

The Juggler Project

Team Guide

The Juggler Project: Team Guide

Published \$Date: 2002/02/20 23:36:09 \$

Table of Contents

I. Introduction	1
1. Welcome	3
Required Reading	3
Introduction to the Juggler Development Team	3
2. Background Information	4
Required Reading	4
VR Juggler Virtual Platform: Mission Statement	4
II. Compiling and Installing	6
3. Build Systems	8
Required Tools and Utilities for Building VR Juggler from Source	8
4. Get the Source	10
Introduction	10
Getting the Source Code from the CVS Repository	10
Getting a Snapshot of the Source Code	11
5. VR Juggler Build System	12
Introduction	12
Sections	12
Compiling VR Juggler Libraries: Quick Start	12
Step by step build of the Juggler distribution	12
Win32 Compiling and Development Issues	13
Necessary Tools	14
Environment Variables	14
Compiling VR Juggler	15
Known Problems	15
6. Using the VR Juggler Configure Script	17
General Quick Start	17
Autoconf and Autoheader	17
Usage	18
Environment Variables (UNIX vs. Win32)	18
Quick Start	18
Options	19
Regenerating Files	22
When to Run configure Again	22
Advanced Use	22
Debugging Configuration Errors	22
Template (.in) Files	23
7. Compiling VR Juggler	24
General Quick Start	24
Targets	25
Useful Variables	27
GNU Make Options	28
Process of Building (Individual Steps)	28
Developer Installation	29
8. Installing VR Juggler	30
General Quick Start	30
Targets	30
Useful Variables	32
Process of Full Install (Individual Steps)	32
Custom Scripts	33
mtree(1) and mtree.pl	33
install-dir.pl	34
install-src.pl	34
makefiles-gen.pl	34

InstallOps Perl Module	35
9. Maintaining and Extending the Build System	37
III. User Community	38
10. Contributing to VR Juggler	40
Reasons to Contribute	40
Why is VR Juggler cool, why do I care?	40
Why Should I, an Intelligent, Helpful Person, Contribute to the Development of VR Juggler?	40
I Want to Fix a Bug	40
I Want to Add New Features	40
I Want to Add Missing Features	40
I Think the Current Developers are Overworked	41
Yes, I Want to Contribute! Where Do I Go From Here?	41
Rules for Contributions	41
Submitting Patches to Fix Bugs	41
Submitting New Features, Implementing Missing Features	42
Creating and Submitting Patches	42
Recommended Reading	42
Making a Patch	42
Submitting Patches	44
How to Become a VR Juggler Committer	45
Why create documentation?	46
IV. Tutorials	47
11. Adding	49
V. Appendices	50
A. Coding Standard	52
Index	53

List of Tables

3.1. Requirements by Platform 8

Part I. Introduction

Table of Contents

- 1. Welcome 3
 - Required Reading 3
 - Introduction to the Juggler Development Team 3
- 2. Background Information 4
 - Required Reading 4
 - VR Juggler Virtual Platform: Mission Statement 4

Chapter 1. Welcome

Do you want to contribute to a cool project? Want to learn more about VR, cross platform techniques, and system abstractions? This is your guide. This book is for those people who wish to go further with their VR Juggler work by contributing to the project and even becoming full-fledged developers with commit access.

Required Reading

By now you should have read the *Getting Started Guide* so that you know about environment variables and about compiling VR Juggler applications. It is expected that you already know about compilers and linkers and shells and most everything else developers of UNIX and Windows software deal with on a regular basis.

Introduction to the Juggler Development Team

To become a developer with commit access on the Juggler opensource project, refer to the section called “How to Become a VR Juggler Committer”.

As a member of the Juggler Development Team, you will work on extremely scalable and extensible VR system software, use modern software engineering techniques such as UML and Design Patterns, and gain exposure within the VR development community. It is your contributions that will run on some of the worlds most powerful VR configurations, as well as smaller single-user PC workstations.

We don't require much, but in order to happily coexist, we ask that you please refer the section called “Required Reading” which lists required background information. We appreciate your effort on the team.

Sincerely, The Juggler Development Team

Chapter 2. Background Information

Required Reading

- Our coding standard [<http://www.vrjuggler.org/html/juggler.team.guide/CppCodingStandard.html>].
- *Design Patterns* (Book [<http://www.amazon.com/exec/obidos/ASIN/0201633612/o/qid=947544224/sr=2-1/002-9163340-6788217>]; ISBN 0-201-63361-2) and Design Patterns in general [<http://hillside.net/patterns/patterns.htm>]
- Autoconf [<http://www.gnu.org/software/autoconf/autoconf.html>] documentation. (Try doing **info autoconf**).
- Automake [<http://www.gnu.org/software/automake/automake.html>] documentation. (Try doing **info automake**).
- GNU make [<http://www.gnu.org/software/make/make.html>] documentation. (Try doing a **man** or **info** on `make` or `gmake`. There is a good book on using **make** by O'Reilly and Associates [<http://www.ora.com/>]).
- Perl [<http://www.perl.com/>] documentation. (Try doing a **man** on `perl`. There are several good books on Perl by O'Reilly and Associates [<http://www.ora.com/>]).
- Doxygen [<http://www.doxygen.org/>] documentation (used in VR Juggler to autogenerate documentation for C++ classes).
- CVS [<http://www.cvshome.org/>] documentation (used for revision control).

VR Juggler Virtual Platform: Mission Statement

VR technology as a whole is stagnating with its closed source solutions. This means that anyone new to VR need to completely develop a new solution, often inferior than others. Releasing an open VR system is vital to the development of VR. Instead of reinplementing VR systems, people need to focus on implementing new VR ideas and methods so that VR technology can mature. Now, because of VR Juggler, people can focus on implementing these new VR ideas and methods for the future because now this standard technology is readily available to all, free from cost or legal ties.

Juggler is a platform for virtual reality applications. The name comes from the delicate balance that must be struck between flexibility, performance, and ease-of-use. You develop your application once for Juggler, then anywhere that Juggler is supported, your application is also supported.

VR juggler's design addresses several key requirements for a virtual reality development system.

- Performance
- Flexibility
- Rapid application development
- Ease of use
- Extensibility
- Portability

Flexibility is achieved by placing common abstractions over I/O devices. New devices can be added easily, and existing devices can be reconfigured or replaced, even while an application is running.

To allow optimal performance, applications are given direct access to graphics APIs (currently including OpenGL and Iris Performer).

VR Juggler includes built-in support for performance monitoring of applications and graphics subsystems. It supports multiple-processor machines and will support distributing applications across multiple machines.

Using VR Juggler and developing applications are kept as simple as possible. Small base classes provide a skeleton for application development, while the abstractions of I/O devices simplify programming. During run-time, any VR Juggler application can be controlled or reconfigured by a Java-based graphical interface.

VR Juggler is designed to support a wide array of VR hardware on a variety of architectures. Several tracking systems, gloves, and input devices are already supported. Juggler supports projection-based displays such as the CAVE or C2, and includes support for head-mounted devices. Development work is being done on *IRIX*, *Linux*, *Windows NT*, *Solaris*, and *FreeBSD* systems, with reports of success on *HP/UX*.

Part II. Compiling and Installing

Table of Contents

3. Build Systems	8
Required Tools and Utilities for Building VR Juggler from Source	8
4. Get the Source	10
Introduction	10
Getting the Source Code from the CVS Repository	10
Getting a Snapshot of the Source Code	11
5. VR Juggler Build System	12
Introduction	12
Sections	12
Compiling VR Juggler Libraries: Quick Start	12
Step by step build of the Juggler distribution	12
Win32 Compiling and Development Issues	13
Necessary Tools	14
Environment Variables	14
Compiling VR Juggler	15
Known Problems	15
6. Using the VR Juggler Configure Script	17
General Quick Start	17
Autoconf and Autoheader	17
Usage	18
Environment Variables (UNIX vs. Win32)	18
Quick Start	18
Options	19
Regenerating Files	22
When to Run configure Again	22
Advanced Use	22
Debugging Configuration Errors	22
Template (.in) Files	23
7. Compiling VR Juggler	24
General Quick Start	24
Targets	25
Useful Variables	27
GNU Make Options	28
Process of Building (Individual Steps)	28
Developer Installation	29
8. Installing VR Juggler	30
General Quick Start	30
Targets	30
Useful Variables	32
Process of Full Install (Individual Steps)	32
Custom Scripts	33
mtree(1) and mtree.pl	33
install-dir.pl	34
install-src.pl	34
makefiles-gen.pl	34
InstallOps Perl Module	35
9. Maintaining and Extending the Build System	37

Chapter 3. Build Systems

There are a few different ways to build the projects related to the VR Juggler project. This chapter explains all the ins and outs of compiling VR Juggler and related projects including the documentation you are reading right now.

Required Tools and Utilities for Building VR Juggler from Source

This section lists the comprehensive set of software tools that we require when compiling VR Juggler from its source. If you are reading a printed version of this file, please check the website to ensure that you have the latest information. There have been several instances where bugs in third party software caused problems with compiling VR Juggler and running VR Juggler applications.

Table 3.1. Requirements by Platform

Platform	Minimum Software Versions	
All	CVS	1.11
	GNU make	3.78
	GNU Autoconf	2.13
	GNU Automake	1.4
	Perl	5.004
	Java 2 Platform (only for VjControl)	1.3.0
IRIX	Operating System	6.5
	MIPSpro Compilers	7.3.1.1m
RedHat Linux	Operating System	7.2
	GCC Compilers	2.96-98
	Mesa 3D	3.2
Win32	Operating System	Windows NT 4.0a
	Microsoft Visual Studio	6.0 (SP4 or SP5)
	Juggler Tools	Latest
	Cygwin	1.3 (including Perl 5.6.1)
Platforms not Officially Supported		
HP-UX	Operating System	11.00
	aCC	A.03.30
	OpenGL	1.1
Solaris™b	Operating System	7
	Patch Set	June 7, 2001
	GCC	2.95.2, 3.0.2
	GNU Binutils	2.12
	Mesa 3D	3.2

Build Systems

Solaris™	Operating System	7
	Patch Set	June 7, 2001
	GCC	2.95.2, 3.0.2
	GNU Binutils	2.12
	Mesa 3D	3.2
FreeBSD	Operating System	4.5
	GCC	2.95.2
	Mesa 3D	3.2

aWindows NT 4.0 is needed only for compiling the VR Juggler libraries. Applications can be compiled against the libraries on Windows 95/98/Me.

bSupport for the Forte™ C++ (WorkShop™ C++) development package on Solaris™ has not been tested, so only freely available GNU utilities are listed.

Chapter 4. Get the Source

Introduction

There are two ways to get the VR Juggler source code:

1. Get the source directly from the CVS repository.
2. Download a snapshot of the source code as a TAR file from the “Download” page.

Both methods are described in this section, though the second is mostly self-explanatory.

Getting the Source Code from the CVS Repository

Before you can use CVS to get the source code, you must be sure that CVS itself is set up properly on your system. Depending on your platform, CVS may be available in several different forms. The most common version is the original command-line interface. If CVS is not available on your system, you can get the source from CVShome.org [<http://www.cvshome.org/>]. This version is available for all platforms including Win32. For graphical interfaces, you can choose from the following:

- MacCVS/WinCVS: A freely available front-end for CVS available from WinCvs.org [<http://www.wincvs.org/>]
- jCVS: A CVS client written in Java available from jCVS.org [<http://www.jcvs.org/>]
- tkCVS: A Tcl/Tk-based interface to CVS available from Two BarleyCorns [<http://www.twobarleycorns.net/tkcv.html>]

Once you have CVS running in some form, you are ready to download the source from the VR Juggler repository, hosted at SourceForge [<http://www.sourceforge.net/>]. The instructions given here are from the CVS section [http://sourceforge.net/cvs/?group_id=8041] VR Juggler project page and are subject to change if the SourceForge policies change. Note that there are two sets of directions on that page: one for anonymous access and one for developers with commit access. Each of those is discussed in the following subsections.

Anonymous CVS Access

Anonymous CVS access is for those who want the latest source code but do not have commit access. This would include developers who are working on patches they wish to submit for addition to the source base. As of this writing, the following are the basic instructions for anonymous CVS access:

1. “Log in” to the SourceForge CVS server. This step only has to be done the first time you access the repository. Afterwards, the login setting is saved in your \$HOME/.cvspass file.

```
% cvs -d:pserver:anonymous@cvs.vrjuggler.sourceforge.net:/cvsroot/vrjuggler login
```

2. Check out a copy of the current VR Juggler source tree: If what you really want is the VR Juggler 1.0 release branch instead of the developmental main branch, use the following command:

```
% cvs -d:pserver:anonymous@cvs.vrjuggler.sourceforge.net:/cvsroot/vrjuggler checkout ju
```

(The string `RELENG_1_0` is the name of the branch in the CVS repository containing the VR Juggler 1.0 release code. “RELENG” means “release engineering”.)

```
% cvs -d:pserver:anonymous@cvs.vrjuggler.sourceforge.net:/cvsroot/vrjuggler checkout -r
```

Once you have the source checked out, you can use CVS as you normally would keeping in mind that you have read-only access to the repository when using anonymous access.

You may find it useful to retain the `CVSROOT` value rather than always giving the `-d` option on the command line. Setting the `$CVSROOT` environment variable does just that. Defining an alias such as `cvsjuggler` that sets the environment variable is the recommended method.

Developers with Commit Access

For those developers with commit access, the steps are slightly different. You must have some version of SSH installed to access the repository as a developer. This is required by SourceForge, and we (the VR Juggler Team) have no control over this. The steps then to get a read/write copy of the VR Juggler source tree are as follows:

1. Set the environment variable `$CVS_RSH` to use `ssh(1)`.

```
% CVS_RSH = ssh
```

(Note that this is using the shell-independent environment variable syntax defined in the *Getting Started Guide*.) It is recommended that you set this environment variable in your shell's configuration appropriate configuration file or as part of an alias that configures your environment to work with the VR Juggler CVS repository.

2. Check out the source using your SourceForge login name.

```
% cvs -dlogin_name@cvs.vrjuggler.sourceforge.net:/cvsroot/vrjuggler checkout juggler
```

Be sure to use the appropriate branch name.

Note that there is no login step in this case. Your SourceForge password is required for all CVS commands.

After this point, you can access the CVS repository as though it is a normal, local repository. As with the anonymous CVS access, you can set the `$CVSROOT` environment variable to the value given with the `-d` option above. For further information about using SSH with SourceForge, please refer to the appropriate SSH documentation and to SourceForge's documentation [http://sourceforge.net/docman/?group_id=1].

Getting a Snapshot of the Source Code

As a developer, you will probably need access to the latest VR Juggler source and therefore should be getting it from the CVS repository. In some cases, however, it may be helpful to get a snapshot of the code in an archived format. The download page [<http://www.vrjuggler.org/html/download/index.html>] on the VR Juggler website has a section listing source code snapshots [<http://www.vrjuggler.org/html/download/index.html#src>]. Here, you can get a compressed tar file containing the official source code for a particular release of VR Juggler. These snapshots are not for developmental purposes and will not give you access to the CVS repository.

Chapter 5. VR Juggler Build System

Introduction

Of particular interest, the VR Juggler project uses the **autoconf** and **automake** utilities to stay portable across platforms. This means you will have to “configure” the project before you make it. To make sense of this documentation, please understand the concepts listed in the section called “Required Reading”. Posting about these subjects to the mailing list will probably be ignored.

Sections

- Quick Start: A quick overview of how to build Juggler.
- Win32-Specific Considerations: Getting set up for Juggler use and development in a Win32 environment.
- Using the Configure Script: How to use **configure** to set up a build environment before compiling.
- Compiling VR Juggler: How to use the Juggler build system in detail.
- Installing VR Juggler: How to install Juggler after compiling it.

Compiling VR Juggler Libraries: Quick Start

Step by step build of the Juggler distribution

1. Choose a directory to put Juggler. From this point on, we will refer to that directory as SRC_DIR.
2. Go to that directory and checkout juggler using CVS. Make sure CVS is set up and pointing to the VR Juggler repository. Refer to the SourceForge instructions [http://sourceforge.net/cvs/?group_id=8041] for getting VR Juggler with CVS.

```
% cd SRC_DIR
% cvs checkout juggler
```

3. Enter the newly checked-out VR Juggler source directory.

```
% cd SRC_DIR/juggler
```

4. Set the environment variable VJ_BASE_DIR

Note

- a. The directory `instlinks` is semi-important. What does the word “instlinks” mean? It is a working distribution of the Juggler library. After `gmake` is done, you’ll be able to use this directory as though it were an installed version of VR Juggler.
- b. This directory also determines where juggler files will be created. Juggler files will be created in

some-directory/instlinks/. . (in other words, files are created one directory back from instlinks).

Make sure the following environment variable assignment line is in your shell's startup file:

```
VJ_BASE_DIR=SRC_DIR/juggler/instlinks
```

This is required by Juggler's build system (makefiles, etc.).

5. Compile juggler (using the Autoconf system and GNU make):

a. Auto-generate the configuration script **configure**:

```
% autoconf
```

b. Auto-generate the `vjDefine.h.in` header template:

```
% autoheader
```

c. Configure the build. This probes the system for capabilities, generate `Makefiles`, etc. It “configures” the source tree for the current system. The **configure** script will let you change how VR Juggler compiles. For example, you could build VR Juggler without the Performer API using an option passed to the **configure** script.

Note

You can type **configure --help** for options.

```
% ./configure
```

d. Build (make) the distribution using GNU make. This compiles, links, and creates the final distribution tree (in the form of symbolic links).

```
% gmake
```

6. Advanced Topics

a. Send compiler output to a different directory than the source tree. Run this in the directory you want compiler output to go. This is typically done with a subdirectory in the juggler source tree. We will call it `<BUILD_DIR>`.

```
% mkdir <BUILD_DIR>
% cd <BUILD_DIR>
% <SRC_DIR>/configure
```

Win32 Compiling and Development Issues

When VR Juggler was first ported to the Win32 environment, a Microsoft Visual C++ project file was used for

compiling the library. Development on the GNU Autoconf-based build system on UNIX platforms had been going strong for a long time prior to this, and it was quickly realized that having two separate methods for compiling would complicate matters. The build system used for compiling VR Juggler is now the same on UNIX and Win32 platforms. Only the Autoconf-based system is supported which provides the same build environment on all platform forms.

Since the UNIX and Win32 environments have many striking differences, special information must be provided for using the Juggler build system on Windows-based platforms. This document provides all the information necessary for getting set up in a Win32 environment prior to compiling and installing Juggler for developmental use. This is intended primarily for developers who wish to work with the Juggler source and (hopefully) submit work to the Juggler team. Users who are doing application development on a Win32 platform may want to read it too so that they may get a better understanding of what is available as part of the Win32 support.

Necessary Tools

Developers should refer to the Win32 section of the download page [<http://www.vrjuggler.org/html/download/>] for the list of packages that must be installed. The `juggler-tools` package is especially important. It contains exactly the utilities used for compiling Juggler (aside from Visual C++ of course) including some modified versions of programs. Of note, the version of **make** included is what comes with the Bamboo Win32 tools. It is a modified version of GNU make that will run in a DOS shell and can handle DOS-style paths. Also, **makedepend** has been modified to output UNIX-safe paths and to know about the `INCLUDE` environment variable used by Visual C++.

In addition to what is listed on the download page, the following packages are needed for Win32 development of Juggler (but not for application development):

- Cygwin [<http://sources.redhat.com/cygwin/download.html>] 1.1.8 or newer. With the newer versions of Cygwin, the tools work better in the Win32 environment, and many pre-compiled packages are available including CVS, Perl, Autoconf, and Automake. It is strongly recommended that the Cygwin versions of all these utilities be used.
- GNU Autoconf [<ftp://ftp.gnu.org/gnu/autoconf/>] version 2.13 and GNU Automake [<ftp://ftp.gnu.org/gnu/automake/>] version 1.4. Versions beyond these are not currently supported on any platform form. This is needed to compile a new configure script after modifying `configure.in` and to update `vjDefines.h.in` when necessary. Note that these tools can only be run from a BASH shell (included with the Cygwin package). Autoconf and Automake can be compiled and installed on a Win32 platform without modification using the Cygwin tools.

Environment Variables

There are several environment variables that need to be set properly when compiling and installing Juggler and when running Juggler applications. These variables apply to all users, both those who are compiling and running applications as well as those who are compiling VR Juggler itself.

`%PATH%` All users must have the `juggler-tools` directory in their path. If the Cygwin utilities are installed, it is especially important that `juggler-tools` be listed *before* the Cygwin bin directory so that the supported utilities are found first. Unfortunately, when running a BASH shell, the Cygwin path is always first, but in the BASH environment, this is acceptable. The ordering is far more important for DOS shells. Besides `juggler-tools`, the directory containing the Visual C++ executables must also be in the path. The configure script needs this so that it can find **CL.EXE** when it is setting up the C and C++ compilers.

`%INCLUDE%` This should also be set for all users. It provides paths to the Visual C++ include directories that **CL.EXE** needs when compiling. On Windows NT, it is set only for the

user who installed Visual C++. Be sure to change this to be included in *all* users' environments. Alternatively, it can be set using the **VCVARS32.BAT** batch file on a per-shell basis, but users will see much better results if it is set in the general environment. In particular, a BASH shell does not get the environment changes made by the batch file. The batch file also uses “8.3” names which may confuse the Cygwin tools.

%LIB% This is another Visual C++ environment variable that needs to be set in the same manner as **%INCLUDE%**. Again, this is initially set only for the user who installed Visual C++ and should be changed so that it is set for all users.

%VJ_BASE_DIR% The value for this should be set using the DOS path with /'s as the path separator rather than \'s as would normally be the case with DOS/Win32 paths. For example, if VR Juggler is installed in C:\software\vrjuggler, **%VJ_BASE_DIR%** would be set to C:/software/vrjuggler. Never use //c/software/vrjuggler or /cygdrive/c/software/vrjuggler. The Visual C++ I/O libraries will not be able to handle these special Cygwin paths.

For more information, refer to the section titled “Environment Variables” in the *Getting Started Guide*.

Compiling VR Juggler

Finally, the actual process of compiling (and installing) VR Juggler can begin. After taking the steps in the preceding sections, this will be an easy task (assuming that the source is stable of course). This section focuses primarily on the process when done within a DOS shell. In a BASH shell, the environment is almost identical to that on a UNIX platform, so there are no special considerations to be made.

The first step, as usual, is to configure the build environment with the configure script. (This assumes that **configure** and **vjDefines.h.in** have already been generated.) If running in a DOS shell, **configure** must be invoked using the **sh** utility. The following will *not* work:

```
> ./configure [options]
```

Instead, it must be run as:

```
> sh ./configure [options]
```

As with the UNIX environment, it can be run from a separate build directory. Refer to Chapter 6, *Using the VR Juggler Configure Script* for more information (specifically, the section called “Usage”). Options with paths should use /'s for the path separator rather than \'s. They should also be enclosed in double quotes (“path/to/something”) in case the path includes one or more spaces.

Once configuration has completed, compiling can be done in either a DOS shell or a BASH shell using the version of GNU make in **juggler-tools**. *The Cygwin B20.1 version will not work in a DOS shell.* This is when it is especially important to have the **juggler-tools** directory before the Cygwin directory in the user's path.

Beyond this, everything (aside from the problems listed in the next section) will be the same as in a UNIX environment.

Known Problems

Listed here are things to keep in mind when working in a Win32 environment. It is not as stable as its UNIX counterpart due, for the most part, to reasons beyond the control of this project. Please read this in full before reporting problems with the Win32 build.

- In versions of Cygwin prior to 1.1, the UNIX utilities will appear to become blocked or may die with an error saying that they cannot fork. This may have been due to a race condition in the old Cygwin code. Typically, it happens when one or more BASH shells are open while a build is happening in a DOS shell. Closing one or more (sometimes all) of the BASH shells usually fixes the problem. It is strongly recommended that developers update to Cygwin 1.1.8 [<http://sources.redhat.com/cygwin/download.html>] and to the latest `juggler-tools` package [<http://www.vrjuggler.org/pub/juggler-tools-1.1.zip>].
- The “developer installation” does not work in any Win32 file system. It relies heavily on symbolic links which, while common on UNIX file systems, have no meaning in a Win32 environment. Because of this, VR Juggler must be fully installed after it has been compiled (refer to Chapter 8, *Installing VR Juggler* for more information on this process). Once that is done, `$VJ_BASE_DIR` can be set to point to the installation directory and development can continue in the build tree.
- The configure script assumes that the compiler knows how to find the OpenGL libraries and headers. In VR Juggler 1.0, no check is made to verify that the OpenGL libraries are available. This has been corrected in VR Juggler 1.1 and beyond.
- Dependencies must be generated differently than on UNIX-based platforms. The Visual C++ compiler does not output the dependencies of a given source file in the same manner as all UNIX C and C++ compilers supported by the Juggler build system. Because of this, **makedepend** is used. It may not handle all the `#ifdef`'s properly and report non-fatal warnings. These can be ignored typically, especially if it is trying to find some file that is clearly UNIX-specific (e.g., `pthread.h`). The version of **makedepend** in `juggler-tools` has been modified to output UNIX-style (and UNIX-safe) paths, but it is possible that not all situations have been considered. If it is clear that **makedepend** is not handling a path properly, please report the problem [<http://www.vrjuggler.org/html/bugs/bugs.html>].
- A configure script generated in a Win32 environment will not work in a UNIX environment. The “^M problem” causes the shell to fail when trying to run the script. A configure script generated in a UNIX environment will however work in a Win32 environment because **sh** can handle both UNIX- and DOS-style newlines.
- The application makefiles (*not* the library makefiles) all use the `$(VAR_NAME)` syntax for referencing variables. This is so that the makefiles will work with **nmake**. These are intended to be usable with all version of **make** (or as many as is feasibly possible), so they deviate from the style used in other makefiles to accommodate this particular version of **make**.

Chapter 6. Using the VR Juggler Configure Script

The VR Juggler project uses the `autoconf` utility to stay portable across platforms. This means you will have to “configure” the project before compiling it. To make sense of this documentation, please understand the concepts listed in the section called “Required Reading”. Posting about these subjects to the mailing list will probably be ignored.

In this chapter, the following conventions are used for text formatting and naming:

- Program, file and directory names are represented `this way`.
- Commands that are intended to be run by the user are written in **this style**.
- References to system and library calls are written as `call_name(##)` where “##” is the manual section where that call’s documentation can be found.
- Makefile targets are named as `'target'`, and makefile variables are named as `$(VAR_NAME)`.
- Environment variables are named as defined in the “Environment Variables” chapter in the *Getting Started Guide*.

General Quick Start

When first downloading the Juggler source to compile it, users must generate the configure script and `vjDefines.h.in` template using **autoconf** and **autoheader**. These files are not part of the repository because they are generated using third-party tools from files in the repository (i.e., `configure.in` and `acconfig.h`). The usual way to generate the files is:

```
% autoconf ; autoheader
```

This must be executed in the top-level Juggler source directory.

Once the `configure` script is generated, it can be invoked by typing:

```
% ./configure
```

To get a list of all the available options, do:

```
% ./configure --help
```

Autoconf and Autoheader

VR Juggler uses tools from the GNU Autoconf package for automatically generating its configure script and its template configuration header file (`vjDefines.h`). The command **autoconf** is used to “compile” the file `con#figure.in` in the top-level directory into the configure script, aptly named `configure`. **autoheader** reads *both* `configure.in` and `acconfig.h` to generate the template file `vjDefines.h.in` which the configure script later uses to generate `vjDefines.h`.

It is important that the generated files remain synchronized with their templates for proper compiling of the library.

To facilitate this, revision numbers of `configure.in` and `acconfig.h` are available at the tops of the files, and these can be compared against the revision number of `configure` and `vjDefines.h.in`. If the numbers do not match, it is best to run **autoconf** and/or **autoheader** to get everything up to date. Typically, **autoconf** needs to be used far more frequently than **autoheader**, but there are times when changes made only to `configure.in` require that both be run. It hurts nothing to run both whenever `configure.in` is updated.

Note that it is possible to use a configure script generated on a UNIX platform in a Win32 environment. The version of **sh.exe** distributed in `juggler-tools.zip` handles the UNIX newline characters. It is, however, not possible to use a configure script generated in a Win32 environment on a UNIX platform. The `^M` characters confuse the shell and cause the script to fail immediately.

Usage

Environment Variables (UNIX vs. Win32)

In a UNIX environment, no environment variables have to be set for **configure** to work properly. The `PATH` should be reasonable and must include tools such as C and C++ compilers, a working GNU make and standard UNIX utilities such as **cat**, **sh**, **test**, etc. The configuration process can of course use environment variables such as `$CC`, `$CXX`, `$CFLAGS`, etc. as it runs. Users can define values for these variables to force the configuration process to use user-defined settings. This is common in configure script use.

In a Win32 environment, some variables have to be defined for the Microsoft Visual C++ compiler to work properly. These are mentioned on the Download page [<http://www.vrjuggler.org/html/download/>]. Refer to that document to ensure that the environment is set up right for building and installing.

Quick Start

The easiest way to configure Juggler is to run **configure** with no options in the top-level source directory. Doing this will root compiling in this directory. This will use all the default values for the host platform with the script doing its best to make the right choices. In some cases, however, this may not be sufficient.

Possible situations that may cause **configure** to report a fatal error or a warning about a feature being disabled include the following:

- The version of **sh** available does not support the features being used in **configure**. In this case, the configure script will exit with an error status. To get around this, invoke **ksh**, **bash**, or **zsh** on **configure** (in that order as available). These shells have newer features and fully support **sh** syntax. (In particular, this is known to be a problem with `/bin/sh` on Solaris 7 and possibly Solaris 8.)
- A “modern” C++ compiler was not found. *This is a fatal error.* VR Juggler uses many features of the latest C++ standard, and a C++ compiler supporting these features is absolutely required. To use an alternate C++ compiler, specify a value for the environment variable `CXX`.
- The OpenGL libraries and/or header files were not found. *This is a fatal error.* To remedy this problem, use the `--with-oglroot` option to **configure** to specify the root directory containing the OpenGL installation.
- The OpenGL Performer libraries and/or header files were not found. This is not a fatal error and simply causes the Performer API to be disabled for that build. If Performer is available, use the `--with-pfroot` option to **configure** to specify the root directory containing the Performer installation. It may also be necessary to use the `--with-pfver` option if OpenGL Performer 2.4 is not the version installed.
- A version of Perl greater than or equal to 5.004 was not found. *This will cause compiling of **vjcontrol** and installation of the library to fail.* To provide a valid Perl binary for use by the makefiles, use the `--with-perl` option to **configure**. The path given should be the full path up to but *not* including the **perl** binary itself. For example, if a **perl5** binary were in `/usr/unsupported/bin`, give this directory as the argument to -

`--with-perl`. In a Win32 environment, use `/`'s as the path separator rather than `\`'s.

- No Java compiler was found. This disables the compilation of **vjcontrol**. There are three ways to fix this with all giving the same results:
 1. Set the environment variable `JDK_HOME` to the Java installation directory (e.g., `/usr/local/java` which contains a `bin` subdirectory with all the JDK utilities).
 2. Set the environment variable `JAVA_HOME` to the Java installation directory. This serves the same purpose as the previous option but offers users a different environment variable name to allow for varying configurations between sites and platforms.
 3. Give **configure** a path to the Java installation through the `--with-jdkhome` option.

Note

In a Win32 environment, the path separator character should be a `/` rather than a `\`. This is consistent with other uses of paths in a Win32 environment.

Options

The following is a list of the current Juggler-specific options that can be invoked at configuration time as of Revision 1.273.2.3 of `configure.in` (the revision number can be found at the top of the `configure.in` file and at the top of the configure script). The output for all options is generated by doing **configure --help**.

- | | |
|-----------------------------|---|
| <code>--with-cc</code> | Force the use of a C compiler other than what would be detected automatically. |
| <code>--with-cxx</code> | Force the use of a C++ compiler other than what would be detected automatically. |
| <code>--with-gcc</code> | Force the use of GCC as the compiler. Currently as of Revision 1.273.2.3 of <code>configure.in</code> , use of this option is not fully implemented. Its current state is for use only in a Win32 environment so that the Cygnus-Win32 version of GCC can be used to compile Juggler instead of the Microsoft Visual C++ command-line compiler. |
| <code>--with-abi=OPT</code> | Define the application binary interface (ABI) for which Juggler is compiled. In some cases, the instruction set architecture (ISA) can also be defined in combination with the ABI. The possible values for this are: <ul style="list-style-type: none">• <code>N32_M3 (IRIX)</code>: Use the new 32-bit ABI and the <code>mips3</code> ISA (using MIPSpro compiler options <code>-n32 -mips3</code>. <i>This is the default for building on IRIX.</i>• <code>N32_M4 (IRIX)</code>: Use the new 32-bit ABI and the <code>mips4</code> ISA (using MIPSpro compiler options <code>-n32 -mips4</code>).• <code>64_M3 (IRIX)</code>: Use the 64-bit ABI and the <code>mips3</code> ISA (using MIPSpro compiler options <code>-64 -mips4</code>).• <code>64_M4 (IRIX)</code>: Use the 64-bit ABI and the <code>mips4</code> ISA (using MIPSpro compiler options <code>-n32 -mips4</code>).• <code>ELF_i386 (Linux, Solaris, FreeBSD, NetBSD)</code>: Use the ELF ABI on x86 hardware. This uses the default ABI and ISA when compiling on the i386 versions of the named operating systems. Other ISA's may work but have not been tested. For |

other architectures, let the configure script figure out the ISA based on what it determines the target platform to be. *This is the default option for the named operating systems.*

- WIN32_i386 (*Windows NT, Windows 9x*): Use the standard Win32 ABI on x86 hardware. This uses the default settings for compiling in these environments. No support is planned for other architectures. *This is the default option for the named operating systems.*
- HP (*HP-UX*): Use the HP PA-RISC ABI. This will be supported in the future when the HP-UX port is functional. *This is the default for building on HP-UX.*
- ALPHA (*Digital UNIX*): Use the standard Digital UNIX COFF ABI. This will be supported in the future if a port to Digital UNIX is made.

- with-threads=OPT Define the threading implementation. The options are IRIX_SPROC (use IRIX `sproc(2)` threads), POSIX (use pthreads) and WIN32 (use Win32 threads). The default is IRIX_SPROC, though configure knows what other platforms should use. For example, POSIX threads are used on Linux, Solaris and FreeBSD without specifying this option.
- with-jdkhome=PATH Give the JDK installation directory. This is the base directory containing a `bin` subdirectory with all the JDK utilities (such as `javac` and `jar`). The default is the value in `$JDK_HOME` or `/usr/java` if `$JDK_HOME` has no value. In a Win32 environment, this option must be given using `/`'s as the path separator rather than `\`'s.
- with-pfroot=PATH Give the OpenGL Performer installation directory. This is the base directory containing an `include` subdirectory that in turn has a `Performer` subdirectory and at least a `lib` directory with the Performer libraries. On IRIX, there may also be a `lib32` subdirectory and a `lib64` subdirectory. They will be used appropriately depending on the ABI chosen (see above). The default path is `/usr`.
- with-pfver=VER Give the IRIS/OpenGL Performer version installed. Between Performer 2.2 and 2.4, the names of the libraries were changed. In 2.2, they had an `_ogl` extension, but in 2.4, that extension was dropped. Specifying the version number defines which libraries will be linked in the Performer sample applications. The default version is 2.4.
- with-oglroot=PATH Give the OpenGL installation directory. This is the base directory containing an `include` subdirectory that in turn has a `GL` subdirectory and at least a `lib` directory with the OpenGL libraries. On IRIX, there may also be a `lib32` subdirectory and a `lib64` subdirectory. They will be used appropriately depending on the ABI chosen (see above). This option is particularly useful (i.e., necessary) on platforms such as Linux, FreeBSD and HP-UX where OpenGL is not distributed as part of the base operating system. Common values are `/usr/local`, `/usr/X11R6` and `/opt/graphics/OpenGL` (HP-UX 10.20). In VR Juggler 1.0, this option is not supported in a Win32 environment. The default path is `/usr`.
- with-mesa Tell the VR Juggler build system that the target system is using the Mesa OpenGL implementation. This will look for `libMesaGL` and friends rather than `libGL`, and it will use the `libMesa*` libraries when linking sample applications.
- -
with-audioworks=PATH Give the AudioWorks installation directory. This is the base directory containing an `include/PSI` subdirectory and at least a `lib/PSI` subdirectory with the AudioWorks libraries. If AudioWorks is found, the Juggler AudioWorks sound wrapper will be compiled and installed. The default path is `/usr`.
- with-sl=PATH Give the SL installation directory. This is the base directory containing an `include/`
-

sl subdirectory and at least a lib subdirectory with the SL libraries. Before testing for SL, a test will be performed checking for the SGI Audio library if compiling on IRIX. If it is found, then the test for SL is performed. If SL is found, the VR Juggler SL sound wrapper will be compiled and installed. The default path is /usr.

--with-perl=PATH Name the directory containing a Perl 5.004 or newer binary. A working Perl 5.004 or newer binary is necessary for a complete build and install of Juggler. The version of the binary can be determined by executing (**perl -v**). This path is the full path leading up to the binary (e.g., /usr/local/bin). There is no default, but common values are hard-coded into the configure script. They are /usr/bin, /usr/local/bin and C:/Perl/bin. Note that in a Win32 environment, paths must be given using /'s as the path separator rather than \'s.

- Name the group used for ownership of installed files. This allows installation of Juggler under a group name controlled by the site administrator. If the group name is not found, no group is specified during installation which is typically fine. The default is "vrjuggler".

- with-in#

stall-group=NAME

--with-file-perms=FILE PERMISSIONS Provide permissions for installed (normal) files. This will be used for header files, source code, config files, etc. The format must be UNIX octal-style (e.g., 0644). The default is 0644.

- Provide permissions for installed executables. This will be used for scripts, executables, and shared libraries. The format must be UNIX octal-style (e.g., 0644). The default is 0755.

- with-exec-perms=EXEC PERMISSIONS

--with-dir-perms=DIR PERMISSIONS Provide permissions for directories created in the installation tree. The default is 0755.

- Set the version number for this build. If not specified, it will be determined automatically from the contents of the VERSION file.

-

with-ver#

enable-gl-api Enable the OpenGL API in the library. *This option cannot be disabled and is on by default.*

--enable-pf-api Enable the Performer API in the library. This turns on (or turns off) compiling of the Performer code in VR Juggler. Disabling is done using the standard method for Autoconf-generated scripts. On platforms that do not have Performer, a warning is issued and the API is disabled. *This option is enabled by default.*

- Enable (or disable) the performance-testing code. This defines which method of performance testing should be used or disables it altogether. The possible options are as follows:

-en#

able-performance=OPT

- SGI: Use SGI-specific methods for doing performance testing.
- POSIX: Use portable POSIX-specified code for performance testing.
- NONE: Disable performance code

Disabling this code can be done using the NONE option or by using the standard disable option for Autoconf-generated scripts.

Another commonly used option is --prefix which is used to set the base directory for the installation. In a Win32 environment, be sure to use /'s as the path separator rather than \'s to be consistent with all other paths used by **configure** and the makefiles it generates.

Regenerating Files

After running **configure**, one of the generated files is a shell script called `config.status`. This script can be used to regenerate all the output files from their respective `.in` files without going through the entire configuration process again. This is useful, for example, when only a `Makefile.in` file changes without an update in the `configure` script. In general, when any `.in` file (other than `configure.in`) changes, **config.status** can be used to quickly incorporate the template file changes into the generated files.

When to Run configure Again

The `configure` script should always be run after it is regenerated from `configure.in`. Thus, whenever **autoconf** is run (see the section called “Autoconf and Autoheader” for more information), the script itself should be run again to ensure that the latest build environment is available.

Advanced Use

The VR Juggler `configure` script has been designed to work in a multi-platform environment. It allows Juggler to be built for multiple platforms using the same source tree (distributed with NFS for example) with all versions co-existing safely.

To take advantage of this feature, make a directory somewhere for building the library. This directory can go anywhere, but the typical location is a subdirectory of the top-level Juggler source tree. (Note that on a Win32 platform, this directory **MUST** be a subdirectory of the Juggler source tree.) It can also be called whatever seems appropriate. A common name is `build`, and alternate options are `build-PLATFORM` where `PLATFORM` is the name of the platform on which VR Juggler is being built. Multiple build directories can exist side by side. They fully separate all parts of the compile process and can be removed safely at the root. After making the directory, **cd** into it and run the `configure` script from it. For example, if a directory called `build` has been created as a subdirectory of `juggler`, do the following:

```
cd build ../configure [options]
```

This will run the `configure` script as normal but will create a full, unique build environment as a subdirectory of `build`.

So the question becomes “To `build` or not to `build`?” There are advantages to running **configure** from the top-level directory, and there are disadvantages to that method. A benefit of the simpler method is that it provides a quick start to building with no in-depth knowledge of the Juggler build system necessary. It has a “familiar feel” compared to other projects that use Autoconf-generated scripts for configuration. The main disadvantage of this method is that the generated files are “next to” the template files. As a developer, this can and has caused confusion about which file to modify. For example, it is very easy to forget that a quick modification to a `Makefile` will be destroyed when **configure** is run again if that change is not also made in the corresponding `Makefile.in`. It is also easy to want to check in a generated file when only the templates should be in the CVS repository.

Similarly, there are advantages and disadvantages to creating a separate build environment. As mentioned above, building can be done for multiple operating systems using the same source tree. (This is how the NSPR API is currently built using their `configure` script.) In addition to building for multiple operating systems, it is possible to easily build multiple configurations from the same source tree. This is extremely useful for building different threading models and different ABI/ISA combinations to test changes in a single source tree. Also mentioned above, the entire build environment can be removed from the root. This is because all aspects of the build are separated from the source tree so that object files, makefiles, dependency files, etc. do not pollute the source tree. On the other hand, this is not a typical use of a `configure` script (though it is a documented method of configuration in the Autoconf documentation) and is thus not familiar to users. It can also result in complicated paths depending on where the build environment is rooted and how the path to **configure** is specified.

Debugging Configuration Errors

When **configure** runs, a file called `config.log` is constantly updated in the working directory. When something goes wrong with the configuration process, the first thing to do is refer to the bottom of this file to see what was happening at the time of the error and why that step failed. Debugging the actual script is more difficult. Interested readers should refer to Chapter 9, *Maintaining and Extending the Build System* for more information on this particular topic.

Template (.in) Files

<code>vjDefines.h.in</code>	This file is the template for <code>vjDefines.h</code> (the equivalent of <code>config.h</code>) which is generated by configure . It provides definitions for various system-specific features that are available. <code>vjDefines.h</code> is included by <code>vjConfig.h</code> and should not be included by any source file directly. This file is generated by autoheader from <code>configure.in</code> and <code>acconfig.h</code> (the section called “Autoconf and Autoheader”) and is the only <code>.in</code> file that does not use the configure substitution strings (<code>@VAR_NAME@</code>). If modifications are necessary, they should be made to <code>acconfig.h</code> .
<code>Makefile.in</code>	There are many <code>Makefile.in</code> 's in the Juggler source tree. All are used by configure to generate the actual makefiles that are used for compiling and installing Juggler. They make extensive use of the configure substitution strings. Modifications should always be made to the <code>Makefile.in</code> file. The full list of available substitutions is available near the bottom of <code>configure.in</code> . The file <code>Makefile.base.in</code> provides most of these values as makefile variables (macros) and can be included by other makefiles to maximize sharing.
Other <code>.in</code> files	Besides the previous examples, configure can be used to generate any kind of file from a <code>.in</code> template file. The substitution strings are expanded just as they are in makefiles. An example of a generated file that takes advantage of this is <code>VARSP.pl.in</code> , found in the top-level directory. It is used by the Perl script <code>makefiles-gen.pl</code> to do substitution in makefiles at installation time in the same manner as the <code>configure</code> script by storing the expanded strings in a Perl hash.

Chapter 7. Compiling VR Juggler

To improve portability and consistency across platforms, VR Juggler uses GNU make for building the library (installation is covered Chapter 8, *Installing VR Juggler*). The makefiles used employ many features that are unique to GNU make thus requiring that people who wish to compile Juggler from its source are required to have it installed. The Juggler makefiles are known to work with GNU make versions 3.78 and newer. To make sense of this documentation, please understand the concepts listed in the section called “Required Reading”. Posting about these subjects to the mailing list will probably be ignored.

In this chapter, the following conventions are used for text formatting and naming:

- Program, file and directory names are represented `this way`.
- Commands that are intended to be run by the user are written in **this style**.
- References to system and library calls are written as `call_name(##)` where “##” is the manual section where that call’s documentation can be found.
- Makefile targets are named as `'target'`, and makefile variables are named as `$(VAR_NAME)`.
- Environment variables are named as defined in the “Environment Variables” chapter in the *Getting Started Guide*.

General Quick Start

As mentioned in the introduction, Juggler requires the use of GNU make for compiling. This section provides a very short introduction to the use of make in general. It is assumed that the GNU make command is “gmake”. To verify that the version of GNU available is one of the known working versions, do:

```
% gmake -v
```

It will return something similar to the following output:

```
GNU Make version 3.79.1, by Richard Stallman and Roland McGrath.  
Copyright (C) 1988, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 2000  
Free Software Foundation, Inc.  
...
```

The emphasized text is the version information. For best results, this should be one of the version numbers listed in the section called “Required Tools and Utilities for Building VR Juggler from Source”.

Most, if not all, versions of make (not just GNU make) use the following syntax for execution:
`make [[VARIABLE=value...] [target...]]`

The options in square brackets are not required. The `[. . .]` strings mean that any number of the previous option can be given. If executed with no arguments, the default target will be built. This is either the first target defined in the corresponding makefile or one explicitly denoted as 'default' in the makefile. Typically, the default target is 'all'. Any target(s) can be built by simply naming them on the command line.

Variables used in the makefile can be overridden by command-line arguments. Simply name the variable (e.g., `CC`, the C compiler command) and give it a value (e.g., `CC=/usr/local/bin/gcc`). This forces all references to the variable in the makefile to use what is on the command line.

Other useful options for GNU make are the following:

- `-k`: Keep going even when errors are encountered
- `-f filename`: Explicitly name the makefile to use instead of the default (either `Makefile` or `makefile` found in the current directory)
- `-j [count]`: Do a parallel build of the target(s), optionally limiting the number of processes spawned to `count`
- `-l float-value`: Limit the load on the machine to the specified value when doing a parallel build

Targets

This section describes all the targets related to building VR Juggler from its source code. The targets are grouped together depending upon how they are related. For information on what is actually done as part of the compiling process, see the section called “Process of Building (Individual Steps)”. A current list of all the targets (with descriptions) can be found at the top of `Makefile.in` in the top-level VR Juggler source directory.

world Clean up the build environment and then build and install everything using the default ABI and ISA. This is a simple target for those who just want to build and install VR Juggler as simply as possible. “Everything” in this case is the following:

- Debugging and optimized versions of the library binaries
- Shared and static versions of the library binaries (if both are supported on the target platform)
- Header files
- Sample applications, test code and user tools
- Data files (sample config files, model files, etc.)
- VjControl (if it was built)

world-all-abi This is the same as the 'world' target except that it builds and installs *all possible* ABI and ISA combinations for the target platform. On IRIX, for example, this means that all combinations of N32, 64, mips3 and mips4 (debugging and optimized versions) are built and installed. Most platforms currently support only one ABI/ISA combination thus making this target the same as 'world'.

buildworld This target is the same as 'all'; In other words, it builds everything. It executes the first phase of the 'world' target (i.e., only the build phase, not the install phase). Since it builds both debugging and optimized versions of VR Juggler without installing, it is useful for testing changes to the library code to ensure that it works in both the debugging and optimized cases.

all-abi In the same manner as 'world-all-abi', this target builds, but *does not install*, all possible ABI/ISA combinations for the target platform. Thus, it provides functionality that could also be called 'buildworld-all-abi', but the length of such a target's name is unwieldy.

- `debug` Build only the debugging version of the library binaries (both static and dynamic). In other words, the libraries are built so that debugging symbols are turned on. It is the combination of 'dbg' and 'ddso' (see below). *This is the default target and is what gets built if running **make** with no arguments.*
- `optim` Build only the optimized version of the library binaries (both static and dynamic). This is built with no debugging symbols at all. It is the combination of 'opt' and 'dso'.
- `dbg` Build only the *static* debugging version of the libraries. This does the same thing as 'debug' but does not compile the dynamic libraries.
- `ddso` Build only the *dynamic* debugging version of the libraries. This does the same thing as 'debug' but does not compile the static libraries.
- `opt` Build only the *static* optimized version of the libraries. This does the same thing as 'optim' but does not compile the dynamic libraries.
- `dso` Build only the *dynamic* optimized version of the libraries. This does the same thing as 'optim' but does not compile the static libraries.
- `gui` Compile the VjControl Java files and collect them into `VjControl.jar`. This is only done if the configure script found a working Java compiler during its phase of the compile process.
- `links` Set up the developer pseudo-installation environment. More detail is given on this subject below in the section called "Developer Installation".
- `clean` Clean up everything in the build environment. This uses a 'clean' target defined in `Makefile.base` that is automatically shared by all makefiles that include that file. The cleaning process is recursive just as the build process is. Each makefile defines which files are safe for cleaning, but generally core files, compiler-generated files and object files are the only things removed during this process.
- `cleandepend` Clean up the automatically generated dependency files (the `.d` files in each directory). This method for cleaning up deletes only these files and nothing else--ever.
- `clean-links` Remove the developer pseudo-installation environment. More detail is given on this subject in the section called "Developer Installation".
- `clobber` Clean up (clobber) the entire build environment except what was generated by **configure**. This runs the previous clean-up targets and removes the object directory(ies) and `lib` directory(ies). Its

purpose is to totally reset the build environment to its state just prior to running **configure**.

info Print out all the targets with short descriptions as well as default values for some variables that may be overridden by the user. For more information about variables in the Juggler makefiles, see the section called “Useful Variables”.

Build targets executed from the top-level directory generate the object files in a directory tree created during compilation named as `obj/platform/ABI/ISA/opt|debug`. It is always a subdirectory of the top-level directory in the build environment. The path to this directory is passed as an argument to the recursive make process overriding the `$(OBJDIR)` variable.

Similarly, a subtree is made for the library binaries. It is named as `lib/ISA/opt|debug` (or `lib32/...` or `lib64/...` depending on the platform and the ABI). Since the library binaries are built only by the Makefile in the top-level directory, it is only used by builds from that directory.

Targets such as `'dbg'`, `'opt'` and `'clean'` are available for use in the individual subdirectories. Building a target from within a subdirectory is localized to that subdirectory. Thus, all object files would be generated in the directory where the target was run rather than in the `obj` subdirectory. For example, the following process would build only the object files in the `Config` directory:

```
% cd Config
% gmake opt
```

Those object files can be removed by running the `'clean'` target in that directory.

Useful Variables

As with all **make**-based build systems, variables used in the makefiles can be overridden by the user. This section describes variables that developers may find useful to override while compiling. The variables are listed alphabetically.

prefix Provide an alternate installation prefix than what was set at configuration time. Refer to Chapter 8, *Installing VR Juggler* for more information about installing VR Juggler.

vjABI Define an alternate ABI/ISA combination than the default set by **configure**. This accepts exactly the same values as the `--with-abi` argument to the configure script. Results of its use can be seen near the bottom of `Makefile.base`.

NO_CLEAN Do not clobber the build environment (via the `'clobber'` target) before running the `'world'` target. Set this variable to 1 to enable it.

NO_JAVA_GUI Disable compiling and installing of VjControl by setting this variable to 1 when building. This is very handy when VjControl is not building properly or is not needed for whatever reason.

OBJDIR Set an alternate directory in which object files are created. The default method is to use the platform-

and ABI-specific directory when compiling from the top-level directory or to use the current directory when compiling in a subdirectory.

OBJ_FILE_SUFFIX Choose an alternate suffix for object files that are compiled. When compiling with MS Visual C++, “obj” is the default. In all other cases, “o” is the default. Only specify the suffix, not the ‘.’ as that is always present.

PERL Provide a different version of a Perl 5.004 or newer binary than what **configure** found.

L

GNU Make Options

Two options can be used with GNU make when compiling VR Juggler. They provide the capability to build the library in parallel to take advantage of hardware that can do this effectively. To enable parallel building, specify the `-j` argument on the command line. With no argument, there is no limit to the number of jobs that are spawned by the build process. An integer argument limits the number of jobs to that value.

The second argument enforces load balancing on the machine while doing a parallel build. To do this, use the `-l` option with a floating-point value as the load limit. Note that to be allowed to do this, GNU make must be installed setgid on many platforms. When used in combination with the `-j` option, parallel compiling can be done while not loading the machine down too heavily.

Process of Building (Individual Steps)

This section describes the steps taken by the 'world' target as it *builds* VR Juggler. All other build targets are subsets of 'world', and this information can be applied accordingly. Note that 'world' can only be run from the top-level directory of the build environment. The steps are:

1. Clobber the build environment so that it is as it would be after running **configure** in a clean directory tree.
2. Build the object files for the optimized version of the library binaries. Part of this process is to generate dependencies for each `.c` and `.cpp` file on the fly. These dependencies are listed in corresponding `.d` files and are updated automatically whenever a change in a source or header file is detected. All targets, except 'cleandepend', run in the subdirectories of the source tree (not just the one that builds optimized object files) ensure that dependencies are up to date all the time. Compiling from the top-level directory of the build environment puts the optimized object files in a platform- and ABI-specific subdirectory named as `obj/platform/ABI/ISA/opt`.
3. Create the static and dynamic versions of the library binaries from the optimized object files. The binaries are built in the subdirectory named as `lib/ISA/opt` and are linked to from the `lib` directory. Note that `lib` may be named differently on some platforms. On IRIX, for example, depending on the default ABI, it may be named `lib32` or `lib64`. It is referred to as `lib` for simplicity.
4. Repeat steps 2 and 3 for the debugging version of the object files and library binaries.
5. Once all libraries are built, links are set up in the `lib` directory and its ISA subdirectory (if there is one) that point to the default library version. This is currently the debugging version since that is what developers are likely to use most frequently. These links overwrite those made in step 3 (and repeated in step 4).
6. Compile VjControl if the Java compiler is available and compiling of it has not been disabled by the user (refer to the section called “Useful Variables” for more information).

The remainder of the work done is installation. More information on installing is available in Chapter 8, *Installing VR Juggler*.

Developer “Installation”

To facilitate development of VR Juggler applications, an environment is created after compiling that simulates a fully installed version of the library and its components. This is done entirely within the build environment and is rooted in a directory called `instlinks` (a subdirectory of the top-level build directory).

When the build process completes, a message is printed telling the developer the value to which s/he should set the environment variable `$VJ_BASE_DIR`. By doing this simple step, the developer is given a pseudo-installation where Juggler applications (that are part of the source tree) can be compiled and executed. These applications are linked statically on IRIX, but on all other UNIX-based platforms, they are linked dynamically. In those cases, `$LD_LIBRARY_PATH` should be set to the value `$VJ_BASE_DIR/lib` which is a normal step in the setup process for using an actual installation.

As of this writing, this environment is not available on a Win32 platform. This is because the creation of it relies heavily on symbolic links. Because of this, Juggler must be fully installed for application development. Suggestions for enabling an equivalent development environment for Win32 are welcomed. It is possible, after installing Juggler, to continue working within the build environment. Set `$VJ_BASE_DIR` to the installation directory and continue working as on any other platform. All library binaries and headers will be referenced from the installed version, but source code (for applications as an example) comes from the Juggler source tree.

Chapter 8. Installing VR Juggler

To improve portability and consistency across platforms, VR Juggler uses GNU make for installing the library (compiling is covered in Chapter 7, *Compiling VR Juggler*). The makefiles used employ many features that are unique to GNU make thus requiring that people who wish to compile Juggler from its source are required to have it installed. The Juggler makefiles are known to work with GNU make versions 3.78.1 and newer. To make sense of this documentation, please understand the concepts listed in the section called “Required Reading”. Posting about these subjects to the mailing list will probably be ignored.

In this chapter, the following conventions are used for text formatting and naming:

- Program, file and directory names are represented `this way`.
- Commands that are intended to be run by the user are written in **this style**.
- References to system and library calls are written as `call_name(##)` where “##” is the manual section where that call’s documentation can be found.
- Makefile targets are named as `'target'`, and makefile variables are named as `$(VAR_NAME)`.
- Environment variables are named as defined in the “Environment Variables” chapter in the *Getting Started Guide*.

General Quick Start

Please refer to the section called “General Quick Start”. The information there provides everything needed to get started with installing Juggler after building it.

Targets

This section describes all the targets related to installing VR Juggler *from its source code*. The targets are grouped together depending upon how they are related. For information on what is actually done as part of the full installation process, refer to the section called “Process of Full Install (Individual Steps)”. A current list of all the targets (with descriptions) can be found at the top of `Makefile.in` in the top-level Juggler source directory.

<code>installworld</code>	This target is the same as <code>'install-all'</code> . It performs a full installation of library binaries (both debugging and optimized for the default ABI/ISA combination) and all associated files. The steps taken for this installation are as described in the section called “Process of Full Install (Individual Steps)”. This is the essentially second phase of the <code>'world'</code> target described in Chapter 7, <i>Compiling VR Juggler</i> . It requires that <code>'buildworld'</code> be successfully completed beforehand.
<code>install-all-abi</code>	This is the same as the <code>'installworld'</code> target except that it installs <i>all possible</i> ABI and ISA combinations for the target platform. On IRIX, for example, this means that all combinations of N32, 64, mips3 and mips4 (debugging and optimized) are installed. Most platforms currently support only one ABI/ISA combination thus making this target exactly the same as <code>'installworld'</code> . The steps taken for this installation follow those described in the section called “Process of Full Install (Individual Steps)”.

install-debug	This target is the same as 'install'. It installs only the debugging version of the library binaries (both static and dynamic) and all associated files. It includes the combination of 'install-dbg' and 'install-ddso' (see below). The steps taken for this installation follow those described below. This is the target that is run when using the traditional make install command.
install-dbg	Install only the <i>static</i> debugging version of the library binaries. Nothing besides the libraries is installed.
install-ddso	Install only the <i>dynamic</i> debugging version of the library binaries. Nothing besides the libraries is installed.
install-optim	Install only the optimized version of the library binaries (both static and dynamic) and all associated files. It includes the combination of 'install-opt' and 'install-dso' (see below). The steps taken for this installation follow those described in the section called "Process of Full Install (Individual Steps)".
install-opt	Install only the <i>static</i> optimized version of the library binaries. Nothing besides the libraries is installed.
install-dso	Install only the <i>dynamic</i> optimized version of the library binaries. Nothing besides the libraries is installed.
install-headers	Install all the header files from the VR Juggler source tree.
install-samples	Install the sample applications and generate all the application makefiles from the <code>Makefile.in</code> templates. This makes a copy of the <code>samples</code> directory tree found in the VR Juggler source.
install-test	Install the test code and generate all the application makefiles from the <code>Makefile.in</code> templates. This makes a copy of the <code>test</code> directory tree found in the VR Juggler source.
install-tools	Install the user tools and generate all the necessary makefiles from the <code>Makefile.in</code> templates. This makes a copy of the <code>tools</code> directory tree found in the VR Juggler source.
install-data	Install the standard VR Juggler data files. This makes a copy of the <code>Data</code> directory tree found in the VR Juggler source.

- `install-gui` Install the VjControl JAR file and its runnable files. In a UNIX environment, the `vj#control` scripts are installed. On a Win32 platform, the `vjcontrol.bat` batch file is installed. This is done only if the `configure` script found a working Java compiler and the target is not being disabled via the `$(NO_JAVA_GUI)` variable.
- `hier` Build the directory hierarchy for the installation. This is done before any of the “full” install targets (`installworld`, `install-all-abi`, `install-debug` and `install-optim`) is run. This is useful for testing parts of the installation or doing partial installations. Run it before doing any testing to create the full install hierarchy and then run the selected install targets. For example, to install only the static debugging libraries with nothing else, do:
- ```
make hier install-dbg
```

## Useful Variables

As with all make-based build systems, variables used in the makefiles can be overridden by the user. This section describes variables that developers may find useful to override while installing. The variables are listed alphabetically.

- `prefix` Provide an alternate installation prefix than what was set at configuration time. This is handy if testing the installation process or if no prefix was given at configuration time. Specifying a value for this variable is all that is necessary to change the entire installation path of Juggler.
- `vjABI` Define an alternate ABI/ISA combination than the default set by **configure**. This accepts exactly the same values as the `--with-abi` argument to the `configure` script. Results of its use can be seen near the bottom of `Makefile.base`. If this was used as part of compiling, it must also be used for installing to ensure that everything is installed properly.
- `NO_JAVA_GUI` Disable the installation of VjControl by setting this variable to 1 when building. This is very handy when VjControl is not building properly or is not needed for whatever reason.
- `PERL` Provide a different version of a Perl 5.004 or newer binary than what **configure** found. Perl is crucial to the installation process because several Perl scripts are used (see the section called “Custom Scripts”). If it is necessary to use a different perl binary than the one that **configure** found, it can be named with this variable. All Perl scripts will use it.

## Process of Full Install (Individual Steps)

This section describes the steps taken by the “full” installation targets `installworld`, `install-all-abi`, `install-debug`, and `install-optim` when they are run by the user. These four targets install library binaries and all associated VR Juggler files (data files, VjControl, headers, sample apps, etc.). The steps are:

1. Do any pre-installation steps necessary. This includes creating the entire installation directory hierarchy.

2. Install the library binaries (those that are appropriate for the target being run). Refer to the section called “Targets” for information on which binaries are installed by which targets. In particular, the 'installworld' target sets up links in the installed `lib` directory pointing to the default library version (currently the debugging version).
3. Install all the header files needed for compiling applications.
4. Install all the sample code provided as examples of working VR Juggler applications.
5. Install the test code provided for testing various aspects of VR Juggler (threads, math, input devices, etc.) to ensure that it is in good working order.
6. Install the various user tools included as helper applications in using VR Juggler.
7. Install all the data files (config files, gesture files, calibration tables, etc.).
8. Install VjControl (if it was built).
9. Do any post-installation steps necessary. Currently, nothing is done at this stage except informing the user that installation is complete.

## Custom Scripts

The installation process heavily depends on scripts written for VR Juggler. Most of the scripts themselves are generalized such that they are not specific to Juggler, and in the case of `mtree.pl`, the script is written to fill in if the `mtree(1)` application is not found by `configure`. The use of each script is described in detail here.

### `mtree(1)` and `mtree.pl`

`mtree(1)` is a utility for mapping directory hierarchies. It dates back to 4.3BSD-Reno, and development of it has continued in FreeBSD and in NetBSD. `mtree.pl` is a Perl script written to mimic the behavior of `mtree(1)` (the version distributed with FreeBSD 3.0) so that Juggler's installation can take advantage of its features on platforms without `mtree(1)`. The two are not completely functionally equivalent at this time, but the basic features needed by VR Juggler are present.

The key aspect of `mtree(1)` that the Juggler installation uses is the creation of directory hierarchies from specification files (named with the extension `.dist`). These specifications can be used to create the full installation tree with all directories guaranteed to have the same permissions and ownership. The actual specification files are generated by `configure` from template files to allow maximum flexibility. One key file, `VJ.lib.dist`, is generated by `configure` from scratch (i.e., without a template) so that it can take advantage of `configure`'s internal knowledge about the target platform.

The options that are used during installation are the following:

- U If any of the owner, group or permissions for a directory do not match the specification, update them accordingly. All three must be specified for a missing directory to be created. (Note that this extra stipulation is not yet implemented in `mtree.pl`.) On a Win32 platform, the owner and group are essentially meaningless because the Perl 5 port does not implement the functions that can read and modify ownership.
- d Ignore everything except directory file types. This is used only for ensuring the integrity of the directory hierarchy.
- e Ignore files in the hierarchy that are not in the specification. This is especially important since the VR Juggler specification files only contain directory information.

- f specification file      Read from the named specification file rather than from the standard input.
- p path                      Start at the name directory rather than the current directory.

**mtree(1)** provides a wide range of features but unfortunately not all are implemented in **mtree.pl**. For example, `be#` sides being used to build a hierarchy, it can be used to generate a specification file for an existing hierarchy. For more information about what can be done with this utility, refer to the manual page if **mtree(1)** is distributed with the operating system. More work on bringing the two closer together in functional equivalence is anticipated.

## install-dir.pl

This Perl 5 script is used to install an entire directory tree without any of the CVS directories. Its use is very simple. In general, the base of the tree to be installed and the destination directory are all that is necessary. The basic usage is:

```
install-dir.pl [{-i source directory} | {-o destination directory} | [-u user name] | [-g group name] | [-m mode]]
```

Options for the script are as follows:

- i source directory      Name the directory to be installed. *Required.*
- o destination directory      Name the directory to which source directory is installed. *Required.*
- u user name              Name the user that will own the installed files and directories. This has no meaning on a Win32 platform.
- g group name              Name the group that will own the installed files and directories. This has no meaning on a Win32 platform.
- m mode                      Provide the permission bits (UNIX octal style) of the installed files. Note that this sets the same permissions for all files. If used in combination with **mtree(1)**, permissions can be set on a per-file basis after this script is run.

Currently, this script is used to install the `Data` directory since it contains no source files (see **install-src.pl** below), and it is used in the process of creating `VjControl.jar`. Thus, it is versatile enough to be used during both the compile and install phases.

## install-src.pl

Similar to **install-dir.pl**, this script installs the contents of a directory tree to a specified location. It is, however, more selective in what it installs. Only source files of recognized types (`.txt`, `.TXT`, `.C`, `.c`, `.h`, `.cxx`, `.CXX`, `.cpp`, `.CPP` and `makefiles`) are copied from the source to the destination. Any CVS directories are not copied over. The options for this script are exactly the same as those for **install-dir.pl**.

## makefiles-gen.pl

This script is used to perform the same functions as **configure** when it expands the “`@. . .@`” strings in `.in` template files. Its use is reserved for the installation phase rather than the configuration phase. The installation process copies over many `Makefile.in`'s that **configure** would normally generate for use by developers. To simplify maintenance, the same template files are used for installations that users can access. Since the templates still contain the “`@. . .@`” strings, they must be expanded again. Thus, **makefiles-gen.pl** steps in for this purpose.

Options for `makefiles-gen.pl` are slightly different from any of the other scripts. It uses GNU-style options for the sake of clarity since some of the options correspond directly to a “`@. . .@`” string to be expanded. The options

described here are required unless otherwise noted.

|                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>--vars=filename</code>           | The name of the file given for this option is a Perl file containing “@. . .@” substitution <i>values</i> . These are used when replacing the “@. . .@” strings in the template files. For more information about this file, see the paragraph following this list. Note that the file name given should be the one generated by <b>configure</b> and not the <code>.in</code> template version. The generated file has the expanded strings that this script needs for its job. |
| <code>--srcdir=path</code>             | This option is typically a simple value providing a path to the source code <i>relative to the makefile</i> . In almost all cases, it will be <code>`.'</code> since the source code and the <code>Makefile</code> being generated are in the same directory. This replaces <code>@srcdir@</code> .                                                                                                                                                                              |
| <code>--prefix=path</code>             | Use this option to name the base directory where the <code>Makefile</code> 's will go. For example, to generate the makefiles in the <code>samples</code> installed subtree, the path given is <code>\$(datadir)/samples</code> .                                                                                                                                                                                                                                                |
| <code>--srcdir=path</code>             | This option names the base of the directory tree containing the <code>Makefile.in</code> templates. In other words, it is a subdirectory in the Juggler source tree in which the recursive search for <code>Makefile.in</code> 's starts.                                                                                                                                                                                                                                        |
| <code>--SUBDIRS=directory list</code>  | In conjunction with the previous option, name the directories to be searched for <code>Makefile.in</code> templates. Only the directories here will be searched and used in the generation process.                                                                                                                                                                                                                                                                              |
| <code>--uname=owner's user name</code> | Give the name of the user who will own the installed files. This option has no meaning on a Win32 platform. This option is not required.                                                                                                                                                                                                                                                                                                                                         |
| <code>--gname=group name</code>        | Give the name of the group that will own the installed files. This option has no meaning on a Win32 platform. This option is not required.                                                                                                                                                                                                                                                                                                                                       |
| <code>--mode=mode bits</code>          | Give the permissions (UNIX octal style) to be set for the installed files. This defaults to the running user's <code>umask</code> . This option is not required.                                                                                                                                                                                                                                                                                                                 |

The key to this script is `VAR.S.pl.in`, located in the top-level VR Juggler source directory. It is a Perl file that can be evaluated at run time by the script to set values used for substitution. The format used in the file is:

```
$VARS{ 'VAR_NAME' } = '@VAR_NAME@' ;
```

Note that `VAR_NAME` should be the same in both places for ease of understanding. The `%VARS` hash is used in the substitution process. Occurrences of its keys found in a `Makefile.in` are replaced with the value associated with that key (the expanded “@. . .@” string). Also note the use of apostrophes to prevent expansion of variables by the Perl interpreter. Be careful to use the right string delimiters when adding new values.

## InstallOps Perl Module

Many of these Perl scripts use the `InstallOps` module (found in `InstallOps.pm` with the other scripts). This module provides subroutines implementing commonly used techniques in working with directory trees and files during installation. Ideally, it is general enough to be used in writing new scripts that can be added to the installation process. A short description of the available subroutines and their use is provided here.

```
recurseDir();
```

```
($start_dir, $base_inst_dir);
```

Recurse through the given directory tree beginning at `$start_dir`. The second argument, `$base_inst_dir`, names the base directory to which the files will be installed. This routine requires that the calling package define a subroutine called `recurseAction( )` that defines what actions to take when a normal file is encountered during the recursion process. It is only called when non-directory type files are found in the current directory of the recur#

sion process. Since this is specialized for installation, it ensures that the destination directories (rooted at `$base_inst_dir`) exist.

**newDir();**

`($base_dir, $newdir);`

Create a new directory (`$newdir`) in a given directory tree (`$base_dir`). If it already exists, it is not created.

**installFile();**

`($filename, $uid, $gid, $mode, $dest_dir);`

Install a given file (`$filename`) with specified permissions (`$mode`) in UNIX octal style to a destination directory (`$dest_dir`). Ownership is set based on the provided values for `$uid` and `$gid` (which has no effect on a Win32 platform).

**replaceTags();**

(Replace tags of the form `@...@` in the given file (`$infile`) with known replacement values (found in `%VARS`). The tags are the keys of `%VARS`, and the replacement values are the values associated with those keys.);

---

# Chapter 9. Maintaining and Extending the Build System

Someday, I will write this.

---

## **Part III. User Community**

---

---

# Table of Contents

- 10. Contributing to VR Juggler ..... 40
  - Reasons to Contribute ..... 40
  - Why is VR Juggler cool, why do I care? ..... 40
  - Why Should I, an Intelligent, Helpful Person, Contribute to the Development of VR Juggler? .. 40
  - I Want to Fix a Bug ..... 40
  - I Want to Add New Features ..... 40
  - I Want to Add Missing Features ..... 40
  - I Think the Current Developers are Overworked ..... 41
  - Yes, I Want to Contribute! Where Do I Go From Here? ..... 41
  - Rules for Contributions ..... 41
  - Submitting Patches to Fix Bugs ..... 41
  - Submitting New Features, Implementing Missing Features ..... 42
  - Creating and Submitting Patches ..... 42
  - Recommended Reading ..... 42
  - Making a Patch ..... 42
  - Submitting Patches ..... 44
  - How to Become a VR Juggler Committer ..... 45
  - Why create documentation? ..... 46

---

# Chapter 10. Contributing to VR Juggler

Part of being an open source project includes accepting contributions from the user community. When members of the user community contribute code to a project, everyone benefits. In this chapter, we explain why contributions are helpful, and we explain how to make contributions.

## Reasons to Contribute

People who have limited experience with open source projects may not be familiar with the ideas surrounding user contributions. Because users have direct access to the source code, they have the opportunity to fix bugs, add new features, and otherwise improve the project. When users make such changes, they send them to the development team for inclusion in the next release.

## Why is VR Juggler cool, why do I care?

VR technology as a whole is stagnating with its closed source solutions. This means that anyone new to VR need to completely develop a new solution, often inferior than others. Releasing an open VR system is vital to the development of VR. Now instead of reimplementing VR systems, people can focus on implementing new VR ideas and methods all relying on this standard technology to be there, and free from cost or legal ties.

## Why Should I, an Intelligent, Helpful Person, Contribute to the Development of VR Juggler?

Because VR Juggler will benefit from your efforts, that's why! As an OpenSource project, we want your help, your input, your ideas, and most of all, your code contributions. You don't have to be a hot-shot, fast CPUs, fast download programmer to contribute. VR Juggler is a big project, and it is still growing. Whether your background is in UNIX-based systems or Win32, you can help. We want motivated, creative people to help us make VR Juggler the world's best virtual reality platform. If you're still not convinced, take a look at the rest of this page for some more motivation.

## I Want to Fix a Bug

Only the most trivial software is bug free, and VR Juggler is certainly non-trivial. We do our best to track down and fix bugs, but our fly swatter is only so big. If you find a bug, then by all means, submit a bug report [[http://sourceforge.net/tracker/?group\\_id=8041&atid=108041](http://sourceforge.net/tracker/?group_id=8041&atid=108041)] and give us a patch! Don't worry, we won't be too embarrassed.

## I Want to Add New Features

As with most projects, the features present in VR Juggler are those deemed necessary by the developers and users. New features are added, but the developers with commit access only have so much time each day to devote to the advancement of the source. With a growing user base and advances in the field of virtual reality, new features help everyone. If you have an idea or some earth-shattering code, we want to see it!

## I Want to Add Missing Features

Everyone in the field of virtual reality has differing ideas, goals and equipment. If VR Juggler is to prove usable as the basis for advanced (and not-so-advanced) virtual reality applications, it will need to provide support for the needs of as many users as possible. Getting to this point means that missing features need to be added by someone. Eventually, the current developers could find the time to meet everyone's needs, but those needs can be met faster with the help of contributors.

The most obvious case of a missing feature is a driver for Device X. VR Juggler provides a highly flexible mechanism for adding a device driver to the source. If you have driver source code for a currently unsupported device, chances are that driver can be added to VR Juggler easily.

Other examples of missing features can include a wide variety of extensions. To save yourself time, it would be best to discuss your ideas in a public VR Juggler forum (insert `vrjuggler-devel` link) first so that you can get feedback on whether your ideas are applicable to the VR Juggler core or would be better suited to a higher-level tool.

## I Think the Current Developers are Overworked

The VR Juggler team loves what it does and is excited about VR Juggler and its continued development. We are, however, a small group with big ideas and busy schedules. We want your help because, with support from many bright, creative developers, we think that VR Juggler can achieve these big goals and more.

## Yes, I Want to Contribute! Where Do I Go From Here?

The remainder of this chapter details the rules for contribution, the steps to create and submit patches, the steps to become a full-fledged developer with commit access. Read on!

## Rules for Contributions

Now that you have decided that you're more than cool enough to contribute to VR Juggler, it is time for some ground rules. These rules are here to help the committers take advantage of your help with maximum efficiency and speed, so please do your best to keep them in mind.

## Submitting Patches to Fix Bugs

The best way to do this is with a bug report [[http://sourceforge.net/tracker/?group\\_id=8041&atid=108041](http://sourceforge.net/tracker/?group_id=8041&atid=108041)]. Copies of bug reports go to `vrjuggler-devel@lists.sourceforge.net` [[http://sourceforge.net/mail/?group\\_id=8041](http://sourceforge.net/mail/?group_id=8041)] where they are seen by everyone subscribed. Before submitting a bug report, however, please review the open bugs [[http://sourceforge.net/bugs/?group\\_id=8041](http://sourceforge.net/bugs/?group_id=8041)] to verify that the bug you have found is not already reported. If you have a fix for an open bug, attach your patch as a follow-up comment.

If you have found an unreported bug, your bug report should contain the following information:

- Who you are (so we can give you credit if your patch is committed)
- The version of VR Juggler you are using
- The environment in which you are working (for example, operating system name and version, compiler version, etc.)
- A full description of the bug
- How to repeat the problem
- How to fix the problem (in other words, explain how your patch fixes the problem)

The interested developer(s) will take responsibility for reviewing your patch and follow up with comments as appropriate. If and when your patch is committed, you will be given due credit for the submission. Woo hoo!

## Submitting New Features, Implementing Missing

## Features

If you have a new feature to submit for addition to VR Juggler, you have hit the big time, baby! The best way to present what will most likely be a large patch is to post it on `vrjuggler-devel@lists.sourceforge.net` [`mailto:vrjuggler-devel@lists.sourceforge.net`] for review. When doing this, please include the following information:

- A *full* description of your feature and how it works
- Some reasoning about why your feature should be added
- The patch itself as a text/plain attachment if possible (this makes it easier for us to apply the patch and test it out)

Ideally, your patch will include fully documented code so that the developers do not have to read the code, get a good understanding of it and document it themselves. Including documentation, therefore, is a big plus when trying to get something committed. Hint, hint.

## Creating and Submitting Patches

This document explains in detail how to create patches to the VR Juggler developers for reviewing and (hopefully) committing, and it provides pointers on what the best audience for your patch may be. The information presented is based on the method used by the UNIX `diff(1)` and `patch(1)` utilities because that is what CVS uses.

## Recommended Reading

- Rules for making contributions (see the section called “Rules for Contributions”)
- CVS documentation
- `diff(1)` man page

## Making a Patch

There are two ways to create a patch to the VR Juggler source: using a copy of the source code checked out from CVS or making a comparison against an unmodified copy of a file. The first way is more convenient, but the second method does not necessarily require CVS knowledge.

## Using CVS to Make a Patch

If you have checked out a copy of the VR Juggler source code and have been modifying the code in that tree, making a patch (big or small) is easy. (Refer to the CVS documentation for information on how to check out a copy of VR Juggler from the CVS repository.) You can edit the checked-out file(s) all you want without retaining a backup copy because you can always get an unmodified version from the repository.

Once your work is done and you are ready to create the patch, you want to use the `cv diff` command to get the differences between the original version and your new work. This command is capable of doing a lot more than just creating patches, but we will focus only on that aspect of it here. Assuming a shell that allows redirection of standard output to a file, the command should be used as follows to create the patch for a single file:

```
% cvs diff filename filename.patch
```

Since CVS knows about the entire repository, you can use **cv diff** to make a larger patch of several files, a single directory, a subtree of the larger directory tree or the entire VR Juggler source tree. You should be careful to keep only related patches together in a single file. Examples of how to perform each of these operations follow.

- Patching several files: You can of course use wildcard expansions here as supported by your shell.

```
% cvs diff filename1 filename2 ... patch
```

- Patching all the files in a directory (either the current directory or some subdirectory of the whole VR Juggler source tree):

```
% cvs diff directory_name patch
```

Here, you can use ``.`` to represent the current directory as usual. If the directory you are patching contains sub# directories, CVS will recurse into those by default and perform the diff operation there as well. To prevent re# cursion, use the standard `-l` (local) CVS command option:

```
% cvs diff -l directory_name patch
```

- Creating a patch against the entire VR Juggler source tree works just the same as the previous method except that is executed from the top-level directory of the source tree.

## Using diff(1) to Make a Patch

Instead of using CVS to get the source and make the comparison (which can be slow if your network connection to SourceForge is hurting), it is possible to make a patch the “old-fashioned way”. Before editing a file, however, *make sure to save a backup copy of it*. You need this for comparison once your modifications are complete. The common convention is to make a copy the old file with the same name and extra extension (for example, `vj# Thread.cpp.orig`). Once you are satisfied that your work is complete, use the **diff(1)** utility to create the patch as follows (assuming a shell that supports redirection of standard out to a file):

```
% diff [options] filename.orig filename filename.patch
```

This must be repeated for every modified file. Make sure that the original file always goes *first*.

A nice thing about the **patch(1)** utility is that it can extract multiple patches from a single file. Submitting a single file is certain to make the developers happier than dealing with individual patch files for every file you modified. (Of course, you should not just lump all your patches into one file if they are unrelated. Use common sense when creating the patch files, and everything will be fab.) Assuming a UNIX or UNIX-like shell with redirection and ap# pending of files, there are two ways to combine individual patches into a single file:

1. Create a patch file and append subsequent patches to it:

```
% diff [options] filename1.orig filename1 $HOME/my.patch
% diff [options] filename2.orig filename2 $HOME/my.patch
% ...
```

2. If that does not suit you, you can use the simple yet powerful **cat(1)** command to concatenate all your distinct patch files into one big patch:

```
% diff [options] filename1.orig filename1 $HOME/filename1.patch
% diff [options] filename2.orig filename2 $HOME/filename2.patch
```

```
% ...
% cat $HOME/filename1.patch $HOME/filename2.patch ... > my_big_patch
```

## Patch Formats

There are three styles of diffs that can be generated for use as patches: standard diff, unified diff and context diff. In the author's experience, unified diffs are the easiest to read, and standard diffs are the hardest to read. The following two pages from the online VR Juggler CVS repository show how unified and context diffs look:

- **Unified diff**  
[[http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/vrjuggler/juggler\\_1.0/configure.in.diff?r1=text&tr1=1.259&r2=text&tr2=1.260&diff\\_format=u](http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/vrjuggler/juggler_1.0/configure.in.diff?r1=text&tr1=1.259&r2=text&tr2=1.260&diff_format=u)] between Revisions 1.259 and 1.260 of `configure.in`
- **Context diff**  
[[http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/vrjuggler/juggler\\_1.0/configure.in.diff?r1=text&tr1=1.259&r2=text&tr2=1.260&diff\\_format=c](http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/vrjuggler/juggler_1.0/configure.in.diff?r1=text&tr1=1.259&r2=text&tr2=1.260&diff_format=c)] between Revisions 1.259 and 1.260 of `configure.in`

We cannot generate fancy, colored diffs

[[http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/vrjuggler/juggler\\_1.0/configure.in.diff?r1=text&tr1=1.259&r2=text&tr2=1.260&diff\\_format=l](http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/vrjuggler/juggler_1.0/configure.in.diff?r1=text&tr1=1.259&r2=text&tr2=1.260&diff_format=l)] for use as patches unfortunately. It certainly is nice that **cvswEB** [<http://stud.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/>] will do that, however.

To generate a unified diff, specify the option `-u` on the command line when running the `diff` command. To create a context diff, specify the `-c` option instead. To create a standard diff, give no options at all. These work with both the CVS diff command and the `diff(1)` utility. To always create a specific kind of diff by default with CVS, add the following line to your `$HOME/.cvsrc` file:

```
diff option
```

## Submitting Patches

Now that you have made your patches, it's time to send them off to the VR Juggler developers with commit access so that they can review them, commit them and praise you for your fine efforts. Before reading this section, please be sure to have read the section called "Rules for Contributions". Failing to follow these rules could result in your slow review and/or acceptance of your patches. What is presented here will repeat some of what is written in that document, but it is important to read both.

The method used to present your patches to the committers depends on what kind of patches you have made. As of this writing, the most common types of patches we would like to see and expect to see are as follows:

- *Bug fixes*: VR Juggler is not free of bugs, and patches from users who fix bugs are very welcome
- *Feature enhancements*: VR Juggler offers a lot of features to users, but some interfaces may be incomplete or could be further extended
- *Feature additions*: VR Juggler needs more device drivers and abstractions useful to programmers who are developing VR applications every day

If your patch is accepted, you will be given proper credit in the commit log as follows:

```
Description of patch
```

Submitted by: Helpful User [great.person@wonderful.net](mailto:great.person@wonderful.net)

## Submitting Bug Fixes

This is probably the most common type of patch VR Juggler users will submit. The best way to submit a patch for a bug is to attach it to a bug report [[http://sourceforge.net/tracker/?group\\_id=8041&atid=108041](http://sourceforge.net/tracker/?group_id=8041&atid=108041)] through Source# Forge [<http://sourceforge.net/>]. Copies of bug reports are sent to [vrjuggler-devel@lists.sourceforge.net](mailto:vrjuggler-devel@lists.sourceforge.net) [[http://sourceforge.net/mail/?group\\_id=8041](http://sourceforge.net/mail/?group_id=8041)] where all subscribers of that list see them. The bug report will be as# signed to a developer with commit access, and that person will take charge of reviewing your patch and any further communication with you, the helpful submitter.

## Submitting Feature Enhancements

Submitting an enhancement to existing code requires a little bit more work than submitting a bug report. If you are implementing part of an interface that is not yet finished (for example, a method that only prints out “Not imple# mented yet!”), you should submit a bug report [[http://sourceforge.net/tracker/?group\\_id=8041&atid=108041](http://sourceforge.net/tracker/?group_id=8041&atid=108041)] as above but describe it as a feature enhancement in your report. If you are extending an interface, you should first submit your proposed changes (in a high-level format with some pseudocode if appropriate) to [vrjuggler-devel@lists.sourceforge.net](mailto:vrjuggler-devel@lists.sourceforge.net) [<mailto:vrjuggler-devel@lists.sourceforge.net>] for discussion before doing any hard work making the changes. This will allow interested people to respond with feedback, questions or a short “Go for it!” message. Once your patches are ready, then you should submit them to the list for reviewing before an interested committer will take them and actually commit them.

## Submitting Feature Additions

Feature additions stand to be controversial if not presented properly. For example, if you want to submit a complex computational geometry algorithm implementation, you had better have a really good reason. The first step you should take is a high-level proposal to [vrjuggler-devel@lists.sourceforge.net](mailto:vrjuggler-devel@lists.sourceforge.net) [<mailto:vrjuggler-devel@lists.sourceforge.net>] which will no doubt generate much good discussion among de# velopers. This step is encouraged to find out if people are interested in seeing your features as a part of the core VR Juggler library or would prefer to see them as something written at a higher level using VR Juggler as a basis. No one wants to see talented people doing a lot of work only to have it rejected because it doesn't fit with the intended design and use of a given project. (This is not intended to discourage people from trying to extend the design of VR Juggler but instead to suggest that VR Juggler be used as a *basis* for more complex tools when appropriate.)

After your proposal has been refined and heartily accepted, it's time for you to get to work on making the changes. Periodic progress reports are helpful so that the developers know that you are still hard at work and have not given up for lack of time or resources. Posting developmental patches at key stages helps too because the patches can be tested by other developers who can then provide feedback and bug reports.

Once your changes are ready (or some stage of your changes is ready to be committed), post your triumphant mes# sage with the attached patches. After all the traffic generated because of your work, the committers will no doubt jump at the chance to commit your changes and make them available to the community of users.

## How to Become a VR Juggler Committer

We are interested in expanding the list of official VR Juggler team members (i.e., developers with commit access). As an OpenSource project, VR Juggler is just getting started, but the current group has been working together suc# cessfully on the project for a long time. As more people become interested in the project and in furthering its goals, we hope to see more and more people joining the VR Juggler team.

If you are interested in getting commit access to the VR Juggler CVS repository, the best way to fulfill your interests is to contact us directly [<mailto:vrjuggler-contact@vrjuggler.org>] and let us know you'd like to join. If you have never contributed to VR Juggler before, you should provide us with some examples of your VR Juggler-related work or some basis for why you should be on the team. We will review your request and get back to you promptly.

Remember, it's up to you to prove to us how you can make VR Juggler better by having full commit access.

In the future, we would love to have the opportunity to invite people to join the team based on their contributions (which would be judged primarily on quality rather than on quantity). As part of the process for reviewing submitted patches, we will be watching for promising developers around the world who we would like to see having full commit access to VR Juggler.

## Why create documentation?

You, as a part of the Juggler development team, should want to write great documentation. Think of it as an investment to offload work. The more you educate everyone, the better they understand your stuff, the more likely that someone will spot your bugs, add new features, and most importantly use your feature, like you for it, and tell everyone about how cool you are. Do it for the people, but of interest to you is what you personally get back from writing documentation. Not just overwhelming satisfaction for a job well done, and not just an enormous lift of responsibility from not being the only one who understands that obfuscated code segment. It clarifies your thought process, uncovers bugs in your own logic, and produces that same clarity in other people who then love you for it - and in turn want to help you by developing more features and hunt for bugs.

Don't be lazy, document. It is vital to the success of *your* project!

---

# Part IV. Tutorials

---

---

# Table of Contents

11. Adding ... ..... 49

---

# Chapter 11. Adding ...

Chapters here are for tutorials explaining how to add various things. There is already a device driver tutorial in the *Programmer's Guide*.

---

# **Part V. Appendices**

---

---

# Table of Contents

A. Coding Standard ..... 52

---

# Appendix A. Coding Standard

New coding standard goes here.

---

# Index