VR Juggler

Getting Started Guide

VR Juggler: Getting Started Guide

Version 2.2

Published $Date: 2007-10-21 11:38:15 -0500 (Sun, 21 Oct 2007) $

Table of Contents

Preface vii
1. Installing VR Juggler 1
    Installing from a Compressed TAR File 1
    Installing from a ZIP File (Windows® only) 2
    Installing from a Disk Image (Mac OS X only) 4
2. Environment Variables 5
    How to Set Environment Variables 5
        Common Conventions and Background 5
        C-Style Shells (csh, tcsh) 5
        sh-Derived Shells (sh, ksh, bash, zsh, etc.) 6
        DOS Shell 6
        Windows® GUI 7
        Mac OS X 10
        Syntax Used in this Document 11
    Relevant Environment Variables 11
3. VR Juggler Sample Applications 18
    Tutorial Applications 18
    Advanced OpenGL Performer Applications 18
4. Compiling a VR Juggler Sample Program 20
    Required Reading 20
    Compiling an Application 20
        Compiling from the Command Line 20
        Compiling Using Microsoft Visual Studio 20
5. Running a VR Juggler Sample Program 25
    Required Reading 25
    Running an Application with a Simulator Configuration 25
        Starting the Application 26
        Basic Desktop Configuration Controls 33
        Using a Simulated Glove 35
        Using a Simulated Analog Device 35
    Running an Application in a VR System 36
        Single Machine Configuration 36
        Cluster Configuration 37
A. GNU Free Documentation License 38
    PREAMBLE 38
    APPLICABILITY AND DEFINITIONS 38
    VERBATIM COPYING 39
    COPYING IN QUANTITY 39
    MODIFICATIONS 40
    COMBINING DOCUMENTS 41
    COLLECTIONS OF DOCUMENTS 42
    AGGREGATION WITH INDEPENDENT WORKS 42
    TRANSLATION 42
    TERMINATION 42
    FUTURE REVISIONS OF THIS LICENSE 43
    ADDENDUM: How to use this License for your documents 43

List of Figures

1.1. Windows Folder View of ZIP File 2
1.2. Open WinZip Window 3
1.3. WinZip Extract Dialog 3
2.1. Windows® 2000 System Properties Dialog 7
2.2. Windows® XP System Properties Dialog 8
2.3. Windows® Environment Variable Editor Dialog 9
4.1. Selecting the Visual C++ Project File 21
4.2. MPApp Project 21
4.3. Project Menu 22
4.4. Build `MPApp.exe` 24
5.1. MPApp Running on a Linux Desktop with Multiple Input Windows 27
5.2. MPApp Running on a Linux Desktop with One Window 28
5.3. Setting Command Arguments 29
5.4. Execute `MPApp.exe` 29
5.5. MPApp Running on Mac OS X Using Cocoa with One Window 31
5.6. MPApp Running on Mac OS X Using X11 with One Window 33

List of Tables

5.1. Moving the simulated head 34
5.2. Moving the simulated wand 34
5.3. Moving the camera (multi-window configuration only) 35
5.4. Analog Device Simulator Keys 36

Preface

This book is for people who are just getting started with VR Juggler. It guides new users through getting and installing VR Juggler, configuring users' environment to use it, and compiling and running a sample application.

The prerequisites for reading this book are minimal. They are:

- Some experience with a command-line interface (i.e., a shell such as tcsh or the DOS shell)
- Creating and browsing directories

Those users who want to get more involved with VR Juggler to do more than just run applications should be aware right away of the following prerequisites:

- Knowledge of C++ and object-oriented design
- Knowledge of one of VR Juggler's currently supported graphics APIs (OpenGL [http://www.opengl.org/], OpenGL Performer [http://www.sgi.com/software/performer/], OpenSG [http://www.opensg.org/], or Open Scene Graph [http://www.openscenegraph.org/])

Chapter 1. Installing VR Juggler

As with most Open Source projects, VR Juggler is distributed as compressed archive files using popular formats. Installing a distribution requires very little effort, but you do need to know how to use archiving utilities to extract the installation tree. Automation of the installation is a goal of the VR Juggler team, but we are still finalizing the details of cross-platform installation management. Before reading further, you should know where you want to install VR Juggler, and you should make sure that you have access to write to that directory.

Installing from a Compressed TAR File

The TAR (Tape ARchive) format has been around for a long, long time in the UNIX world. It is simply a collection of files in a directory tree that are lumped into a single file suitable for writing to a tape or for downloading. The format is a standard, and the **tar**(1) utility is available on every UNIX-based platform and on Windows®. A free version can be downloaded from the GNU Project [http://www.gnu.org/]. A compressed TAR file is made for each VR Juggler distribution, and some distributions come in other formats as well. You can always count on the availability of a TAR file, though. The TAR files are compressed using either GZIP or BZIP2, both of which are standard compression formats. The **gzip**(1) utility is freely available from the GNU Project, and the **bzip2**(1) utility can be downloaded for free from Red Hat, Inc. [http://sources.redhat.com/bzip2/]. The GNU version of TAR has the GZIP and BZIP2 algorithms built in. The compression algorithm used can be determined by the file extension. Files compressed with GZIP end in `.gz`; files compressed with BZIP2 end in `.bz2`.

Once you have downloaded a VR Juggler TAR distribution, you can unpack it one of two ways depending on what your platform's version of TAR supports. Before extracting the installation tree, make sure that your current directory is the one where you want to install VR Juggler. If your version of TAR does not have GZIP built in (it does not support the `-z` option), the following command will do the decompression and extraction:

```
% gzip -cd vrjuggler-distribution.tar.gz | tar -xvf -
```

For versions of TAR without built-in BZIP2 support (there is no `-j` option) the command is similar:

```
% bzip2 -cd vrjuggler-distribution.tar.bz2 | tar -xvf -
```

Here, you should fill in `vrjuggler-distribution.tar.gz` (or `vrjuggler-distribution.tar.bz2`) with the name of the VR Juggler distribution file you downloaded. The above commands will work with any shell that supports redirection of standard output to a pipe. If that looks too scary, you can separate the decompression and extraction into two commands (for GZIP):

```
% gunzip vrjuggler-distribution.tar.gz
% tar -xvf vrjuggler-distribution.tar
```

or for BZIP2:

```
% bunzip2 vrjuggler-distribution.tar.bz2
% tar -xvf vrjuggler-distribution.tar
```

Note that the distribution file in the second command does not have the `.gz` extension after **gzip**(1) is run. These steps also work if your version of **tar**(1) supports the `-z` option (`-j` for BZIP2), but you can simplify your work if that option is supported. The following illustrates how to decompress and extract a TAR file compressed with GZIP all in one step:

```
% tar -xzvf vrjuggler-distribution.tar.gz
```

The following would be used for a TAR file compressed with BZIP2:

```
% tar -xjvf vrjuggler-distribution.tar.bz2
```

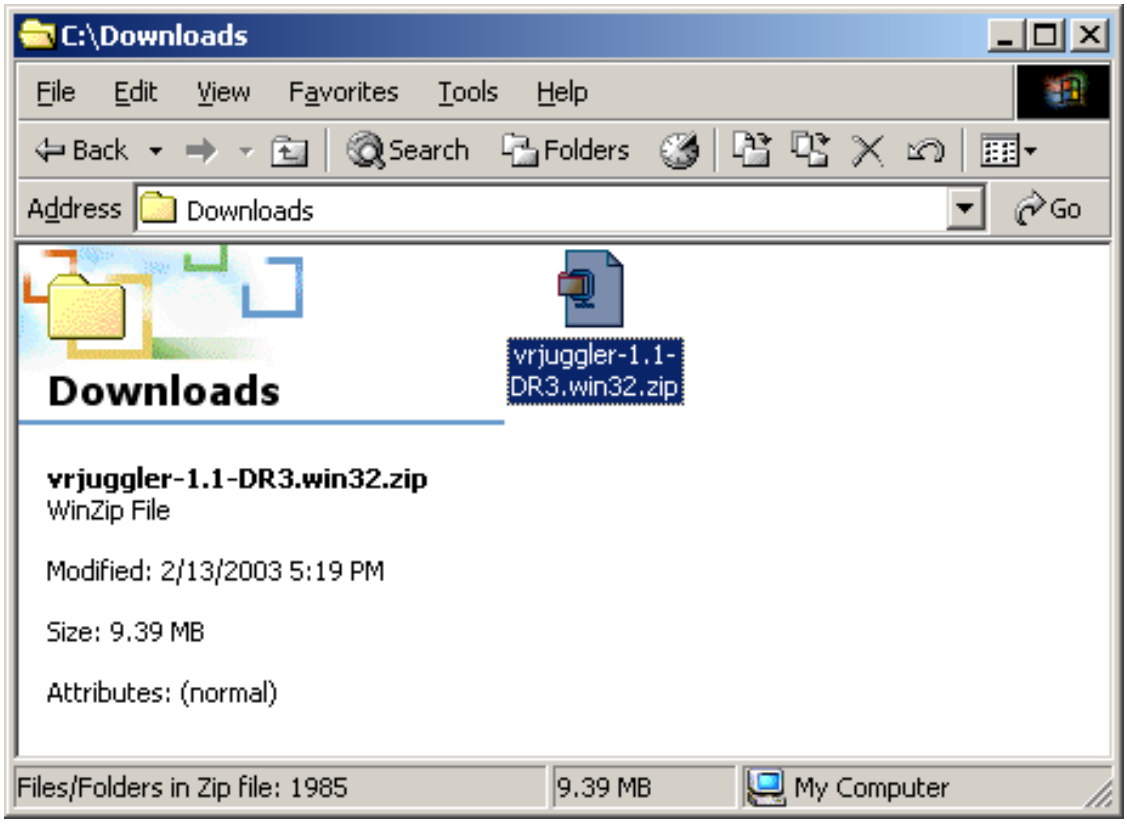
In either case, while the command runs, you will see the name of each file as it is written to disk. This is because of the `-v` option to **tar**(1) that tells it to be verbose in its efforts. **tar**(1) takes care of creating all the directories in the installation tree, so you only need to have the base directory (for example, `/usr/local`) when you start. For more information about these utilities, please refer to the **tar**(1) and **gzip**(1) manual pages.

Installing from a ZIP File (Windows® only)

On the Windows® family of platforms, the ZIP format rules. In the old days, you would use the PKZIP utility to decompress and extract a ZIP file. Nowadays, most people use Windows Explorer, WinZip [http://www.winzip.com/], or some other comparable graphical interface. This documentation covers only the use of WinZip when extracting a ZIP file.

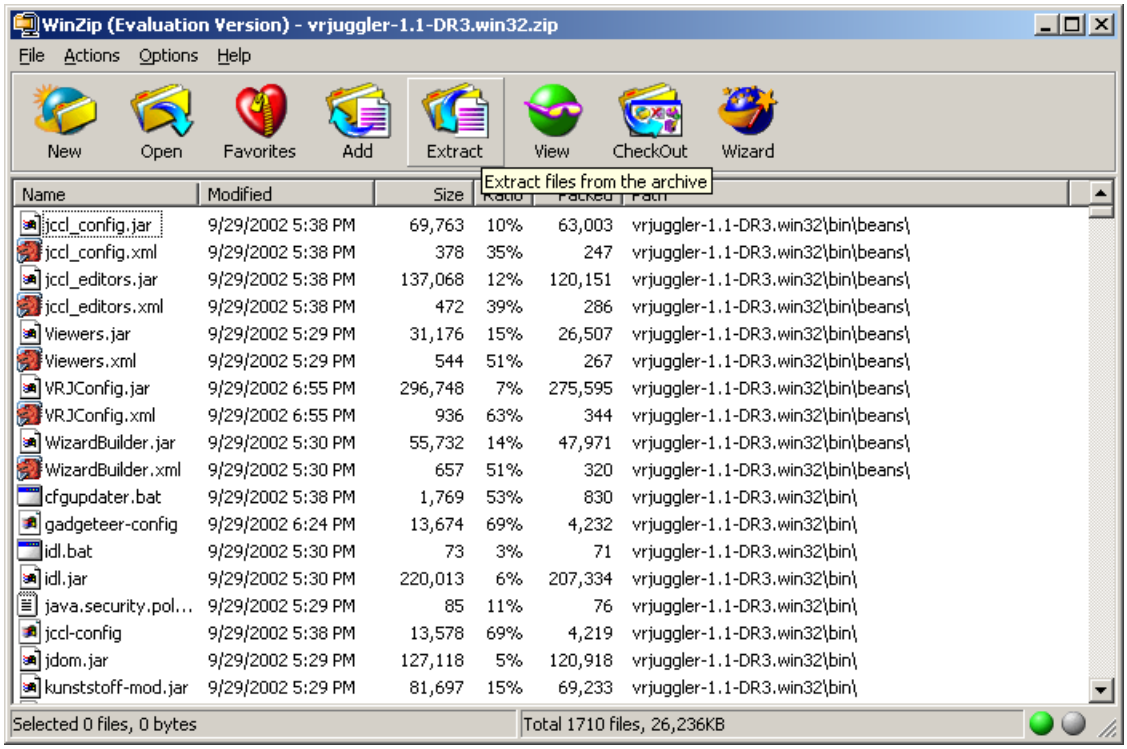
Once you have downloaded the VR Juggler ZIP file, the easiest way to extract it is to double-click on its icon in the open folder window as shown in Figure 1.1, “Windows Folder View of ZIP File”.

Figure 1.1. Windows Folder View of ZIP File



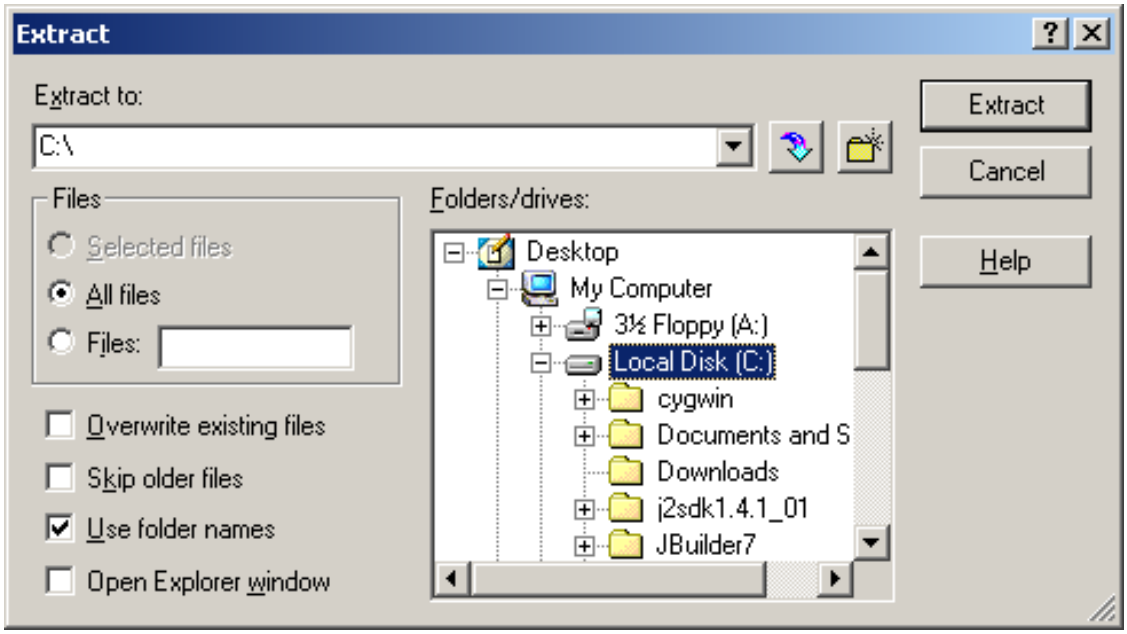
Double-clicking opens the main WinZip window, as shown in Figure 1.2, “Open WinZip Window”.

Figure 1.2. Open WinZip Window



Note that in this screen shot, the Extract button is highlighted. Click this button to open the following window. Note that in this screen shot, the Extract button is highlighted. Click this button to open the dialog box shown in Figure 1.3, “WinZip Extract Dialog”.

Figure 1.3. WinZip Extract Dialog



In this window, choose the directory where VR Juggler will be installed and click Extract. WinZip will then proceed to extract the ZIP file into the directory you named. That is all there is to it.

Installing from a Disk Image (Mac OS X only)

The preferred Internet distribution format for Mac OS X is the disk image (file extension `.dmg`). Double-clicking on the `.dmg` file mounts the disk image which can then be opened. Pre-compiled versions of VR Juggler for Mac OS X are distributed in this manner. Within the mounted disk image, there is an installer program for the VR Juggler release (file extension `.pkg`)[1].

Important

Before installing, read any HTML files included with the VR Juggler distribution. These will contain relevant information about updating an existing VR Juggler installation or how to use the VR Juggler installer that is not covered here.

Double-clicking on the VR Juggler installer will install VR Juggler. The VR Juggler libraries will be installed to `/usr/local`, and the dependencies will be installed to `/usr/local/vrjuggler-deps`. The application bundles for the Tweek Java GUI and VRJConfig will be installed to `/Applications`. Finally, the file `~/.MacOSX/environment.plist` will be updated to set the VR Juggler-related environment variables (see Chapter 2, *Environment Variables*).

Note

The application bundles for the Tweek Java GUI and VRJConfig are still in the early stages of development. For convenience, the command line versions of these applications are also installed. Running **tweek** or **vrjconfig** from a terminal window will start those applications just as would be done on other operating systems. At some point, the command line versions of these applications will probably be removed on Mac OS X in favor of exclusive use of the application bundles.

[1] Unfortunately, the complexity of VR Juggler prevents it from being installed using the preferred drag-and-drop method.

Chapter 2. Environment Variables

There are several *environment variables* that affect the way VR Juggler works. Some of these are required to compile and run applications while others are optional. This chapter lists all such variables and explains their meanings and uses.

How to Set Environment Variables

The syntax for setting or changing an environment variable varies with operating systems and shell interpreters. Instead of choosing one style of syntax that is specific to a particular shell type, we define our own syntax which you must then translate to your shell's specific syntax. Before defining this syntax, we present the method used to set environment variables in the three most common types of shells. We also provide a quick overview of how to set environment variables using Win32-based GUIs.

Common Conventions and Background

A convention used throughout this book is to name the variables using all capital letters. In almost all cases, regardless of the shell, this is the naming convention used for environment variables.

Setting a path with an environment variable can require special syntax. Because of this, the method for doing so may vary from shell to shell. Paths are important with VR Juggler when looking up the path to a shared library (dynamically linked library). For each shell, the syntax for setting a path is given.

Referring to environment variables can also vary from shell to shell. An example of how to print the value of an environment variable will be given for each shell. An example of how to refer to an environment variable is also provided as these two operations may vary even within one kind of shell!

In all shells, an environment variable is only available within that single shell instance. That is, setting an environment variable at a command prompt only affects that specific shell and will not be available from other concurrent or future shells. To make a setting “permanent”, it should be done in file read by all shell instances when they are started. This is addressed briefly as appropriate for each shell type.

C-Style Shells (csh, tcsh)

In a C-style shell (i.e., one whose interface is based on the C programming language), setting environment variables is done using the built-in command **setenv**. It is used as follows:

```
% setenv <VARIABLE_NAME> <value>
```

where the string `<VARIABLE_NAME>` represents the name of the variable you are going to set and `<value>` represents the value assigned to that variable. Both are required. If the named variable did not exist before, it will pop into existence. Otherwise, you overwrite the old setting with the new one.

To print the value of an environment variable, use the following command:

```
% printenv <VARIABLE_NAME>
```

Referring to a variable, however, is done using the following syntax:

```
% cd $VARIABLE_NAME/bin
```

Paths are specified as a colon-separated list. An example of this is:

```
% printenv PATH
/bin:/sbin:/usr/bin:/usr/sbin
```

For these types of shells, a “permanent” setting for a given variable should usually be done in your `.cshrc` file or in your `.login` file, both of which should be in your home directory. In most cases, it is better to use `.cshrc` because it is evaluated for every shell instance.

sh-Derived Shells (sh, ksh, bash, zsh, etc.)

In a shell based on sh, setting environment variables is done using the built-in command **export**. It is used as follows:

```
% export <VARIABLE_NAME>=<value>
```

or

```
% <VARIABLE_NAME>=<value>
% export <VARIABLE_NAME>
```

Here, the string `<VARIABLE_NAME>` represents the name of the variable you are going to set and `<value>` represents the value assigned to that variable. Both are required. Note that there is no space between the variable name and its value. If the named variable did not exist before, it will pop into existence. Otherwise, you overwrite the old setting with the new one. If the variable was already among your current shell's environment variables, the export command is not necessary.

To print the value of an environment variable, use the following command:

```
% echo $VARIABLE_NAME
```

Getting the value of a variable works the same way.

Paths are specified as a colon-separated list. An example of this is:

```
% echo $PATH
/bin:/sbin:/usr/bin:/usr/sbin
```

For these types of shells, a “permanent” setting for a given variable should usually be done in the `.profile` file in your home directory or in your shell's “rc” file. Different shells have different names for this file. Examples are `.bashrc` for BASH and `.zshrc` for Zsh. Please refer to your shell's documentation for more information. In any case, the file will be in your home directory.

DOS Shell

The typical syntax for setting an environment variable from the command line (in a DOS shell window) under Windows® is:

```
C:\> set <VARIABLE_NAME>=<value>
```

Here, `<VARIABLE_NAME>` is the name of the environment variable to be set, and `<value>` is the value being assigned to that variable. If the named variable did not exist before, it will pop into existence. Otherwise, you overwrite the old setting with the new one.

To print the value of an environment variable, use the following command:

```
C:\> set <VARIABLE_NAME>
```

Referring to a variable, however, is done using the following syntax:

```
C:\> cd %VARIABLE_NAME%\bin
```

Paths are specified as a semicolon-separated list. An example of this is:

```
C:\> set PATH
C:\WINDOWS;C:\bin;C:\
```

For some versions of Windows®, a “permanent” setting for a given variable should usually be done in `C:\AUTOEXEC.BAT`. In newer versions (Windows® ME in particular) and in the Windows® NT line of operating systems, the setting is done using the Control Panel. Please refer to the next section for more information on that method.

Windows® GUI

Before reading this section, please be sure to have read the section called “DOS Shell”. This is necessary because the Windows® GUI for setting environment variables is simply a front-end to that older method and thus uses the same conventions and syntax. The versions of Windows® to which this subsection applies are indicated individually since each is a little different. For more detailed information, please refer to the Windows® online help system and search for “environment variables”.

Windows® 2000 and Windows® XP

In the Control Panel, open the System icon. Under the Advanced tab, there is a button labeled Environment Variables, shown in Figure 2.1, “Windows® 2000 System Properties Dialog” (the Windows® XP version is shown in Figure 2.2, “Windows® XP System Properties Dialog”). Clicking this button opens the dialog box shown in Figure 2.3, “Windows® Environment Variable Editor Dialog”. Here, you can set variables for yourself and, if you have the access privileges, for all users.

Figure 2.1. Windows® 2000 System Properties Dialog

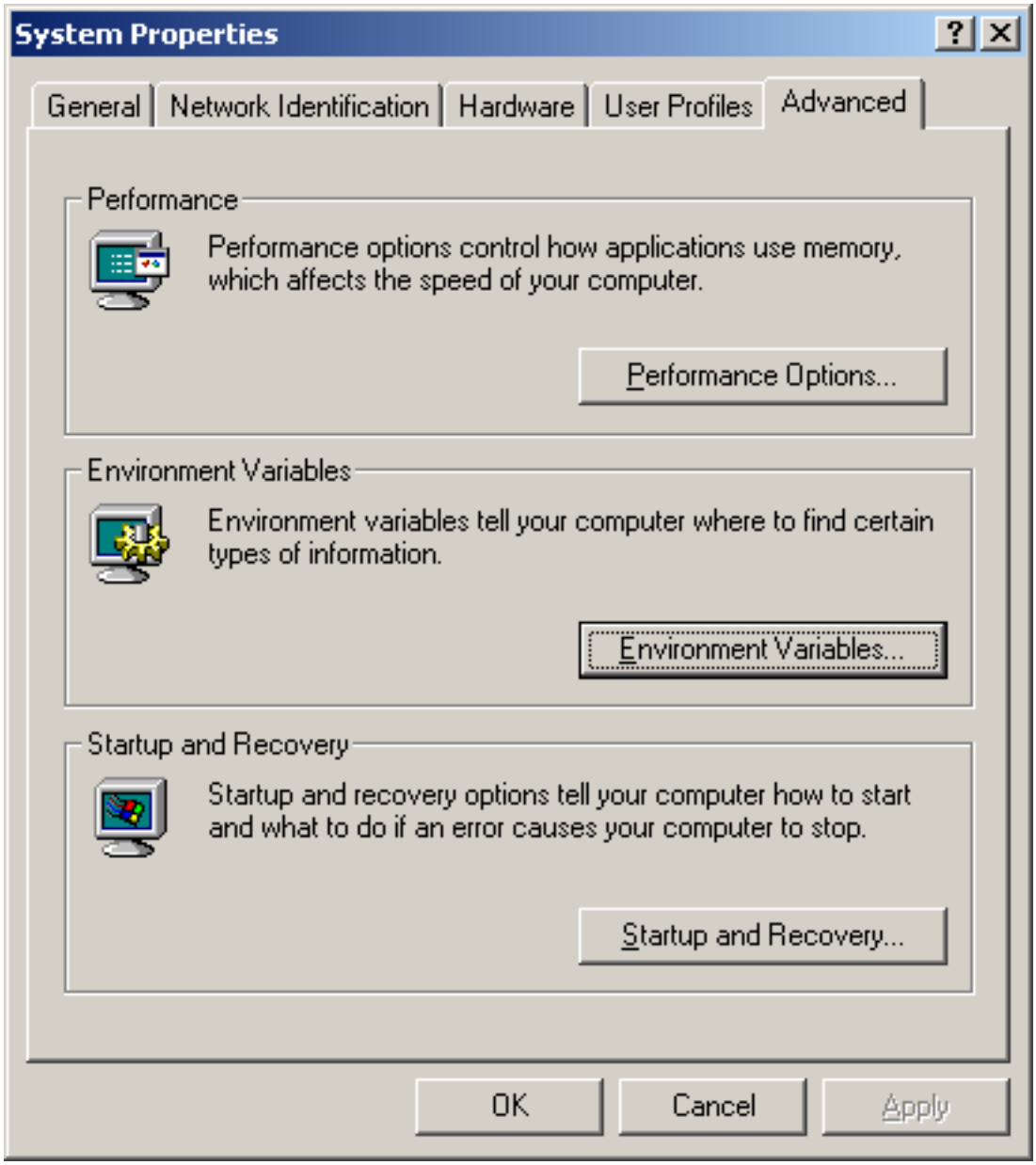


Figure 2.2. Windows® XP System Properties Dialog

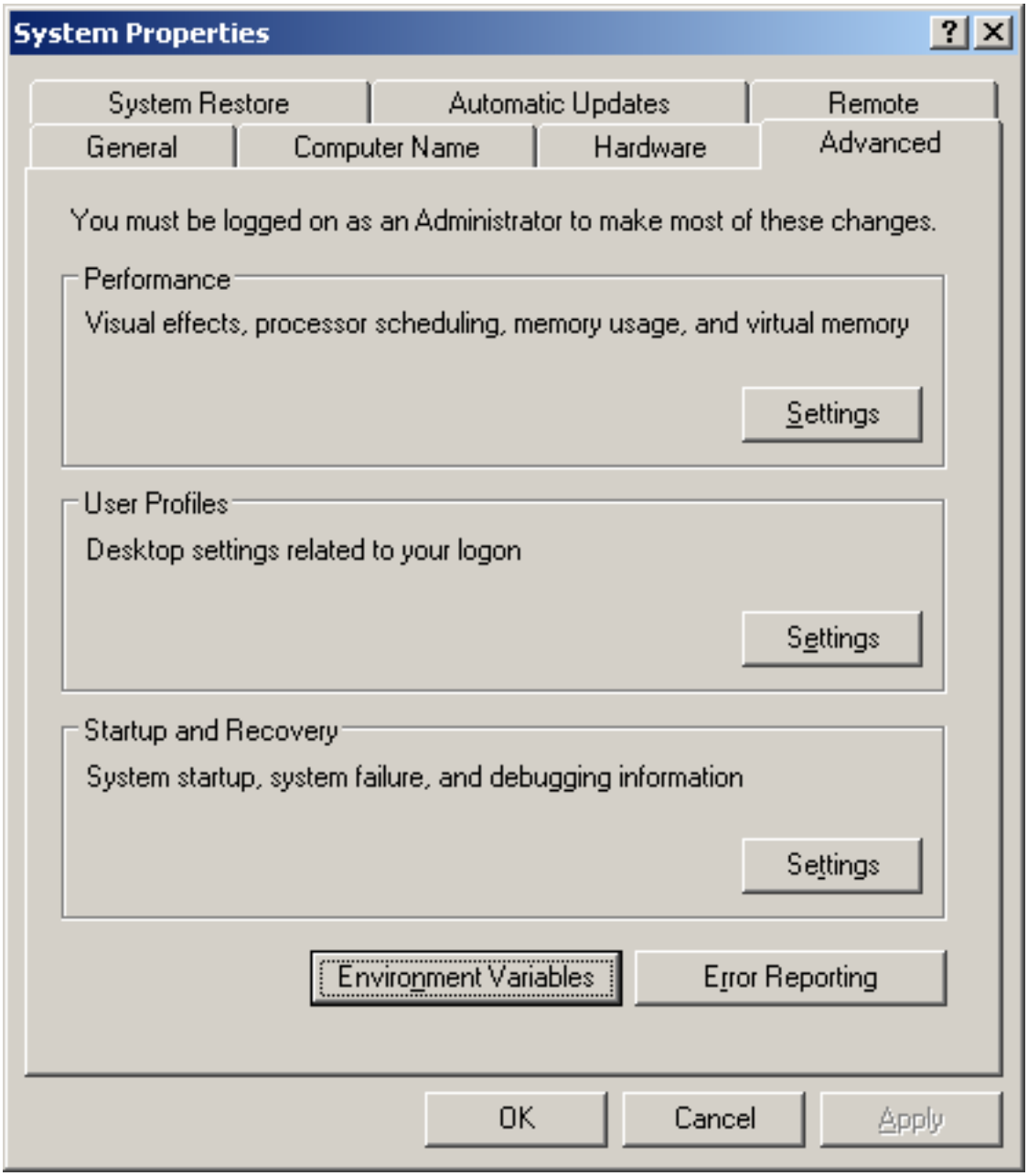
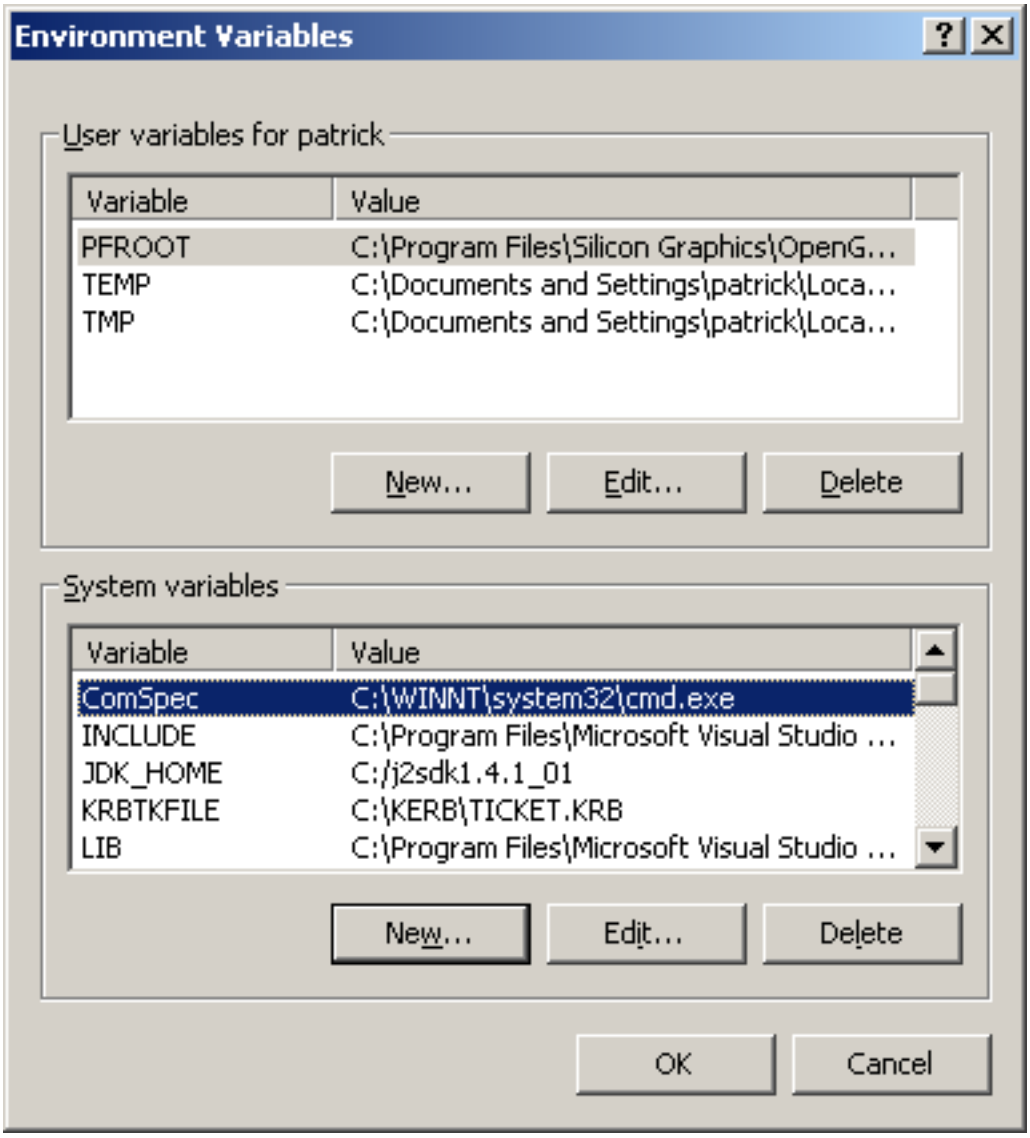


Figure 2.3. Windows® Environment Variable Editor Dialog



Windows NT 4.0

In the Control Panel, open the System icon. The window that is opened has a tab labeled Environment Variables. Here, you can set variables for yourself and, if you have the access privileges, for all users. The GUI is similar to that shown above for Windows® 2000.

Mac OS X

On Mac OS X, environment variables can be set in two different ways, just as on Windows®. They can be set as “global” environment variables available to all applications launched from the Finder, or they can be set within a Terminal window for use within that shell and by all applications launched from that shell. Refer to Apple's Technical Q&A QA1067 [http://developer.apple.com/qa/qa2001/qa1067.html] document for details on how to set global environment variables for use by the Mac OS X desktop interface. To set an environment variable within a Terminal window, refer to either the section called “C-Style Shells (csh, tcsh)” or the section called “sh-Derived Shells (sh, ksh, bash, zsh, etc.)” depending on the user's chosen shell (the default is tcsh).

Syntax Used in this Document

To avoid tying this documentation to a single style of environment variable creation, assignment and reference, the following syntax will be used exclusively from this point onward. Please read this carefully before proceeding.

Naming Environment Variables

When naming an environment variable in the plain text of this document, the variable will be referred to by its name only. For example, to talk about the environment variable containing your path, we will talk about it as PATH.

Creating/Setting Environment Variables

The syntax to set an environment variable is:

```
% <VARIABLE_NAME> = <value>
```

Setting an environment variable also creates it if it is not already present in the current shell's environment.

Printing an the Value of an Environment Variable

Printing an environment variable's value to standard output (stdout) is done as follows:

```
% echo $VARIABLE_NAME
value
```

Referring to the Value of an Environment Variable

To get the value of an environment variable when it needs to be expanded, the following syntax will be used:

```
% cd $VARIABLE_NAME/bin
```

Here, the reference to the value is $VARIABLE_NAME.

Relevant Environment Variables

LD_LIBRARY_PATH (UNIX/Linux only), DYLD_LIBRARY_PATH (Mac OS X only)	UNIX/Linux systems use these environment variables to find dynamically loaded libraries, such as `libvrj-2_2.so`. Unless you are building everything with static libraries, you will need to set these to include the VR Juggler library directory. An example of setting the library path is as follows: `% LD_LIBRARY_PATH = $VJ_BASE_DIR/lib` This is only needed if the Juggler shared libraries are not found by the runtime loader by default. If the Juggler modules were installed using package management such as RPM, then it will not be necessary to set LD_LIBRARY_PATH. Setting DYLD_LIBRARY_PATH on Mac OS X will probably be neces-

sary even when using the OS X installer package.

PATH

On Windows®, the `PATH` environment variable is used in the same way that `LD_LIBRARY_PATH` is used on platforms such as Linux. Thus, the `bin` and `lib` subdirectories of the VR Juggler installation need to be in your path in order for the Juggler DLLs and the **vrjconfig** command to be found. For non-Windows platforms, it is a good idea to have the Juggler `bin` directory in your path so that the **vrjconfig** command can be found from the command line.

FLAGPOLL_PATH

To compile any of the sample applications, the directory containing the Juggler `.fpc` files for Flagpoll [https://realityforge.vrsource.org/view/FlagPoll/WebHome] must be able to be found. By default, the **flagpoll** utility searches `/usr/lib/flagpoll`, `/usr/lib64/flagpoll`, `/usr/share/flagpoll`, `/usr/lib/pkgconfig`, `/usr/lib64/pkgconfig`, and `/usr/share/pkgconfig`. It also scans the directories listed in `LD_LIBRARY_PATH` (`DYLD_LIBRARY_PATH` on Mac OS X) for `flagpoll` and `pkgconfig` subdirectories. If you already have `LD_LIBRARY_PATH` (or `DYLD_LIBRARY_PATH`) set correctly or you have Juggler installed in `/usr`, then there is no need to set `FLAGPOLL_PATH`.

```
% PATH = $PATH:$VJ_BASE_DIR/bin
```

Depending on your shell, you may need to run the **rehash** command after executing the above.

Windows users must also include the directories `$VJ_DEPS_DIR/bin`, `$VJ_DEPS_DIR/lib`, and `$VJ_BASE_DIR/lib` in their `PATH` setting. This is so that the DLLs for VR Juggler and its dependencies will be found when an application is executed.

JDK_HOME

The `JDK_HOME` environment variable is required by the script that starts VRJConfig, the VR Juggler configuration program. If Java™ is installed on your system, `JDK_HOME` may already be set. If not, it needs to be set to the base of the Java™ installation.

VJ_BASE_DIR

The environment variable `VJ_BASE_DIR` identifies where VR Juggler is installed. This information is used for the following purposes:

- Visual C++ project files used for building VR Juggler sample applications on Windows® reference `VJ_BASE_DIR` for the header and library search paths. In this context, `VJ_BASE_DIR` is *required*.
- Important data files and plug-ins needed for proper execution are searched for at run time relative to the path identified by `VJ_BASE_DIR`. As of VR Juggler 2.2, this usage of `VJ_BASE_DIR` has been diminished greatly. The remainder of the description of `VJ_BASE_DIR` explains when it may need to be set and when the default will be more than sufficient.

When using a pre-packaged version of VR Juggler for non-

Windows® platforms, the installation path is at the time of package construction. Therefore, there is no need for the Juggler libraries to be told at run time where they are installed. If VR Juggler is being used on a non-Windows® platform and was *not* installed using packaging such as RPM or a Mac OS X package, then `VJ_BASE_DIR` may be necessary to execute applications successfully.

With the release of VR Juggler 2.2.1, it should not be necessary to set `VJ_BASE_DIR` on any platform in order to *execute* applications. Compiling applications, however, may require the use of `VJ_BASE_DIR`. This varies from application to application.

The reduced need for `VJ_BASE_DIR` is possible because the Juggler shared libraries determine on their own where they are installed when they are attached to the application process. As such, they will automatically set the environment variable(s) that they need to function properly. If an environment variable is already set, then its setting will *not* be changed by the shared library when it is attached to the application process.

Tip

If problems occur at run time with `.jdef` files or plug-ins failing to be found, then setting `VJ_BASE_DIR` will almost certainly fix the problems. If the Juggler installation is corrupted or non-standard (e.g., spread out across several directory trees), then using `VJ_BASE_DIR` in conjunction with some of the environment variables described below will probably prove helpful.

If `VJ_BASE_DIR` needs to be set, then it must be set to the base directory of the installed Juggler Suite. For example, if you downloaded a UNIX version of VR Juggler 2.2 and extracted it to the directory `/home/software`, you would set `VJ_BASE_DIR` with this command:

```
% VJ_BASE_DIR = /home/software/vrjuggler-2.2
```

The last component of the path depends on the particular version of Juggler you have downloaded.

If you downloaded and built VR Juggler from the source code, the compilation creates a directory called `instlinks` which can be used as a VR Juggler base:

```
% VJ_BASE_DIR = $HOME/juggler/my_build_dir/instlinks
```

VJ_DATA_DIR

On non-Windows® platforms, VR Juggler 2.2 and beyond used versioned directories to allow for parallel installations of different versions of VR Juggler. The directories that are versioned are those containing the header files (needed for compiling VR Juggler applications), those containing plug-ins, and those containing data files (needed for running VR Juggler applications) and sample programs. For example, we will refer later to the MPApp sample application. Its source code can be found in `$VJ_BASE_DIR/share/vrjuggler-2.2/samples/OGL`

	`/simple/MPApp`. The value of `VJ_DATA_DIR` for VR Juggler 2.2 ought to be `$VJ_BASE_DIR/share/vrjuggler-2.2`. In VR Juggler configuration files, there is sometimes a need to refer to files in the VR Juggler data directories. In order to allow a configuration to be used with multiple VR Juggler versions (which may or may not be possible depending on the config elements in use) or between Windows® and non-Windows® platforms, users can reference the `VJ_DATA_DIR` environment variable. This environment variable is new in VR Juggler 2.2.1. If the environment variable is not set when an application is launched, it will be set automatically relative to the value of `VJ_BASE_DIR`. In general, users should not set `VJ_DATA_DIR` unless they know that the default value is inappropriate.
VJ_DEPS_DIR	In VR Juggler 2.0, the `VJ_DEPS_DIR` environment variable was introduced as a crutch for finding dependencies such as Boost and CppDOM when building and running VR Juggler applications. That was before Flagpoll and RPMs, however. Now, the recommended usage of VR Juggler is to use Flagpoll for compiling applications and to install everything needed using package management. Flagpoll can find dependencies on its own, and the `FLAGPOLL_PATH` environment variable (see above) extends the search path when necessary. Thus, on platforms where Flagpoll is being used, `VJ_DEPS_DIR` plays no role. This does not currently apply to Windows®, unfortunately. Until Flagpoll is ready for usage on Windows®, the primary build mechanism will continue to be Visual Studio[1]. To make things simple in Visual C++ project files, environment variables can hide path details. The `VJ_DEPS_DIR` environment variable provides the path to the complete set of bundled VR Juggler dependencies (Boost, CppDOM, OpenAL, etc.). It is used by the Visual C++ project files that come with the sample applications. If you downloaded the dependencies as a separate package, set this environment variable to the path where that package was installed. If the dependencies are bundled in the same tree as VR Juggler, then this environment variable does not have to be set.
VJ_DRAW_THREAD_AFFINITY	VR Juggler 2.2.1 introduced the ability for Linux and IRIX users to direct processor affinity for render threads created by the OpenGL Draw Manager. The default way of doing this is to create a multi-pipe/multi-threaded configuration and then set the environment variable `VJ_DRAW_THREAD_AFFINITY` to be a space-separated list of integers identifying the processors in the machine. The optimal setting for this environment variable is likely to vary with hardware and with applications. If more render threads are created than there are processors available, the OpenGL Draw Manager starts over at the beginning of the processor ID list.

Tip

With a multi-processor, multi-core computer, it will generally be best to separate rener threads by processor first and by core second. For example, on a dual processor, dual core machine, a setting of "0 2 1 3" is likely to give good results. In this case, render thread 0 will be assigned to processor 0, render thread 1 to processor 2, and so on.

[1] Visual Studio will probably always be the recommended tool for compiling VR Juggler applications on Windows® even after Flagpoll has Windows® support. The idea is to support *both* Visual Studio and Flagpoll on Windows®.

Note

It is possible to use a value less than zero for a processor ID when setting this environment variable. Doing so will disable processor affinity for that particular rendering thread.

Be aware that the use of this environment variable is optional and that application programmers are free to define their own render thread affinity algorithm that may not use this environment variable at all. For more details on how to do this, refer to the VR Juggler *Programmer's Guide*.

VJ_CFG_PATH

This variable provides a search path for looking up configuration files. It lists one or more directories where VR Juggler configuration files may be found. At run time, this path will be used to find configuration files that are not named using absolute paths. This variable is set using a platform-specific format. On Windows, DOS paths should be used, and they must be separated by the semi-colon (;) character. On UNIX variants and Mac OS X, the paths should be separated by the colon (:) character. This is exactly the way that the `$PATH` environment variable would be set on all of these platforms. If not set, the default search path for configuration files is `$VJ_DATA_DIR/data/configFiles`.

Note

The configuration files are loaded by the module JCCL, and it will recognize the environment variable `JCCL_CFG_PATH`. If `JCCL_CFG_PATH` is set, it takes precedence over `VJ_CFG_PATH`.

JCCL_DEFINITION_PATH

This variable *augments* the search path for JCCL definition files (those files with the extension `.jdef`). It is set using a platform-specific format. On Windows, DOS paths should be used, and they must be separated by the semi-colon (;) character. On UNIX variants and Mac OS X, the paths should be separated by the colon (:) character. This is exactly the way that the `PATH` environment variable would be set on these platforms.

The default search path for configuration files is always `$VJ_DATA_DIR/data/definitions`. Setting the environment variable `JCCL_DEFINITION_PATH` appends directories to the default search path. It is not possible to change the default search path without changing the value of `VJ_BASE_DIR`.

Python programmers will find this sort of behavior familiar. This functionality is based on the way that the Python environment variables `PYTHONHOME` and `PYTHONPATH` behave. In this case, `PYTHONHOME` corresponds to `VJ_BASE_DIR`, and `PYTHONPATH` corresponds with `JCCL_DEFINITION_PATH`. Indeed, the behavior of `JCCL_DEFINITION_PATH` was inspired by the handling of `PYTHONPATH` by the Python interpreter.

VPR_DEBUG_NFY_LEVEL

This variable can be used to control the amount of diagnostic information a VR Juggler application outputs. Its value is a number between 0 (only very important messages are printed) and 7 (vast amounts of data) inclusive. Non-hackers are advised to use levels

0 through 3, as higher debug levels become increasingly cryptic and *can severely impact application performance*. The default is level 1—only errors and critical information are output. An example of setting a value for this variable is:

```
% VPR_DEBUG_NFY_LEVEL = 3
```

VPR_DEBUG_ALLOW_CATEGORIES

This variable can be used to control which components of VR Juggler are allowed to output diagnostic data. If for some reason you set `VPR_DEBUG_NFY_LEVEL` to 5 or higher, this variable can be used to filter the output. The value of `VPR_DEBUG_CATEGORIES` is a space-separated list of Juggler debug component names (defined in `$VJ_BASE_DIR/include/vrj/Util/Debug.h`, `$VJ_BASE_DIR/include/vpr/Util/Debug.h`, `$VJ_BASE_DIR/include/tweek/Util/Debug.h`, `$VJ_BASE_DIR/include/jccl/Util/Debug.h`, and `$VJ_BASE_DIR/include/gadget/Util/Debug.h`). The default value is “DBG_ALL”, which performs no filtering whatsoever. Examples of setting it are as follows:

```
% VPR_DEBUG_ALLOW_CATEGORIES = DBG_ERROR
% VPR_DEBUG_ALLOW_CATEGORIES = "DBG_KERNEL DBG_INPUT_MGR
% VPR_DEBUG_ALLOW_CATEGORIES = "DBG_CONFIG DBG_RECONFIGUR
```

VPR_DEBUG_DISALLOW_CATEGORIES

This variable is basically the opposite of `VPR_DEBUG_ALLOW_CATEGORIES`. Instead of specifying which debugging categories you want to see, you specify which ones you *do not* want to see. Its default value is empty which means that no debugging categories are excluded. Examples of setting it are as follows:

```
% VPR_DEBUG_DISALLOW_CATEGORIES = DBG_ERROR
% VPR_DEBUG_DISALLOW_CATEGORIES = "DBG_KERNEL DBG_INPUT_M
% VPR_DEBUG_DISALLOW_CATEGORIES = "DBG_CONFIG DBG_RECONFI
```

NO_RTRC_PLUGIN

Setting this environment variable to any value will prevent the JCCL [http://www.vrjuggler.org/jccl/] Config Manager from attempting to load the remote run-time reconfiguration plug-in. This plug-in is used to allow VRJConfig to connect to a running VR Juggler application so that the application may be reconfigured on the fly.

Setting this environment variable is useful in two scenarios: when using SPROC threads on IRIX and when using PyJuggler [http://www.vrjuggler.org/pyjuggler/] on Mac OS X 10.3 (“Panther”). The current implementation of the remote run-time reconfiguration plug-in is based on omniORB [http://omniorb.sourceforge.net/], which cannot be used with SPROC threads. In that case, there will not be a plug-in available to load, and setting this environment variable ultimately prevents the IRIX run-time loader from printing a nasty message saying as much. On Mac OS X 10.3, static data initialization in omniORB fails when the omniORB libraries are loaded into the Python interpreter application space, and this leads to a crash. Setting this environment variable allows PyJuggler applications to run cor-

	rectly. This crash does not occur with PyJuggler on Mac OS X 10.4 (“Tiger”).
NO_PERF_PLUGIN	Setting this environment variable to any value will prevent the VR Juggler Performance Mediator from attempting to load the remote performance visualization plug-in. This plug-in is used to allow a JavaBean loaded by the Tweek [http://www.vrjuggler.org/tweek/] Java GUI to connect to a running VR Juggler application and display live performance metrics. Setting this environment variable is useful in two scenarios: when using SPROC threads on IRIX and when using PyJuggler [http://www.vrjuggler.org/pyjuggler/] on Mac OS X 10.3 (“Panther”). The current implementation of the remote performance visualization plug-in is based on omniORB [http://omniorb.sourceforge.net/], which cannot be used with SPROC threads. In that case, there will not be a plug-in available to load, and setting this environment variable ultimately prevents the IRIX run-time loader from printing a nasty message saying as much. On Mac OS X 10.3, static data initialization in omniORB fails when the omniORB libraries are loaded into the Python interpreter application space, and this leads to a crash. Setting this environment variable allows PyJuggler applications to run correctly. This crash does not occur with PyJuggler on Mac OS X 10.4 (“Tiger”).

Chapter 3. VR Juggler Sample Applications

VR Juggler comes with several sample applications in its `samples` directory tree. Many of them are very simple and are designed to demonstrate a specific feature of VR Juggler or a technique to use when writing your own applications. This chapter lists the current sample applications as of this writing and gives a quick description of what you as a potential developer might find interesting in the code. Those users who just want to run applications can safely skip this chapter.

Tutorial Applications

Some sample applications designed for getting started with VR Juggler are found in `$VJ_DATA_DIR/samples/OGL/simple`. All of these applications were designed to be used as part of courses teaching people how to write VR Juggler applications using OpenGL. They contain clear comments explaining what the code is doing, and they are intended to be as simple as possible. These tutorials are as follows:

- simpleInput: An application that demonstrates how to get input from devices. No graphics are rendered with this application. It is intended to be a starting point for getting an understanding of how user input is queried.
- SimpleApp: A very simple OpenGL application that draws a small cube in space and draws the coordinate axes for the cube.
- contextApp: An application demonstrating how to use OpenGL display lists in VR Juggler applications. This extends SimpleApp by using a display list to draw a cube and by moving the cube with the wand.
- ConfigApp: A relatively simple application that demonstrates how user-level code can take advantage of the VR Juggler configuration system, JCCL.
- MPApp: A more complex OpenGL application that demonstrates how to do multi-processing in VR Juggler applications. As it exists in its distributed form, no multi-processing is done. A more detailed lesson is available that explains how to extend the application to employ multi-processing techniques.

For a step-by-step lesson how to use these applications to learn VR Juggler application programming, please refer to the *Programmer's Guide*. It contains sections explaining each of the above applications in great detail. Each lesson ends with an exercise where the reader extends the application to include some new functionality.

Advanced OpenGL Performer Applications

Examples of OpenGL Performer applications can be found in `$VJ_DATA_DIR/samples/Pf/advanced`. These are for more advanced developers who are familiar with Performer and some of the more complicated aspects of VR Juggler. There are two main programs to be found there:

- pfNav: A starting point for basic VR Juggler Performer applications that need to load a model and navigate through it. Users implement their application by inheriting from a provided class, `simplePfNav`. This may be a good place for intermediate-level users of OpenGL Performer to start be-

cause `simplePfNav` hides many of the complicated details (which actually makes that class far from simple).

- pfConfigNav: A more advanced example of a VR Juggler Performer application that can be given its model through a VR Juggler configuration element.

Chapter 4. Compiling a VR Juggler Sample Program

Now that you have VR Juggler installed and you have your environment all configured, it is time for the fun to begin. No, seriously. You are now ready to compile and run VR Juggler applications, and that is the whole point, right? This chapter explains how to compile the applications provided in the directory `$VJ_DATA_DIR/samples/OGL/simple`.

Required Reading

Before reading any further, make sure you have already read the instructions on how to install VR Juggler (in Chapter 1, *Installing VR Juggler*) and on how to configure your environment (in Chapter 2, *Environment Variables*). That information will not be repeated, and it is assumed that you already know what we mean by `VJ_BASE_DIR`. You should also have a basic understanding of how **make**(1) works, but in these examples, nothing more will be necessary than typing `make` on the command line. Refer to the **make**(1) manual page for more information about it.

Compiling an Application

There are two ways to compile VR Juggler applications: from the command line or with Microsoft Visual Studio. Compiling an application on the command line requires the use of GNU make (often installed as **gmake**) so that it will work on all supported platforms. Using Microsoft Visual Studio will only work on Windows®.

Compiling from the Command Line

All the sample programs in `$VJ_DATA_DIR/samples` use the same basic steps to compile unless otherwise noted. Always refer to the top of the sample application's `Makefile` for information that may be specific to building that application. In general, though, all applications' makefiles require the GNU version of the **make**(1) utility, sometimes installed as **gmake**.

The example used here will be the MPApp tutorial application found in `$VJ_DATA_DIR/samples/OGL/simple/MPApp`. It is an OpenGL-based application that will compile and run on all platforms supported by VR Juggler. Begin by changing into the directory `$VJ_DATA_DIR/samples/OGL/simple/MPApp` in a command shell.

To compile MPApp, simply enter the following:

```
% gmake
```

On Mac OS X, enter the following:

```
% make bundle
```

The compile process will then begin. As noted above, the use of GNU make is required to use the distributed makefiles. Now that you have a program compiled, it is time to learn how to run it. (Readers who are not using Visual Studio can skip ahead to Chapter 5, *Running a VR Juggler Sample Program*.)

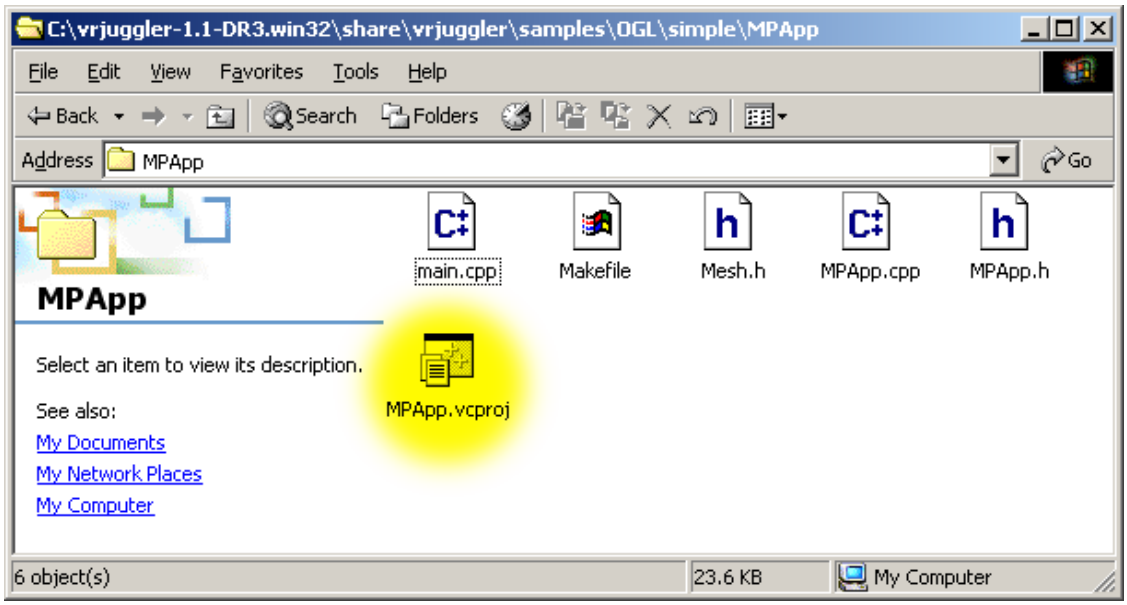
Compiling Using Microsoft Visual Studio

Note

Remember that the Netscape Portable Runtime (NSPR) is *required* to use VR Juggler on Windows. Its DLL directory must be in your path (via the `PATH` environment variable) for proper application execution. The NSPR can be downloaded from the NSPR home page [http://www.mozilla.org/projects/nspr]. If the pre-compiled VR Juggler dependencies are installed, then NSPR is already available.

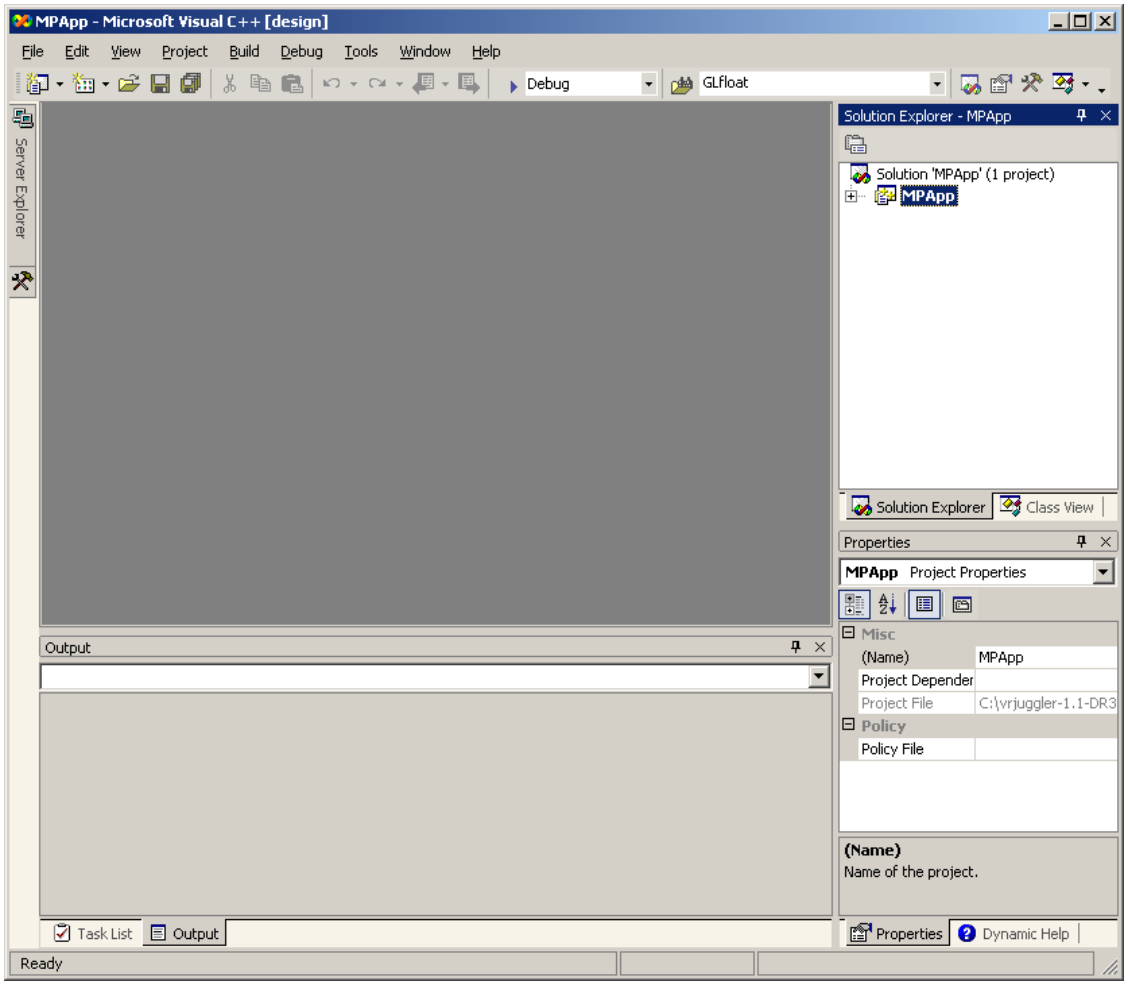
All OpenGL sample applications are shipped with pre-configured Microsoft Visual C++ projects. This is done to help new users get started with compiling VR Juggler applications and to give experienced Visual Studio users a starting place for their application development. To use the workspace for the MPApp application, begin by opening the folder containing the source code and double-clicking on `MPApp.vcproj`.

Figure 4.1. Selecting the Visual C++ Project File



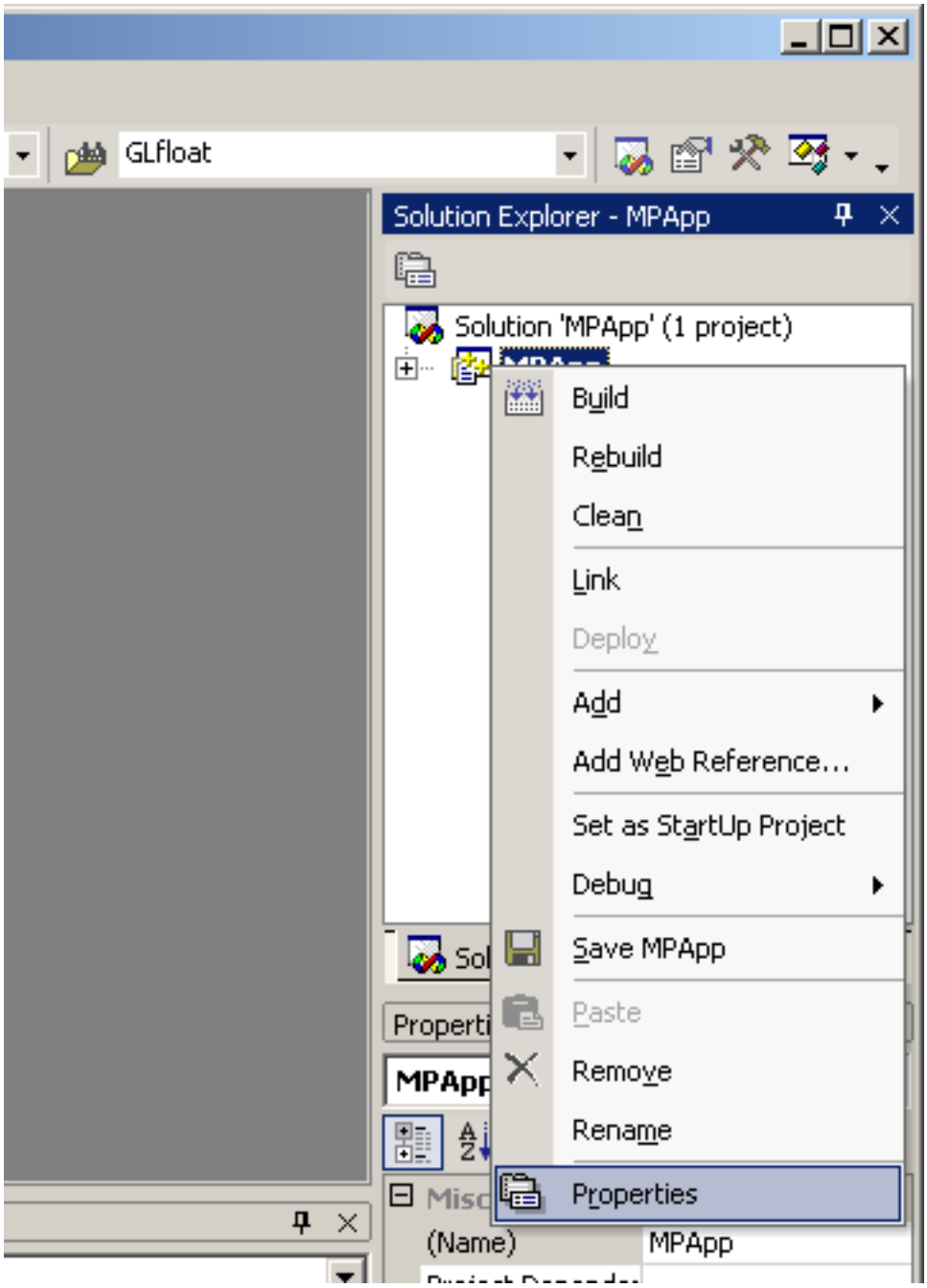
Visual Studio will open, and the MPApp project will be loaded. The unexpanded class view will appear as shown in Figure 4.2, “MPApp Project” when Visual Studio first loads.

Figure 4.2. MPApp Project



In some cases, it may be necessary to change the default project properties. The project properties dialog can be opened in several ways. For example, right-clicking on the project name in the Solution Explorer brings up the menu shown in Figure 4.3, “Project Menu”. We are interested in changing the project's properties, so we select the Properties item from the popup menu.

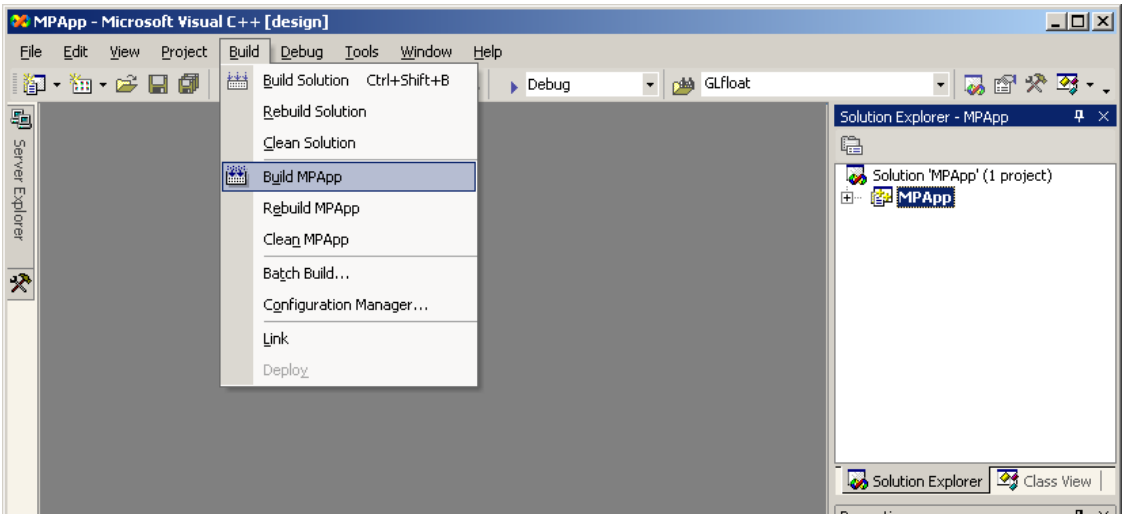
Figure 4.3. Project Menu



Under the MPApp project settings, the path(s) to the VR Juggler C++ dependencies must be filled in. This means setting paths to find headers and libraries. All the Visual C++ project files shipped with VR Juggler refer to the VR Juggler C++ dependency installation via the `VJ_DEPS_DIR` environment variable. If this is not set or cannot be used, the paths must be filled in manually.

Once the program properties are set, compile the application. Under the Build menu, choose the Build MPApp item as shown in Figure 4.4, "Build `MPApp.exe`". Visual C++ will compile the application, and if you have everything configured properly on your computer, the compiling will complete successfully.

Figure 4.4. Build `MPApp.exe`



For the remainder of this book, much of the discussion will concentrate on running applications from the command line rather than from the Visual Studio GUI. Readers can follow whichever method they prefer.

Chapter 5. Running a VR Juggler Sample Program

It is important to note that the same VR Juggler application can be run in a simulator configuration or in a full-scale VR system with no modifications. What does change is the configuration files used when starting the program. In `$VJ_DATA_DIR/data/configFiles`, you can find many basic configuration files including those for running in simulator using a mouse and keyboard to simulate VR input devices and some example files based on those used for the VRAC C4 system. In the directory, you will see some files with names containing "mixin". These are special files that provide a specific capability not necessarily needed by all applications. They can be mixed in (hence the name) with other configuration files as needed. The configuration files found in the `configFiles` directory will be referenced in the examples provided, so be sure you know where they are.

Required Reading

Before reading any further, make sure you have already read the instructions on how to install VR Juggler (see Chapter 1, *Installing VR Juggler*) and on how to configure your environment (see Chapter 2, *Environment Variables*). That information will not be repeated, and it is assumed that you already know what we mean by `VJ_BASE_DIR` and `LD_LIBRARY_PATH`, to name two environment variables. At this point, it is also assumed that you already have compiled an application (MPApp in the case of the examples provided), so you should be sure to have read about how to compile a sample VR Juggler application (in Chapter 4, *Compiling a VR Juggler Sample Program*) before proceeding.

Running an Application with a Simulator Configuration

Running with a simulator configuration means that your input is simulated and your display windows may have limited functionality. (By "simulated input," we mean that input is provided through desktop windows that take keyboard and mouse input and translate that into what would be provided by various types of physical input devices used in VR systems.) Simulator viewports are limited primarily in that they cannot display stereo graphics.

It is important to note that a simulator viewport is a special kind of VR Juggler viewport within a display window. Instead of basing its viewpoint on the head position of one of the users, the viewpoint is controlled by a separate camera that is just another positional device. Within a simulator viewport, VR Juggler draws certain objects to help visualize the environment. For example, the heads of users are represented as blue ellipsoids with gray eyes, and a wand (if present) is drawn as a green pointing device. Besides these common simulator objects, display surfaces can be drawn. These semi-transparent rectangles represent projection screens or HMD viewing projections.

As mentioned, several simulator configuration files are provided with a VR Juggler distribution. These files provide a complete simulation of an immersive environment. Please note that this documentation reflects the state of the configuration files at the time the documentation was written. For more information about the configuration files and how to view or modify the configuration, refer to the *VRJConfig Guide*. (Using VRJConfig is the best way to find out how a specific configuration file is set up.) The configuration files of interest for simulator configurations are as follows:

- `sim.base.jconf` - The basic configuration file used with other simulator configuration "mix-in" files. It defines commonly used VR Juggler concepts that are beyond the scope of this particular book. It also defines simulated head movement using the keyboard. This file also contains the display configuration information needed by other simulator configuration mix-in files. It defines the display

window with its simulator viewport where the rendering occurs.

- `sim.analog.wandmixin.jconf` - A “mix-in” configuration file that defines simulated analog input using the keyboard. This is only required for applications where analog input is used and a physical analog input device needs to be simulated.
- `sim.analog.mixin.jconf` - This version of the analog simulator opens its own window. See the previous file (`sim.analog.wandmixin.jconf`) for other details.
- `sim.c6displays.mixin.jconf` - A “mix-in” configuration file that defines the surfaces of a six-wall CAVE™-like VR system with two additional surfaces at odd angles. Each surface is rendered in a separate display window. This is not required for any application but can be used to test opening multiple display windows (each containing either a surface or a simulator viewport) before running in a multi-pipe VR system. Note, however, that this configuration does not leverage multi-threaded rendering, just multi-window rendering.
- `sim.c6viewports.mixin.jconf` - A “mix-in” configuration file that defines the surfaces of a six-wall CAVE™-like VR system. Each surface is rendered in a separate viewport within a single display window. This is not required for any application but can be used to test using multiple viewports in a single window (each containing either a surface or a simulator viewport).
- `sim.digital.glove.mixin.jconf` - A “mix-in” configuration file that defines simulated digital glove input using the keyboard. This is only required for applications where digital glove input is used and a physical digital input device needs to be simulated.
- `sim.glove.mixin.jconf` - A “mix-in” configuration file that defines simulated gesture-based glove input using the keyboard. This is only required for applications where gesture-based glove input is used and a physical glove needs to be simulated.
- `sim.wand.mixin.jconf` - A “mix-in” configuration file that defines simulated wand input using the mouse. This is only required for applications where wand input is used and needs to be simulated. Wand input is defined as a tracked positional device with several digital buttons.
- `standalone.jconf` - A configuration file that stands on its own and combines the functionality of `sim.base.jconf` and `sim.wand.mixin.jconf`. Note that it uses a single display window for all input.

Starting the Application

Now it is time to run the application—finally! Make sure that all your environment variables are set properly before trying to start the application. Once you are ready, specify the name of the application and all the configuration files it needs. An example of this is:

```
% MPApp sim.base.jconf sim.wand.mixin.jconf
```

On Mac OS X, the command would be the following:

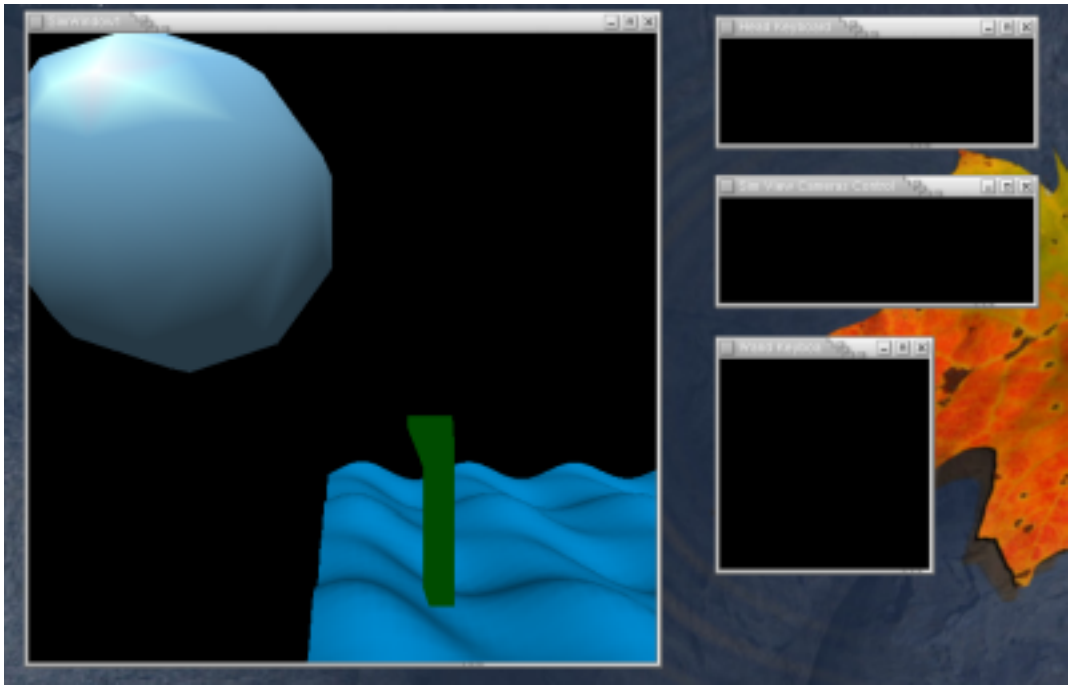
```
% MPApp.app/Contents/MacOS/MPApp sim.base.jconf sim.wand.mixin.jconf
```

Notice that no paths are specified for finding the configuration files. The full paths to the configuration files is not necessary because the default search path will correctly find these files in `$VJ_BASE_DIR\share\vrjuggler\data\configFiles` on Windows or `$VJ_BASE_DIR/share/vrjuggler-2.2/data/configFiles` on all other platforms. Beginning users will typically want to reference the example configuration files in that directory. As you get more comfortable with VR Juggler and its configuration system, you may want to make your own modi-

fied files and put them in the directory `$HOME/.vrjconfig`. The environment variable `VJ_CFG_PATH` is useful in providing a search path for finding your configuration files. (Refer to the section called “Relevant Environment Variables” for more information on using `VJ_CFG_PATH`). To simplify running applications, you may want to make a shell script (or batch file as appropriate) that does all the work of passing configuration files and common command-line arguments.

As the application starts, you will see a status output printed to the console (more or less depending on how you have `VPR_DEBUG_NFY_LEVEL`, `VPR_DEBUG_ALLOW_CATEGORIES`, and `VPR_DEBUG_DISALLOW_CATEGORIES` set), and then one moderately sized simulator display window will open on the left side of your screen while three blank keyboard input windows open on the right side of your screen. The display window will be titled “SimWindow1”, and the keyboard input windows will be titled “Head Keyboard”, “Sim View Cameras Control” and “Wand Keyboard” (in order from the top of the display to the bottom). Do not worry that the keyboard windows are black—that is normal. The display window will have an animated blue mesh, a cyan ellipsoid, and a green pointer. The mesh is what you have come to see; the ellipsoid is the user's head; and the pointer is the user's hand. In Figure 5.1, “MPApp Running on a Linux Desktop with Multiple Input Windows”, we show what this looks on a Red Hat Linux 7.2 desktop for comparison with what you are seeing. Note that the head and wand are only rendered in the simulator windows. They are present because head and wand input are being simulated, and it is typically quite helpful to see the results of that simulated input. To exit the application, press ESC in the window titled “Head Window”.

Figure 5.1. MPApp Running on a Linux Desktop with Multiple Input Windows



With VR Juggler 2.0 and beyond, it is possible to use a single window for graphics and for input. To use such a configuration, execute MPApp as follows:

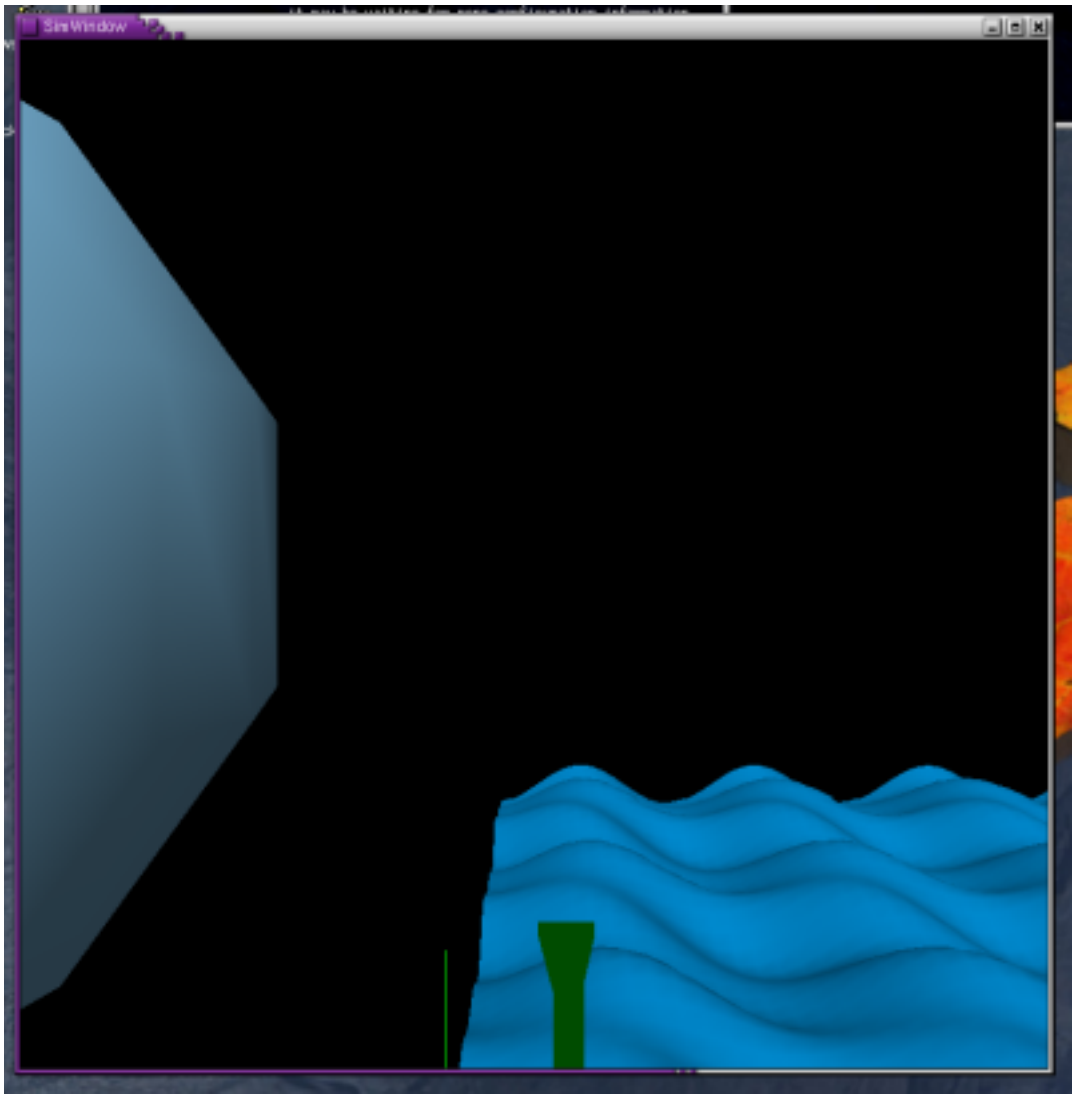
```
% MPApp standalone.jconf
```

On Mac OS X, execute it this way:

```
% MPApp.app/Contents/MacOS/MPApp standalone.jconf
```

This time, only a single window opens, as shown in Figure 5.2, “MPApp Running on a Linux Desktop with One Window”. It shows the same graphics as before, but now it is configured to take keyboard and mouse input. To exit, press ESC in the graphics window.

Figure 5.2. MPApp Running on a Linux Desktop with One Window

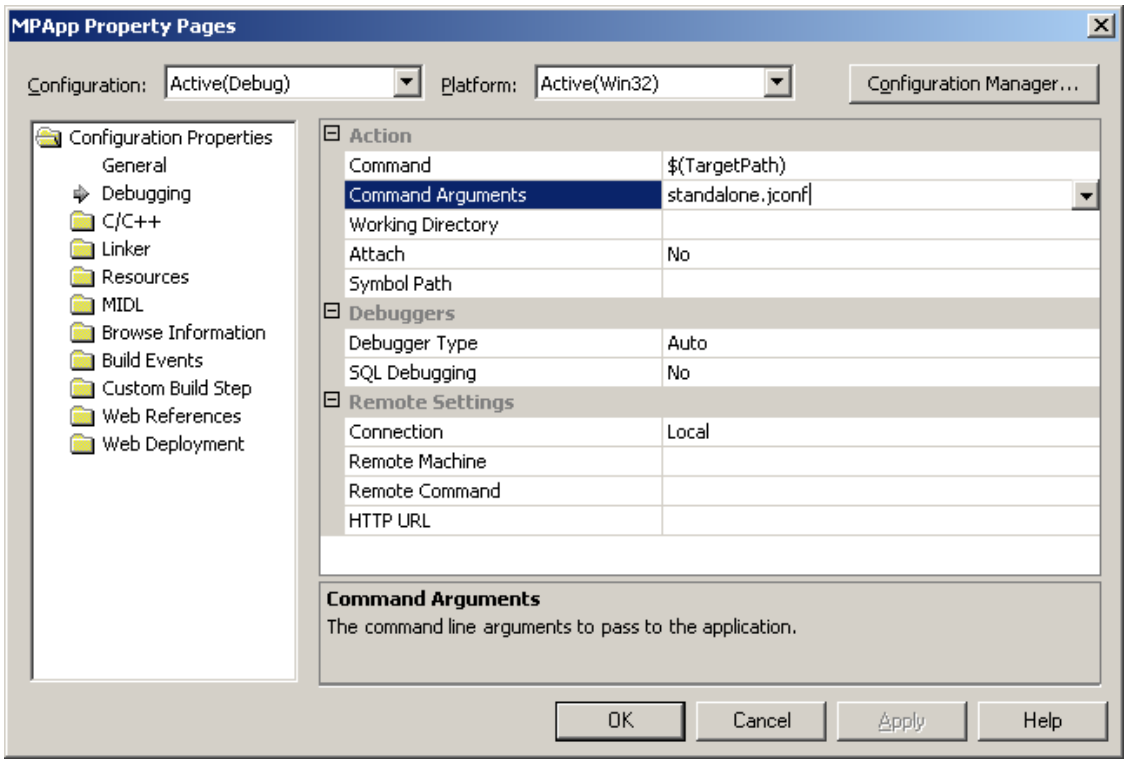


Running an Application on Windows from Within Visual Studio

To run MPApp from within the Visual Studio IDE, the program arguments must be set first. This is done by opening the properties dialog for the project. In this dialog box, choose the Debugging item. There will be an empty text entry field under the heading Command Arguments. Here, enter the full paths to the VR Juggler configuration files that will be used to run the torus application. To use the `VJ_BASE_DIR` environment variable (or any other environment variable), makefile syntax must be used. In other words, to load `somefile.jconf`, use `$(VJ_BASE_DIR)\share\vrjuggler\data\configFiles\somefile.jconf`. As was stated above, the full path need not be specified, so referencing `VJ_BASE_DIR` in the path to the example configuration files will not be necessary in most cases.. In Figure 5.3, “Setting Command Argu-

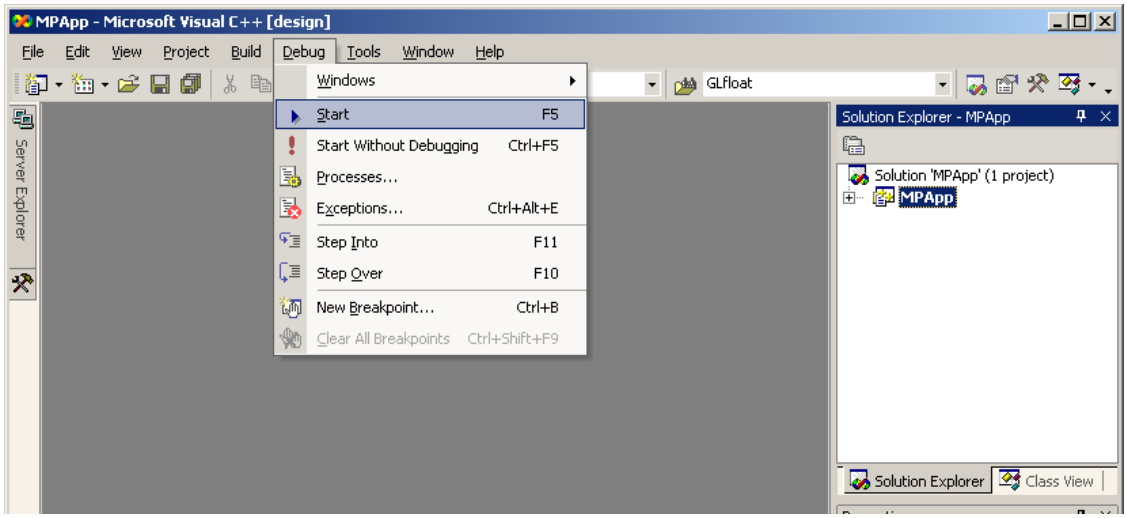
ments", we see the use of `standalone.jconf` as the single command argument to MPApp.

Figure 5.3. Setting Command Arguments



With the application already compiled, execute the MPApp program by choosing the Start item from the Debug menu, shown below in Figure 5.4, "Execute `MPApp.exe`".

Figure 5.4. Execute `MPApp.exe`



Running an Application on Mac OS X with Cocoa Windows

In VR juggler 2.2, Cocoa support has been added for Mac OS X usage. This gives much better results than the previous reliance upon the X Window System. The interface for VR Juggler applications on Mac OS X is noticeably different than other platforms, however. We will explore those differences in this section.

There are two key differences with VR Juggler on Mac OS X from other platforms:

1. Applications are constructed as bundles
2. Every application has the usual Mac OS application menu

The use of application bundles is required for software based on Cocoa. The topic of application bundles is beyond the scope of this document, but Apple has extensive documentation available for perusal. For typical VR Juggler applications, the bundle data files that come with VR Juggler 2.2 will suffice. The application bundle is constructed automatically using these files when the `bundle` target is built for any VR Juggler application whose build is based on Doozer 2.1.4 and newer. For builds that are not using Doozer, see the files in `$VJ_DATA_DIR/data/bundle` for a starting point.

The use of application bundles offers several different ways of launching applications. They are as follows:

- Double-clicking on the application bundle icon in the Finder
- Opening the bundle (e.g., `MPApp.app`) from the command line using **open**
- Executing the contained application directly from the command line

We have seen an example of the third option above, and in this document, we will always show that approach whenever describing how to launch a VR juggler application from the command line.

Using the **open** command offers two different options of its own. It can be used to open the bundle in the same way as double-clicking on its icon in the Finder would work. This usage is shown below:

```
% open MPApp.app
```

The other usage is to open a `.jconf` file and tell the **open** command to use a VR Juggler application bundle to handle the file open operation. This is done as follows:

```
% open -a MPApp.app $VJ_BASE_DIR/share/vrjuggler-2.2/data/configFiles/standalone.j
```

Note that the paths given to the application bundle and to the `.jconf` file must resolve correctly. No path searching will be performed for either in this case.

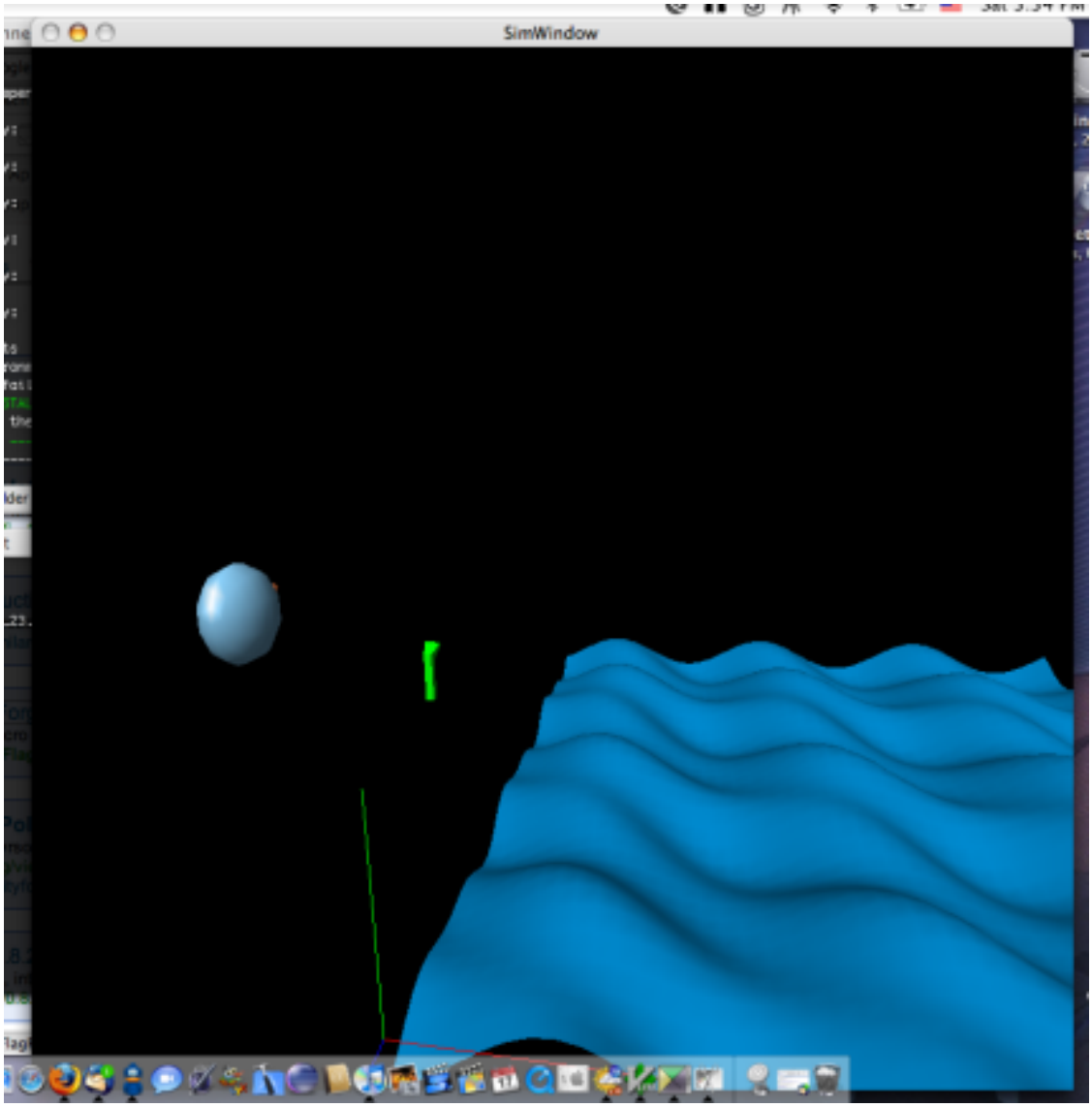
The application menu is defined by the `.nib` contained in the application bundle. This is hooked up to classes in VR Juggler at run time. The application menu is defined with the usual operations for window management as well as the File → Open... (**Cmd+O**) operation for loading `.jconf` files. This feature leverages the existing support for run-time reconfiguration of VR Juggler. Every time a `.jconf` file is opened using the File → Open... (**Cmd+O**) operation, VR Juggler reconfigures itself. This is basically the same as what can be done with VRJConfig when adding new configuration information dynamically. It is not possible to remove configuration information through this interface; VRJConfig must be used for that purpose.

When a VR Juggler application bundle is opened by double-clicking on its icon in the Finder or by using the first usage of the **open** command described above, *no configuration files are loaded*. The File → Open... (**Cmd+O**) operation must be used to load configuration files and configure VR Juggler. (Even remote run-time reconfiguration through VRJConfig may not be available at this point unless omniORB was configured to find a CORBA Naming Service instance through the `OMNIORB_CONFIG` environment variable.)

VR juggler applications on Mac OS X can be exited using the App Menu → Quit (**Cmd+Q**) operation or by using the application shutdown sequence in the same manner as any other platform. The shutdown sequence and the window(s) that accept it depend on the configuration. Usually, this is done by pressing the ESC key in a graphics window or in the input window for the simulated positional device for the user's head.

The Cocoa version of MPApp is shown executing on a Mac OS X desktop in Figure 5.5, “MPApp Running on Mac OS X Using Cocoa with One Window”. Note that in the Dock, the VR Juggler icon is activated. Though it cannot be seen, the application menu is for MPApp.

Figure 5.5. MPApp Running on Mac OS X Using Cocoa with One Window



Running an Application on Mac OS X with the X Window System

Prior to VR Juggler 2.2, using VR Juggler on Mac OS X required the use of the X Window System. VR Juggler 2.2 and beyond support native Cocoa windows, but the X Window System support still exists for those who want to use it (though it is not recommended). This section describes how to run VR Juggler applications on Mac OS X with the X Window System for people who may need to know about this usage. The average Mac OS X user can skip this section since Cocoa will be used instead.

Important

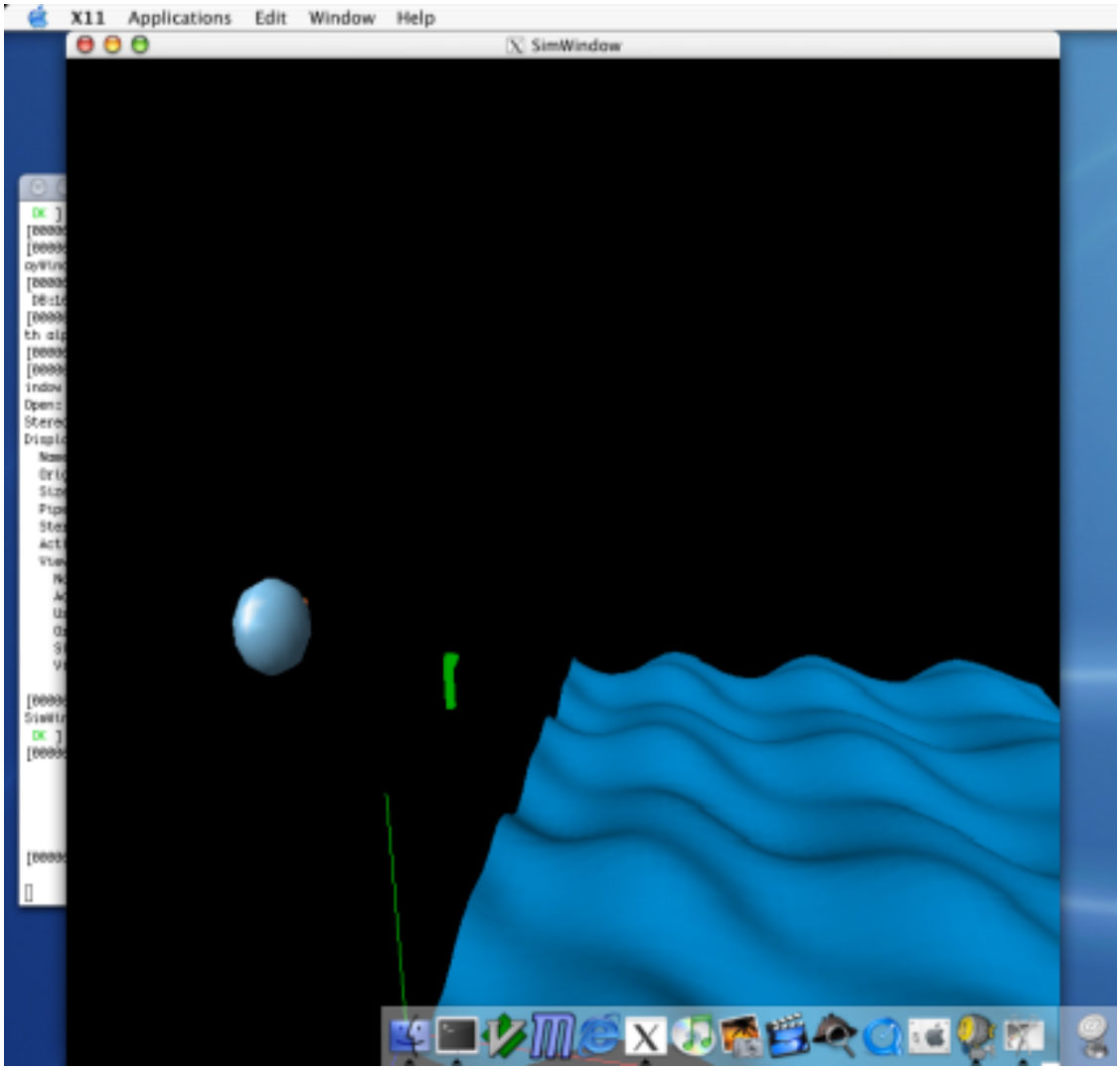
The only way to get full-screen VR juggler windows on Mac OS X is to use the Cocoa windows. If nothing else, this is the biggest single reason not to use the X11 window support on Mac OS X.

Running a VR Juggler application on Mac OS X is slightly different if VR Juggler was compiled against X11 for OS X [http://www.apple.com/macosx/x11/]. Before starting the VR Juggler application, X11 must be running. This can be accomplished by double-clicking on the X11 icon in the Applications

folder. By default, X11 will open a standard xterm when it starts. In this xterm, the `DISPLAY` environment variable will be set correctly, and it is recommended that VR Juggler applications be launched from this xterm. From this xterm, set the necessary environment variables as described earlier in the section called “Relevant Environment Variables”. Once this is done, the application can be executed from the command line just as described in the previous section.

The X11 version of MPApp is shown executing on a Mac OS X desktop in Figure 5.6, “MPApp Running on Mac OS X Using X11 with One Window”. Note that in the Dock, the X11 icon is activated and that the application menu is for an X11 application. This truly is an X11 application running on OS X.

Figure 5.6. MPApp Running on Mac OS X Using X11 with One Window



Basic Desktop Configuration Controls

So now you are probably wondering what you can do with this fancy application. Both of the preceding configurations use the same keyboard/mouse mappings; they vary only in which windows accept the keyboard and mouse input. Using the multi-window configuration, head movement is done with the keyboard in “Head Keyboard”; camera movement is done with the keyboard in “Sim View Cameras Control”; and wand movement is done with the keyboard and mouse in “Wand Keyboard”. Using the single-

window configuration, all input is done with the keyboard and mouse in “Sim Window”. Note, however, that for the single-window configuration, the camera is attached to the user's head for an over-the-shoulder view, and hence, it does not move separately from the head. For information on how to verify these settings and to view the current configuration, refer to the *VRJConfig Guide*. The following list of tables provides all the keyboard and mouse controls for the simulator when using these particular configuration files. Note that it is possible to reconfigure the simulator to suit your preferences. This is provided mainly for those who just want something that works now.

Table 5.1. Moving the simulated head

Transformation	**Key Press**
Move head backward	2 on keypad
Move head left	4 on keypad
Move head right	6 on keypad
Move head forward	8 on keypad
Move head down	7 on keypad
Move head up	9 on keypad
Turn head up	CTRL+2 on keypad
Turn head left	CTRL+4 on keypad
Turn head right	CTRL+6 on keypad
Turn head down	CTRL+8 on keypad
Rotate head clockwise	1 on keypad
Rotate head counter-clockwise	3 on keypad

Table 5.2. Moving the simulated wand

Transformation	**Mouse Input/Key Press**
Move wand backward	ALT+move mouse backward
Move wand forward	ALT+move mouse forward
Move wand left	CTRL+move mouse left
Move wand right	CTRL+move mouse right
Move wand up	CTRL+move mouse forward
Move wand down	CTRL+move mouse backward
Rotate wand left	SHIFT+move mouse left
Rotate wand right	SHIFT+move mouse right
Rotate wand up	SHIFT+move mouse backward
Rotate wand down	SHIFT+move mouse forward
Rotate wand clockwise	Right arrow
Rotate wand counter-clockwise	Left arrow
Wand button #1	Left mouse button
Wand button #2	Middle mouse button
Wand button #3	Right mouse button
Wand button #4	4

Transformation	Mouse Input/Key Press
Wand button #5	5
Wand button #6	6

Table 5.3. Moving the camera (multi-window configuration only)

Transformation	Key Press
Move camera backward	2 on keypad
Move camera left	4 on keypad
Move camera right	6 on keypad
Move camera forward	8 on keypad
Move camera down	7 on keypad
Move camera up	9 on keypad
Turn camera up	CTRL+2 on keypad
Turn camera left	CTRL+4 on keypad
Turn camera right	CTRL+6 on keypad
Turn camera down	CTRL+8 on keypad
Rotate camera clockwise	1 on keypad
Rotate camera counter-clockwise	3 on keypad

Before continuing on to running an application in a full-scale VR system, we provide two asides: using a simulated glove and using a simulated analog device. The examples provided thus far have not discussed this because the information was not relevant to the particular sample application being used. Knowing how to use these simulated devices is important, however, and it is treated separately as a reference for your future endeavors in running VR Juggler applications.

Using a Simulated Glove

If you include the `sim.glove.mixin.jconf` file, your application will also have access to a simulated glove, with position and gesture inputs. The glove is controlled by a window titled “Glove Keyboard”. This window lets you control the glove position and selected gesture. Movement control of the glove uses the mouse and is the same as that of the wand. The mouse buttons are used to select gestures. The mapping of the gesture numbers to actual hand positions is controlled by the “training file” for the simulated glove device. The default training file is `$VJ_DATA_DIR/data/gesture/simpleSimGestures.dat`.

Using a Simulated Analog Device

If you include the `sim.analog.wandmixin.jconf` file, your application will also have access to a set of four analog devices (devices with a value in a range from 0.0 to 1.0). The analog devices are also controlled using the “Wand Keyboard” window which means that their configuration file requires the wand configuration file.

Note

A separate file, `sim.analog.mixin.jconf`, is provided for analog input from a separate simulator window.

The key presses used for controlling the analog devices are listed in Table 5.4, “Analog Device Simulator Keys”.

Table 5.4. Analog Device Simulator Keys

Analog Device Action	Key Press
VJAnalog0 increase	Q
VJAnalog0 decrease	A
VJAnalog1 increase	W
VJAnalog1 decrease	S
VJAnalog2 increase	E
VJAnalog2 decrease	D
VJAnalog3 increase	R
VJAnalog3 decrease	F

Running an Application in a VR System

Running an application full-scale in a VR system tends to be more complicated than running with a simulator configuration. The reason for this is that VR systems tend to differ in configuration and in available hardware. VR Juggler is flexible enough to handle most any configuration you throw at it, but those configurations need to be put together first. VR systems can be driven by a single, multi-pipe machine or a cluster of computers communicating over a network. VR Juggler supports both, and the details are captured in the configuration.

Example configuration files for different VR system configurations come with VR Juggler. An example of using a single multi-pipe computer is shown for VRAC's C4 system when it was driven by a single multi-pipe SGI Onyx computer. For a cluster, there is a configuration file for VRAC's so-called “ptah cluster” that ran Linux. It should be noted, however, that for any particular VR system, custom configuration files will almost certainly have to be written. The idea behind this section is to provide a basic understanding of what is needed to get started with running in a VR system. Configuration topics are addressed in the VR Juggler *Configuration Guide*.

Single Machine Configuration

The example configuration files in the directory `$VJ_DATA_DIR/data/configFiles` modeled after those used for VRAC's Onyx-powered C4 system are as follows:

- `C4.closed.jconf` - The single configuration file that includes the other configuration files necessary for loading a VR Juggler application in the C4 in its closed configuration with full tracking and stereoscopic graphics.
- `C4.closed.mono.jconf` - The single configuration file that includes the other configuration files necessary for loading a VR Juggler application in the C4 in its closed configuration with full tracking and monoscopic graphics.
- `C4.base.jconf` - The basic configuration file needed by all applications when run in the C4. It defines commonly used VR Juggler concepts that are beyond the scope of this particular book.
- `C4.displays.closed.jconf` - The basic display configuration file needed to run with all four walls active and rendering stereoscopic graphics. This defines only the four surface displays to be

opened. The corners are configured for the closed position of the movable walls.

- `C4.displays.closed.mono.jconf` - The basic display configuration file needed to run with all four walls active and rendering monoscopic graphics. This defines only the four surface displays to be opened. The corners are configured for the closed position of the movable walls.
- `C4.mstar.jconf` - The Ascension MotionStar configuration file that defines which bird provides input for the head and for the wand.
- `C4.rfwand.jconf` - The radio frequency wireless mouse that acts as a wand.

Running the application is the same as with a simulator configuration except that the configuration files given on the command line are different. For example, to run MPApp in the C4 with stereoscopic graphics, the following command would be used:

```
% MPApp C4.closed.jconf
```

On Mac OS X, the following command would be used:

```
% MPApp.app/Contents/MacOS/MPApp C4.closed.jconf
```

Cluster Configuration

Clusters involve multiple machines, and running VR Juggler on a cluster means running the application on every node of that cluster. Every node must load the same configuration whether it is defined in multiple files or in a single file. VR Juggler (more specifically, Gadgeteer) sorts out which parts of the configuration apply to each node[1].

The way that any given cluster-ready VR Juggler application works is up to the people who created the application. The sample applications that come with VR Juggler all work the same way. Namely, the same application is run on every node with the same set of command line options. Other applications may use varying command line options depending on the role of the node in the cluster, or different applications may be run on each node, again depending on the role of the node.

For MPApp, the usage on a cluster is the same as what we have shown previously. The only things that change are the configuration files that are loaded and the execution of the **MPApp** binary on *all nodes of the cluster*. If we were to use the example configuration `ptah.cluster.jconf`, the usage on all nodes would be as follows:

```
% MPApp ptah.cluster.jconf
```

On Mac OS X, the following command would be used:

```
% MPApp.app/Contents/MacOS/MPApp ptah.cluster.jconf
```

[1]The details of how to configure a cluster are beyond the scope of this document; readers are directed to the VR Juggler *Configuration Guide*.

Appendix A. GNU Free Documentation License

Version 1.2, November 2002

FSF Copyright note

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sec-

tions then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you

as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

GNU FDL Modification Conditions

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the

title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the

combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> **Sample Invariant Sections list**
>
> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> **Sample Invariant Sections list**
>
> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.