 Operating System Build Paths

This resource defines the OS-specific build paths necessary for RenderWare Graphics to build. This resource takes the form:

```
[OSBUILDPATHS]
{
  [mac]
  {
    APPLEGLSDKPATH
    c:/sdk/appleopengl
  }
}
```

Each RWOS defines a sub-group of this resource, and there are pairs of data items for each sub-group in the order `BUILDPATHNAME` followed by `PATH`. Hence, in the example shown, the Mac OS has a single build path defined, called `APPLEGLSDKPATH`, and is located at `c:/sdk/appleopengl`. The name of the build path is used in the `options.mak` files during builds.

It is highly advised to use the GUI interface in RWB to modify these build paths.

1.6.8 Compiler context strings

This resource defines context strings for warnings and errors for each `RWCOMPILER`. This resource takes the form:

```
[COMPILERCONTEXTINFO]
{
  [cwppc]
  {
    [warning]
    {
      Compiler Warning:
    }

    [error]
    {
      Compiler Error:
    }
  }
}
```

Each compiler (as defined in RenderWare Graphics terms, `RWCOMPILER`) defines a sub-group of this resource. Each sub-group then has two mandatory sub-groups: *warning* and *error*. Each of these subgroups then contains a list of terms that RWB uses to search the build output for to identify warnings and errors.

If a compiler used by RWB is not defined in the resources, then RWG searches for the strings “error” and “warning” by default. This may contexts to be incorrectly identified if a compiler setting used to build RenderWare Graphics also contains instances of these default search terms. This can be resolved by defining a compiler context group in the resources with very specific search terms for the compiler.

1.6.9 Recent MAK files

This resource defines a list of recently imported **options.mak** files. It takes the form:

```
[RECENTOPTIONSMAKFILES]
{
    c:/rw/graphics/d3d8dbg.mak
    c:/rw/graphics/openglrel.mak
}
```

This resource contains a list of **.mak** files to appear on the File menu but is maintained internally by RWB.

1.6.10 Keyboard shortcuts

This resource defines several configurable commands that can be issued through keyboard shortcuts. It takes the form:

```
[SHORTCUTKEYS]
{
    [BUILSDK]
    {
        F7
    }
}
```

Each subgroup defines the command that is currently any of the following:

- BUILSDK
- STOPBUILD
- PREVIOUSCONTEXT
- NEXTCONTEXT

The keystrokes that are necessary to issue the command are then provided as the data to these subgroups. There can be one or more keys necessary to issue a command, for example SHIFT and F4.

The keys that are currently recognized in this context are:

- SHIFT (either left or right)
- CONTROL (CTRL, left or right)

- ESCAPE
- F1—F9

The shortcuts bound to menu options also appear on the menu.

1.6.11 Compiler setup

This resource defines the requirements of each compiler in order for RWB to perform some sanity checks on the build environment.

It takes the form:

```
[COMPILERSETUP]
{
    [visualc]
    {
        [toolchain]
        {
            cl.exe
            link.exe
            lib.exe
            ml.exe
        }

        [setuphelp]
        {
            See the file vcvars.bat in your Visual Studio
            installation.
        }

        [includeenvvar]
        {
            INCLUDE
        }

        [libraryenvvar]
        {
            LIB
        }
    }
}
```

Each compiler (as defined in RenderWare Graphics terms, RWCOMPILER), and then has four mandatory subgroups.

toolchain defines the executables that make up the set of tools necessary to build RenderWare Graphics using this compiler.

setuphelp specifies text to be output to the screen to suggest additional compiler set up help if RWB detects something is wrong.

includeenvvar specifies a list of environment variables that the compiler relies upon to exist to find standard header files.

libraryenvvar specifies a list of environment variables that the compiler relies upon to exist to find standard library files.

For the latter two subgroups, each environment variable that they contain may be marked as “optional”, as some compilers use generic and specific variables. To mark an environment variable as optional, post-fix the name with an asterisk *. For example, the CodeWarrior PS2 compiler is set up with

```
[includeenvvar]
{
    MWCPS2Includes
    MWCIncludes*
}

[libraryenvvar]
{
    MWPS2MIPSLibraries*
    MWPS2MIPSLibraryFiles*
    MWLibraries*
    MWLibraryFiles*
}
```

indicating all but **MWCPS2Includes** are optional.

Any environment variables that are optional, and do not exist when a build is executed, will have a minor warning message box issued. This will not stop the build, like a missing mandatory environment variable will. This warning message box is issued only once for the lifetime of the application so as not to become an annoyance.

1.6.12 Output options

This resource persistently stores the output options from the output menu. It takes the form:

```
[OUTPUTOPTIONS]
{
    [AUTOSCROLL]
    {
        Yes
    }
}
```

Each subgroup defines the option that is currently any of the following:

- AUTOSCROLL
- ECHOCOMMANDS

- STOPONERRORS
- STOPONWARNINGS

The subgroups contain one item of data, either Yes or No, indicating whether the options is used or not.