Revolution

Build System

Version 1.02

"Confidential"

Revolution Build System

Version 1.02

Contents

Revision History ii
1 Overview 1
2 Assumptions 3
3 Running a Simple Demo 5
3.1 Running a Demo from the Command Line 5
3.2 Running a Demo from the Debugger 5
4 Building a Sample Application 7
4.1 Platforms 7
4.2 DEBUG and NonDebug Builds 7
4.3 Cleaning 8
4.4 Build Products 8
4.5 Building Individual Executables 8
4.6 Adding Files 9
4.7 Creating Your Own Modules 10
5 Build Tree Overview 11
5.1 Interfaces 12
5.2 Build Products 13
5.3 Source Code 14
5.4 Demo Modules 15

Code Examples

Figures

Figure 1 - Top-level SDK Overview 11
Figure 2 - Structure of Include Directories 12
Figure 3 - Libraries and Executables 13
Figure 4 - Source Tree 14
Figure 5 - Demo Module Structure 15

RVL-06-0044-001-C
Released: August 24, 2006

Revision History

Revision No.	Date Revised	Items (Chapter)	Description
1.02	8/15/2006	3	Revised the methods for running demos and debugging.
1.01	4/7/2006	3.2	Added a note specific to debugging the samples.
1.00	3/22/2006	-	First release by Nintendo of America, Inc.

1 Overview

This document provides an overview of the build system for the Revolution SDK.

The build system is similar to that of Nintendo GameCube™. For full details on the build system, please refer to the Nintendo GameCube version of this document.

2 Assumptions

This document assumes that the user has successfully configured the hardware and installed the components as described in the Quickstart Guide for the Revolution NDEV Development Environment (`RVL-QuickStartGuide.us.pdf`). This includes:

- Cygwin Tools (including GNU make 3.80)
- Metrowerks Codewarrior 3.0 Alpha X
- Revolution SDK
- NDEV Disk Emulation System

For brevity's sake, we assume that the user has installed Revolution SDK under the `/RVL_SDK` folder in the examples throughout this document.

This document also assumes a basic working knowledge of the GNU make tool.

3 Running a Simple Demo

3.1 Calling the Revolution Shell

The batch file, `RVL_NDEV.bat`, initializes the environment variables and calls a Cygwin bash shell. To build or debug, use `RVL_NDEV.bat` to call a shell.

It is worth noting that the environment variables are valid as local settings only for the duration of the shell session (and any child sessions). These settings are neither permanent nor will they effect other areas of the system. As a result, multiple build environments, such as those for the Nintendo DS™ and the Nintendo GameCube, can exist on the same PC.

Be aware, however, of the following:

- You should only call the Revolution SDK build system from this shell. Build systems for other platforms are not guaranteed to work correctly.
- The shell only supports the bash protocol.

3.2 Running a Demo from the Command Line

Open a Revolution shell. From the command line, type:

```
% cd /RVL_SDK/RVL/bin/demos/gxdemo
% ls
```

The "`*.elf`" files are executables that run on the NDEV. Debug versions have the suffix "`*D.elf`".

To run one of these executables (for example, smp-onetriD.elf), simply type:

```
% ndrun smp-onetriD.elf
```

3.3 Running a Demo from the Debugger

To run the executable from the CodeWarrior IDE/debugger, it must be called from the Revolution shell.

The CodeWarrior IDE/debugger is unique to Revolution. It must be called from this shell session. To launch the IDE/debugger, enter the following:

```
%rvl_ide.sh
```

Once it launches, Revolution project files and ELF files can be dragged and dropped into the debugger.

If this is the first time you are debugging this particular application, the IDE will generate a project for it. This will take a moment as the debugger locates all of the sources referenced in the executable.

The project file is stored as an .mcp file in the same directory as the application. If you delete this .mcp file, all of your project settings will be lost.

The pre-built demos included in the Revolution SDK cannot be debugged as is. To debug, you must rebuild each in its own environment.

As of this writing, the Revolution SDK ships with the same debugger as Nintendo GameCube. For more details on how to use this debugger, please refer to the Nintendo GameCube Programmer's Guide under Build System.

An entirely new IDE is forthcoming, and this document will be updated accordingly.

4 Building a Sample Application

The Revolution SDK includes the samplebuild module as an example of a simple application and library. From the command line:

```
% cd /cygdrive/c/RVL_SDK/build/samplebuild
```

This is a sample application which includes a library upon which the application depends. Developers can use this a framework to start a new project.

To build from the command line:

```
% make
```

4.1 Platforms

Note that the default hardware platform is RVL. Other platforms are for internal use only and not supported. However, future revisions of the NDEV system may require re-definition of the PLATFORM flag to build hardware-appropriate code.

To explicitly define the platform, type:

```
% make PLATFORM=RVL
```

4.2 DEBUG and NonDebug Builds

The default target is DEBUG, in which the compiler does no optimizations. Furthermore, all of the system libraries perform additional sanity checking. Note that debug symbolics are NOT included in the system libraries.

To explicitly build a DEBUG target, type:

```
% make DEBUG=TRUE
```

To build a release (optimized) version:

```
% make NDEBUG = TRUE
```

For NDEBUG builds, the compiler performs optimizations but only file-level interprocedural analysis, biasing for speed rather than code size. Note that debug symbolics are not included.

4.3 Cleaning

To remove all objects, dependency files, and disassembly files, type:

```
% make clean
```

This will clean the samplebuild project for both DEBUG and NDEBUG targets.

To remove all executables and libraries, type:

```
% make clobber
```

To remove all executables and libraries by folder, type:

```
% make superclobber
```

4.4 Build Products

The library associated with the samplebuild module will be located here:

`/RVL_SDK/build/samplebuild/sample/lib/RVL/samplelibD.a`

The executable will be located here:

`/RVL_SDK/build/samplebuild/sample/bin/RVL/samplebinD.elf`

Note that both the library and executable have "`*D.a`" and "`*D.elf`" suffixes; this indicates that they are *debug* versions.

Non-debug versions will appear in the same directories, and will not have the "D" suffix.

Also note that the `bin/` and `lib/` directories are further organized by hardware platform. In this example, all build products are placed under the `RVL/` directory.

4.5 Building Individual Executables

To explicitly build a given executable, you can type:

```
% make bin/RVL/samplebinD.elf
```

This will build the debug version of the samplebin executable.

To build the non-debug version, type:

```
% make bin/RVL/samplebin.elf
```

Note that the appropriate (debug or non-debug) libraries must have already been built for this to work.

4.6 Adding Files

In this example, we'll add source files to the src/ directory of the samplebuild module. One file will be added to the executable; the other to the library.

Ensure that you are in:

```
/RVL_SDK/build/samplebuild/sample/src/
```

Create the files:

```
% touch newlibfile.c
% touch newbinfile.c
```

You can leave the files empty.

Open the makefile in a text editor:

```
/RVL_SDK/build/samplebuild/sample/makefile
```

Add newlibfile.c to the CLIBSRCS variable:

```
CLIBSRCS = samplelib.c newlibfile.c
```

This informs the build system that these two files should be built and linked into the library specified by LIBNAME (samplelib in this case). Note that if LIBNAME is undefined, no library is created.

Add newbinfile.c to the CSRCS variable:

```
CSRCS = samplebin.c newbinfile.c
```

This informs the build system that these two files should be built. It does not indicate which executables they should be linked into. That's done with the dependency line at the bottom of the makefile. Modify the dependency line to add the new object file:

```
$(FULLBIN_ROOT)/samplebin$(BINSUFFIX): samplebin.o/
newbinfile.o/
lib/$(ARCH_TARGET)/samplelib$(LIBSUFFIX)/
$(REVOLUTION_LIBS)
```

This dependency tells the build system when to re-link samplebin, and what should be linked. Note that the build system will only look at this dependency if the binary name, "samplebin", has been added to the BINNAMES variable.

Now, invoke a build:

```
$ cd /RVL_SDK/build/samplebuild/sample
$ make
```

You will see the build system compile and link the newly added files.

To see a simple example of how multiple executables can be created from one makefile, examine the demo:

```
/RVL_SDK/build/demos/wpaddemo
```

4.7 Creating Your Own Modules

The simplest way to create a new module is to copy an existing `samplebuild` module.

From the bash shell:

```
cp -r /RVL_SDK/build/samplebuild/sample /RVL_SDK/build/samplebuild/mymodule
```

This creates a new module "`mymodule`" at the same directory level as "`sample`".

Open the makefile in `mymodule/` with a text editor, and change the `MODULENAME` variable:

```
MODULENAME = mymodule
```

This variable simply indicates where in the `samplebuild` tree this module exists. The build system knows this module is in the `samplebuild` tree because the variable `SAMPLE` is set to `TRUE`.

If we are creating a new library that will be linked in by other modules, the library must be installed into a public location. This is achieved by adding the "`install`" target to the "`all`" target:

```
all: setup build install
```

Each of these three targets is defined by the build system.

The "`install`" target will make the build system copy the library into `/RVL_SDK/RVL/lib`. Other executables that wish to link in the library will simply include it in their dependency line, like so:

```
/RVL_SDK/$(ARCH_TARGET)/lib/samplelib$(LIBSUFFIX)
```

Note that the use of `$(ARCH_TARGET)` and `$(LIBSUFFIX)` allow the build system to use the same dependencies for different kinds of builds.

The library name can be changed by modifying the `LIBNAME` variable. You should not add any suffix, as the build system does that automatically to re-use the same root names across different builds. The real name of the library for a particular build is always:

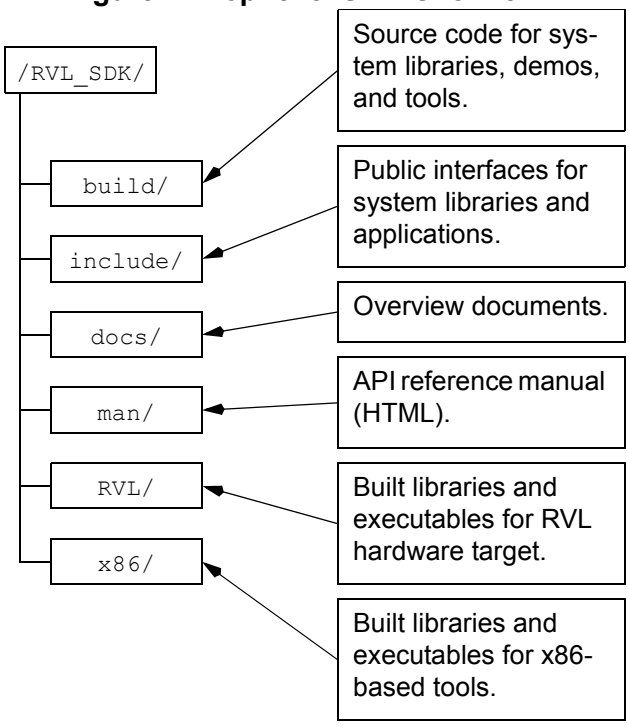
```
$(LIBNAME)$(LIBSUFFIX)
```

The executable name can be changed by modifying `BINNAMES` and the corresponding dependency line at the bottom of the file.

5 Build Tree Overview

The figure below illustrates the high-level structure of the build tree:

Figure 1 - Top-level SDK Overview



5.1 Interfaces

To access all system libraries, include the following header file:

`/RVL_SDK/include/revolution.h`

This header file in turn includes module-specific interfaces. If you need to access specific modules, the associated header files are located under:

`/RVL_SDK/include/revolution`

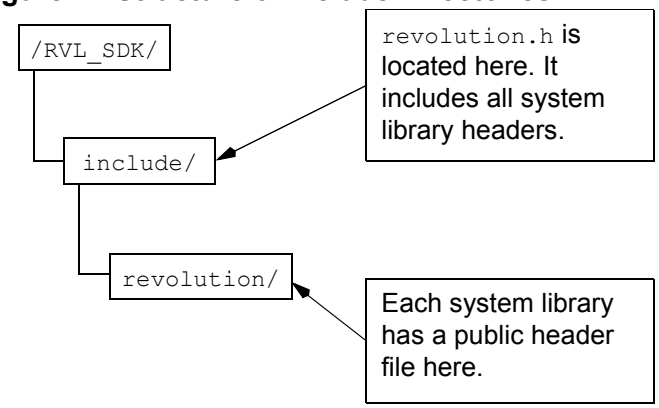
For example:

```
#include <revolution.h> // include all library headers
```

or

```
// include just OS and Graphics interfaces
#include <revolution/os.h>
#include <revolution/gx.h>
```

Figure 2 - Structure of Include Directories



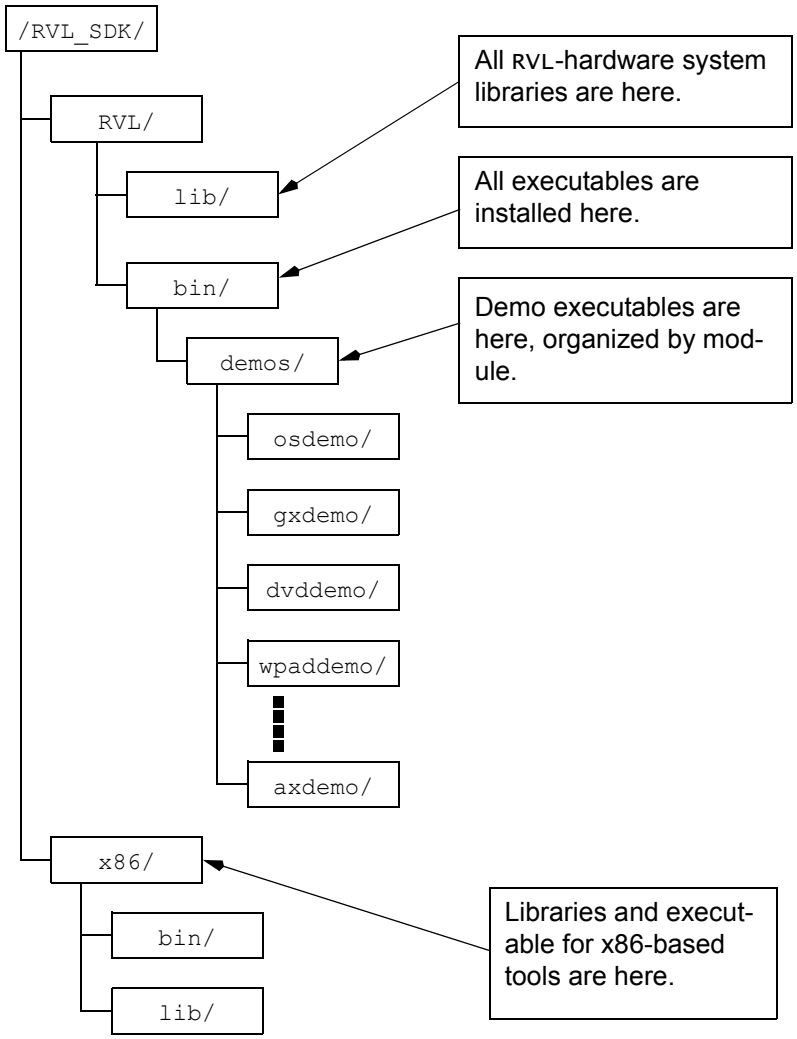
Note that the directory `/RVL_SDK/include/dolphin/` directory is not shown. This directory contains module header files that simply point to the corresponding file under `revolution/`.

These header files are meant to facilitate migration of Nintendo GameCube applications to Revolution.

5.2 Build Products

Libraries and executables are organized by hardware platform, relative to the top-level directory of the SDK.

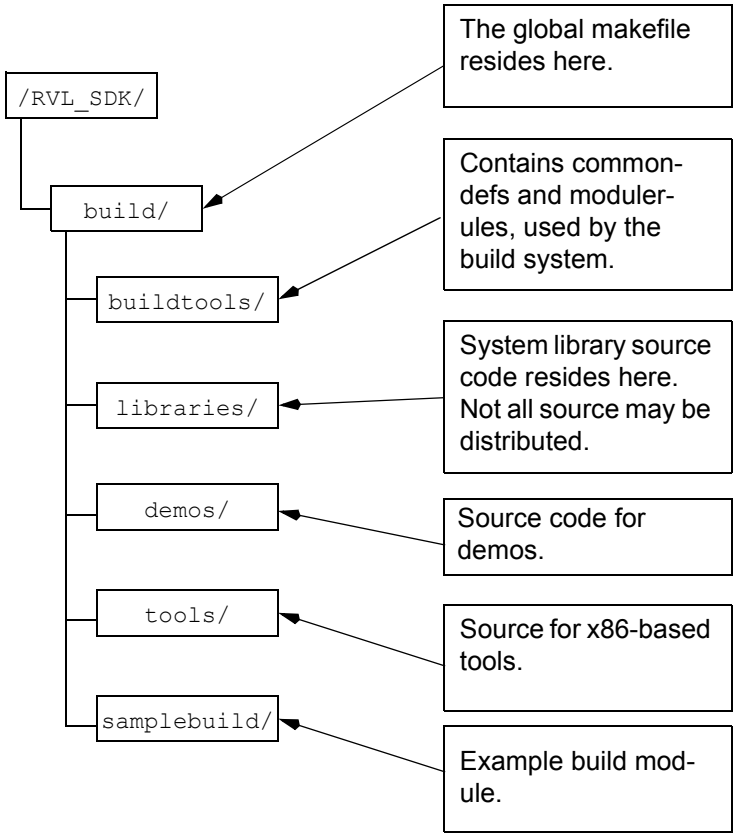
Figure 3 - Libraries and Executables



5.3 Source Code

Source code for all libraries, demos, and tools are located under the `/RVL_SDK/build/` directory. Note that not all system libraries are accompanied by source code.

Figure 4 - Source Tree



5.4 Demo Modules

Demo modules are organized by programming guide (see the `/RVL_SDK/docs`) and may include one or more applications. Local build products are stored in `lib/`, `bin/`, and `obj/` directories, organized by hardware platform.

Figure 5 - Demo Module Structure