

# **VR Juggler**

## **VjControl Guide**

---

# VR Juggler: VjControl Guide

Published \$Date: 2003/03/12 20:53:27 \$

---

---

---

# Table of Contents

I. Configuration Basics .....	1
1. Overview of VR Juggler Configuration Files .....	3
Basic information .....	3
Loading configuration files .....	3
Using modular configuration files .....	4
Referencing other configuration files .....	4
2. Introduction to Config Chunks .....	5
What are Config Chunks? .....	5
Chunk Description Files .....	6
What is a Chunk Description? .....	7
3. Configuring VR Juggler .....	9
Meta Chunks .....	9
Display Chunks .....	9
Input .....	9
Environment Manager .....	11
Miscellaneous .....	11
VjControl-specific chunks .....	12
II. Installing and Using VjControl .....	13
4. About VjControl .....	15
5. Getting Started with VjControl .....	16
System Requirements .....	16
Setup .....	16
Starting VjControl .....	16
The Main Window .....	16
Loading Files .....	17
6. Command-Line Arguments .....	18
Networking Options .....	18
File Loading Options .....	18
7. Working with Config Files .....	19
The ConfigChunk Editing Window .....	19
Including Other Configuration Files .....	21
8. Working with Chunk Description Files .....	22
The ChunkDesc Editing Window .....	22
Labeling Individual Values of a Property .....	23
Properties with Enumerations .....	23
Properties that Reference or Embed ConfigChunks .....	23
Using DescHelp files .....	23
9. Working with Chunk Organization Trees .....	25
Missing Functionality .....	26
10. VjControl Preferences .....	27
VjControl Preference Options .....	27
11. Dynamically Reconfiguring VR Juggler Applications .....	29
Connecting to a VR Juggler application .....	29
Viewing and Changing the Active Configuration .....	29
III. Technical Specification .....	30
12. Config File Format .....	32
Overview .....	32
General Rules .....	32
Config Files .....	33
ConfigChunks .....	33
Property Lines .....	33
Properties with Enumerations .....	34
Example Property Lines .....	34

13. ConfigDesc File Format .....	35
Overview .....	35
General Rules .....	35
ChunkDesc Files .....	36
ChunkDescs .....	36
Property Description Lines .....	36
14. ChunkOrgTree File Format .....	38

---

## List of Figures

2.1. Example Config Chunk .....	5
5.1. VjControl Tabs .....	16
7.1. VjControl Config Editing .....	
7.2. Chunk Removal Menu .....	
7.3. Chunk Insertion Menu .....	
7.4. Config Chunk Editing Window .....	19
7.5. Property Editors .....	20
8.1. Descriptions Panel .....	
8.2. Chunk Description Editor .....	
8.3. Property Label Editor .....	
9.1. OrgTree Example 1 .....	
9.2. OrgTree Example 2 .....	
9.3. OrgTree Editor Panel .....	
9.4. ChunkDesc Add Menu .....	25
11.1. Connection Panel .....	

---

# **Part I. Configuration Basics**

---

---

# Table of Contents

1. Overview of VR Juggler Configuration Files .....	3
Basic information .....	3
Loading configuration files .....	3
Using modular configuration files .....	4
Referencing other configuration files .....	4
2. Introduction to Config Chunks .....	5
What are Config Chunks? .....	5
Chunk Description Files .....	6
What is a Chunk Description? .....	7
3. Configuring VR Juggler .....	9
Meta Chunks .....	9
Display Chunks .....	9
Input .....	9
Environment Manager .....	11
Miscellaneous .....	11
VjControl-specific chunks .....	12

---

# Chapter 1. Overview of VR Juggler Configuration Files

This section describes the basics of VR Juggler configuration files: what they are, how to use them, and what they are made of.

## Basic information

Configuring VR software for a particular hardware system can be a daunting task due to the complexity and diversity of the environments and hardware in use. In addition to this, many users will need to switch between immersive VR hardware and desktop simulators during development, or between one hardware system and another.

VR Juggler's inherent flexibility increases this complexity. VR Juggler components such as displays and device drivers can be combined in many ways. For example, a configuration could mix simulated and real input devices. Or, hardware permitting, VR Juggler could use two separate head-mounted displays (HMDs) simultaneously. Input devices can be chained together, manipulated in a variety of ways, and mapped to different names. The possibilities are endless - but so are the potential pitfalls.

VR Juggler's configuration system was designed to simplify this task as much as possible. While VR Juggler uses plain text configuration files, these files are structured and organized. VR Juggler's config files can be written in a modular fashion, so that different components or hardware features can be easily mixed together. Finally, VR Juggler provides a GUI tool, VjControl, for editing configuration files. The GUI can prevent many kinds of errors and make the user's task easier by presenting the available options, valid arguments, help information, and so on.

## Loading configuration files

When a VR Juggler application is initialized, it creates an internal database of configuration information. This database will contain information about the kind of IO devices available, the display windows VR Juggler should open, and lots of other stuff.

Typically, this database will be filled up by reading one or more configuration files. With many VR Juggler applications, the names of config files to load are given as command-line arguments when the program is run. For example:

```
% cubes ~/.vjconfig/simstandalone.config
```

runs the cubesdemo program and tells it to load a configuration file called simstandalone.config, which provides a basic simulator-mode configuration for running applications.

Configuration files can be stored in several places:

- The VR Juggler distribution contains a number of sample files in `$VJ_BASE_DIR/share/Data/configFiles/`. These include a variety of configuration files for simulator-mode use, as well as a number of files demonstrating how VR Juggler can be configured for various kinds of VR hardware.
- Juggler expects you to have a directory called `.vjconfig` in your home directory. This is a good place to put any config files that you've chosen to modify. For example, if you've changed one of the simulator config files because you didn't like the keyboard shortcuts it was using, you should put it in `~/.vjconfig`.
- Your system administrators may have also created a repository of VR Juggler config files set up to work with your local systems and hardware. For example, VRAC keeps local configurations in the directory `home/vr/vjconfig`.

## Using modular configuration files

One of the difficulties of managing a VR system is the rapid proliferation of configuration files. If the entire configuration is read from a single file, then there must exist separate files for every possible permutation of configurations - simulators, projection systems, projection systems with simulated input devices, or configured with monoscopic displays for photo sessions, or monoscopic displays *and* simulated input devices, etc. Keeping all these files up-to-date can become a full-time job.

VR Juggler sidesteps this problem with the concept of *modular configuration files*. Instead of listing a single configuration file on the application's command line, we list several. Each file contains configuration information relevant to a particular part of the system. For example, one could define all the displays to be used, while another could define the input devices, and a third could describe how those inputs are mapped to a user's head, glove, wand, etc. There could be several variations of each file to choose from: display configs for an HMD, simulator window, desk, monoscopic desk, etc.; input config files for several varieties of hardware, or the user's preferred simulator setup. So long as a consistent set of names are used for input devices and so on, these files could be freely mixed and matched, eliminating the need to have a separate file for every possible combination.

The VR Juggler distribution includes several examples of modular configuration files in the directory `$VJ_BASE_DIR/share/Data/configFiles`. The `sim.*.config` files can be used in various combinations to run an application with various kinds of simulator devices. For example, a basic simulator configuration with one display window and a wand would use these files:

- `sim.base.config`
- `sim.displays.config`
- `sim.wand.mixin.config`

Other files could be used to change the basic configuration. For example, to use a simulated glove instead of a wand, the user would replace `sim.wand.mixin.config` with `sim.glove.mixin.config`.

## Referencing other configuration files

While modular configuration files provide a lot of flexibility, users will not want to list four or five separate configuration files on the command line every time they run a program. Typically, a small set of combinations will see the majority of use. For example, a fully simulator configuration will be used heavily (but each developer is likely to have a slightly different idea of the *preferred* simulator configuration).

To provide ease of use - referencing a complete configuration with a single name, while avoiding proliferation - VR Juggler config files feature an include directive, similar to the `#include` preprocessor directive of C or C++. Thus, a user can create a config file which simply references several files in the system repository. He only needs to keep track of the single file's name, but when any of the included files are updated, he automatically sees the changes.

---

# Chapter 2. Introduction to Config Chunks

This chapter provides a very basic introduction to Config Chunks, the building blocks of VR Juggler configuration files. The first part of this chapter is essential to understanding how to modify VR Juggler configurations and how to use the VjControl program. The later sections, discussing Chunk Descriptions, are primarily of interest to developers attempting to expand VR Juggler's configuration system to support new devices or adding dynamic reconfiguration capabilities to application code.

## What are Config Chunks?

The building blocks of VR Juggler config files are pieces of data called *ConfigChunks*, or “Chunks” for short. There are many different kinds of Chunks, and each one contains the complete configuration information for a particular component of VR Juggler. These components can be things like display windows, input device drivers, the VR Juggler performance measurement system, and so on. To give you a better feel for the concept, here's a list of some of the kinds of chunks used by VR Juggler 1.0:

- *Sim Display* - Simulator-mode display window
- *Surface Display* - Projection surface display window
- *IBox* - Config for Immersive Ibox driver
- *Flock* - Config for Ascension Flock of Birds tracking system driver
- *Position Proxy* - Proxy for a positional device (e.g. the Flock)
- *Performer API* - Setup info for Iris Performer-based applications

When VR Juggler reads a configuration file, it compiles a list of all the ConfigChunks in the file and then tries to find a VR Juggler component that knows how to deal with it. For example, a Flock of Birds Chunk would be passed to the Input Manager, which would create and activate an instance of VR Juggler's Flock of Birds driver.

So what's inside a ConfigChunk? Lots of stuff - and lots of *different* stuff, depending on what kind of Chunk it is. After all, the information the Input Manager needs to set up an input device doesn't have a lot in common with what the Display Manager needs to open up a display window.

The one thing that all Chunks have in common is that they all have names. Names are used to distinguish between multiple Chunks of the same type. For example, a configuration file for a CAVE-like device might define several Surface Display Chunks, with names like "Front Wall", "Left Wall", "Floor", and so on. It's important that Chunks have unique names. If VR Juggler reads two ConfigChunks that have the same name, it will assume that the second one is a redefinition or reconfiguration of the first one, and it will forget the first one and just use the second definition.

The rest of the Chunk consists of a list of Properties. Properties are individual pieces information. Continuing our example with a Flock of Birds tracking system, the Properties would include things like the name of the serial port to connect to, the baud rate to use, the positional offset, and so on. To illustrate, here's what it looks like when you view a single ConfigChunk in the VjControl program:

### Figure 2.1. Example Config Chunk

The screenshot shows a configuration window titled "Configuration for Ascension Flock of Birds". At the top, the "Instance Name" is set to "Flock\_o\_Birds". Below this, several properties are listed in a scrollable area:

- Port (String):** /dev/ttyd3. Description: Serial port the Flock is attached to.
- Baud Rate (Int):** 38400. Description: Serial port speed.
- Num Birds (Int):** 2. Description: Number of birds in the Flock.
- Translation:** X(Float) -0.020833, Y(Float) 7.447917, Z(Float) 1.317708. Description: Translation from tracker coordinates to Juggler coordinate.
- Rotation:** X(Float) 90.0, Y(Float) 0.0, Z(Float) 180.0. Description: Rotation from tracker coordinates to Juggler coordinate s'.
- Calibration file (String):** /home/ucore/jolank/Source/...

At the bottom of the window are three buttons: "OK", "Cancel", and "Help".

"Instance Name" is the name of the Chunk - in this case, "flock\_o\_birds", while its type is "Flock" (the type is usually show in the window's title bar). Below that are the various properties - "Baud Rate", "Port", etc. Different properties take different kinds of values - Port takes a string, while Baud Rate takes an integer. The Translation property is interesting because it takes three separate floating point numbers. There are also Boolean properties and properties whose values are the names of other Chunks (so that you can have Chunks that refer to other Chunks).

## Chunk Description Files

For every kind of ConfigChunk there's an object called a *ChunkDesc* - short for "Chunk Description". ChunkDescs live in ChunkDesc files - and where are they? The primary VR Juggler ChunkDesc file is `$VJ_BASE_DIR/share/Data/chunksDesc` - it defines all the different kinds of chunks that VR Juggler beta1 knows about and needs. When a VR Juggler program starts up, one of the first things it does - even before it reads its config files - is to read the primary chunksDesc file.

Later in the beta series, VR Juggler will have the ability to load user-specific ChunkDesc files. These might include

ChunkDescs for application-specific ConfigChunks.

If you've read this far, you know what VR Juggler's config files are, and you know what they're made of - lists of ConfigChunks. You also know that there are a lot of different kinds of ConfigChunks. Where do these different kinds of Chunks come from? How does VR Juggler know how to make a "Surface Display" Chunk that's different from a "Position Proxy" Chunk?

## What is a Chunk Description?

So, a particular kind of ConfigChunk is defined by a ChunkDesc. So what's in a ChunkDesc? Information about all the properties in the Chunk, with their names, and types, and number of allowed values. There's also help strings and other information used to make the VjControl program a little more user-friendly.

Here's a breakdown of the contents of a ChunkDesc:

- *Name* - This string is the name of the type of ConfigChunk this description defines. In the examples we've used on this page, this would be something like "Surface Display" or "Flock".
- *Token* - Usually this is a short form of the name - no whitespace allowed. The token is used by the VR Juggler library to refer to the Chunk type. This means that it's fairly safe to change the Name in a ChunkDesc - this will just change the way things are labeled in the VjControl GUI - but changing the token is a bad idea - VR Juggler could get confused and be unable to recognize or load Chunks. So, once you define a token, don't change it unless you're prepared for the consequences.
- *Help* - This is a short (one-line) help text, that should contain a description of what Chunks using this ChunkDesc are used for.
- A list of property descriptions. This list always contains one entry - Name. That's the instance name property of a Chunk using this description, as opposed to the name of the description itself.

Other than that, the properties described will completely depend on what Chunks using this description are trying to do. Here's what's in the description of a property:

- *Name* - The name of a property is simply a text string used to refer to it by the GUI, like "Serial Port" or "X Display Name".
- *Token* - The token is a lot like the name, and they are often the same string. The difference is that the token is used internally, in the config file format, and in the library itself when querying the configuration database. The token is restricted in a way similar to C/C++ variable names: no whitespace is permitted, and it may not begin with a number.
- *Type* - There are four primitive types for properties: strings, integers, floating-point numbers, and booleans. In addition to these four types, Property values can also refer to other ConfigChunks. Example: The ConfigChunk for "general setup" might have a Property that lists the active Displays. The type for this Property could just be a string (matching the name of a Display ConfigChunk), but we can also make the Property type be "name of a Display ConfigChunk". This lets the GUI be a little smarter and easier to use when editing that property - but more on that later.
- *Number* - Most Properties have single values. Other times, as with our "Position Offset" example, a Property may need a fixed number of values. It is also possible for a Property to have a variable number of values; the list of active Displays mentioned above would be a good example. There may be no active displays, there may be 6, or 2, or just 1.
- *Value Lists (optional)* - Sometimes a Property might have a certain number of appropriate choices. For example, the handedness of a glove is either "Left" or "Right", and it would be nice if our GUI could display those as choices to the user, instead of requiring them to type in a string. The configuration system supports

this with enumerations, which are simply lists of the appropriate values for a Property. Enumerations for string Properties are just lists of string values, while enumerations for integer and float Properties map descriptive strings to numeric values.

- *Value Labels (optional)* - Some properties need to have several values - for example three floating point numbers to define a point in space. In this case, we can include value labels - short strings, one for each of the values, such as "X coord", "Y coord", and "Z coord", or "Length" and "Height".

---

# Chapter 3. Configuring VR Juggler

VR Juggler applications are configured by using ConfigChunks. These Chunks can be specified in a configuration file which the application reads at startup, or they can be added to a running application using the VjControl tool.

This page documents particular kinds of ConfigChunks, and how they can be used with VR Juggler applications. This includes adding and configuring input devices, controlling displays, and setting up the VR Juggler simulator mode.

The Chunks in the listing below have been divided into general categories. Each Chunk name is a hyperlink to a detailed description.

For many chunks, it is reasonable and useful to have multiple instances in your configuration (displays are an obvious example). Others are "singletons" - they provide some sort of global information, and it wouldn't make sense to have multiple definitions. Singleton Chunks are noted in the descriptions. If a VR Juggler configuration does contain multiple instances of a singleton Chunk - multiple EnvironmentManager Chunks, for example - the most recently read Chunk will supercede older ConfigChunks.

## “Meta” Chunks

VR Juggler defines two ConfigChunks which don't correspond to components of the VR system itself, but rather impact how configuration files are loaded in the first place.

Config File Include	The Config File Include chunk gives the name of an additional config file which should be loaded whenever this chunk is encountered.
ChunkDesc File Include	The ChunkDesc File Include chunk gives the name of an additional chunk descriptions file which should be loaded whenever this chunk is encountered. ChunkDesc Include Chunks are particularly useful in configuration files that include application-defined ConfigChunk types.

## Display Chunks

VR Juggler gives you a great deal of latitude in defining displays. For example, you can use a combination of surface displays and simulator displays, or maybe just one or more simulator displays. Each display window is configured by a Simulator or Surface Display ConfigChunk. To add a new display window, simply add a new Display ConfigChunk to your configuration file and edit it to taste.

Simulator Display	Opens a display window suitable for simulator-style use. The simulator window uses a "camera" attached to an arbitrary position input, and can graphically show the position of a user's head, wands, gloves, surface displays, and projection frustums.
Surface Display	Opens a display window suitable for use with a projection screen. The dimensions and position of the surface can be freely defined in 3D space.
Display System	This chunk (a singleton) tells VR Juggler about the number of video channels on a machine and how they map to X Window displays.

## Input

Any input devices that are defined in your config file will be initialized at application startup. So, to add a new device, all you have to do is add a ConfigChunk which describes its configuration. This section describes the ConfigChunks for the input devices supported by VR Juggler.

- Input Manager

**Input Manager** This singleton chunk specifies some useful default values for the input devices, such as a default position and orientation of positional devices.

- Input Devices

**Flock of Birds** Configuration for Ascension Flock of Birds tracking system.

**MotionStar** Configuration for Ascension MotionStar wireless tracking system.

**Trackd Controller** Configuration for a Trackd controller memory area.

**Trackd Sensor** Configuration for a Trackd sensor memory area.

**Intersense Tracker** Configuration for Intersense trackers.

**IBox** Configuration for Immersion Corp. IBox.

**CyberGlove** Configuration for a Virtual Technologies CyberGlove. The glove is a gesture device, and can also have a positional device attached to it.

**3D Mouse** Configuration for Logitech 3D Mouse.

**Keyboard Input Window** This device opens a window on a display which accepts keypresses and mouse movement information. It is usually used to provide input for the simulated devices described below.

- Simulator Devices

VR Juggler provides a complete set of simulated input devices for use in testing applications. The simulated devices generally attach to a Keyboard Input Window. The mapping of keypresses to actions is completely configurable.

**Digital Simulator** Provides one or more digital inputs (on/off values).

**Analog Simulator** Provides one or more analog inputs (inputs in a continuous range). Keypresses to move up and down for each device, and the increment for a single keypress, can be configured.

**Position Simulator** Simulates a single positional input (i.e. a tracker). Users can configure the keypresses to use for movement and rotation on each access, delta for movement and rotation, and initial values.

**Glove Gesture Simulator** Simulates a gesture input device (i.e. a glove).

**simKeyboardDigital** This is a rather unusual device. It uses digital input devices to simulate a keyboard - which can be used to drive other simulator devices!

**Dummy Position** A fake positional device, which always returns a constant (configurable) position

and orientation.

- Proxies and Proxy Aliases

Proxies are the mechanism for making access to input devices generic. A Configuration file should include a proxy for every possible input source. Each proxy is defined with the name of the device it accesses and a unit number (for devices that provide multiple inputs).

Digital Proxy	Proxy for digital devices.
Analog Proxy	Proxy for analog devices.
Position Proxy	Proxy for position devices. The Position Proxy also allows you to apply a transformation to the position data returned from the device.
Glove Proxy	Proxy for glove devices.
Gesture Proxy	Proxy for gesture devices.
Keyboard Proxy	Proxy for keyboard devices.
Proxy Alias	Proxy aliases are used to create additional names to refer to proxies.

## Environment Manager

The Environment Manager (EM) is the part of VR Juggler that controls communications with the GUI program, VjControl. It also controls VR Juggler's performance monitoring capabilities.

Environment Manager	The Environment Manager chunk, a singleton, allows the user to configure the network setup of a VR Juggler application (whether to accept connections or not, what network port to listen on, etc).
Performance Measurements	This chunk, also a singleton, allows you to turn VR Juggler's internal performance measurements on and off.
EM Connection	These chunks define data connections - files or sockets - that the EM can connect to. An EM Connection is created automatically whenever a VjControl process connects to a VR Juggler application. Another common use of EM Connections is to define an output file for VR Juggler performance data.

## Miscellaneous

Juggler User	One of the eventual goals of VR Juggler is to provide full support for multiple users. The first step is the Juggler User chunk, which currently associates a positional input proxy with a user's head. You'll need to create at least one of these, to use as a target of your Display chunks.
Performer API	This singleton Chunk defines a few properties needed for using Iris Performer-based VR Juggler applications. If you aren't using Performer, you don't need to worry about

this one.

## VjControl-specific chunks

The VR Juggler control program, VjControl, also uses some ConfigChunks for its own configuration. Generally, these will be kept in a separate file (`$HOME/.vjconfig/vjcontrol.config`).

VjControl Configuration	This chunk stores VjControl's preferences information.
Performance Data Display	Chunks of this type give VjControl information on how to display and interpret performance data gathered from a VR Juggler application or a performance log file.

---

## **Part II. Installing and Using VjControl**

---

---

# Table of Contents

4. About VjControl .....	15
5. Getting Started with VjControl .....	16
System Requirements .....	16
Setup .....	16
Starting VjControl .....	16
The Main Window .....	16
Loading Files .....	17
6. Command-Line Arguments .....	18
Networking Options .....	18
File Loading Options .....	18
7. Working with Config Files .....	19
The ConfigChunk Editing Window .....	19
Including Other Configuration Files .....	21
8. Working with Chunk Description Files .....	22
The ChunkDesc Editing Window .....	22
Labeling Individual Values of a Property .....	23
Properties with Enumerations .....	23
Properties that Reference or Embed ConfigChunks .....	23
Using DescHelp files .....	23
9. Working with Chunk Organization Trees .....	25
Missing Functionality .....	26
10. VjControl Preferences .....	27
VjControl Preference Options .....	27
11. Dynamically Reconfiguring VR Juggler Applications .....	29
Connecting to a VR Juggler application .....	29
Viewing and Changing the Active Configuration .....	29

---

# Chapter 4. About VjControl

VR Juggler is (C) Copyright 1998, 1999, 2000, 2001 by Iowa State University

Original Authors:

Allen Bierbaum, Christopher Just,  
Patrick Hartling, Kevin Meinert,  
Carolina Cruz-Neira, Albert Baker

This library is free software; you can redistribute it and

---

# Chapter 5. Getting Started with VjControl

This section will tell you how to set up and run the VjControl program. It assumes that VR Juggler has already been installed on your system.

## System Requirements

VjControl is a Java application designed to take full advantage of Sun's Java Foundation Classes (the "Swing" classes). The current version requires at least JDK1.2, though JDK1.3 is recommended.

## Setup

Before trying to run VjControl, you should perform the steps discussed under "User Setup" in the VR Juggler *Getting Started Guide*. In particular, make sure all your environment variables are set correctly.

It is important that you create a `~/ .vjconfig` directory, as this is where VjControl stores its own preferences.

You should also find out where the VR Juggler config files you should use are located. The VR Juggler distribution includes a number of sample config files in the directory `$VJ_BASE_DIR/share/Data/configFiles`. Your system administrators may also have a location for customized config files. For ease of use, you may wish to copy commonly-used config files to your `~/ .vjconfig` directory. This is also a good place to keep any config files that you modify to your own tastes.

## Starting VjControl

On Unix/Linux systems, the command is **vjcontrol**; on Win32, the command is **vjcontrol.bat**. If `$VJ_BASE_DIR/bin` is not in your search path, you will need to give the entire path to VjControl on the command line.

VjControl's Preferences [Preferences.html] control what files are loaded automatically on startup. There are also command-line arguments [cmdline.html] that can be used to load files at startup. With the default preferences, VjControl will attempt to load `$VJ_BASE_DIR/share/Data/chunksDesc`, which contains definitions for the basic types of VR Juggler configuration information.

## The Main Window

The main VjControl window consists of a set of panels accessed by a row of tabs (see Figure 5.1).

Figure 5.1. VjControl Tabs



Clicking on a tab exposes its panel; each panel gives access to a particular VjControl function. The functions are as follows:

Configure	Panel for editing and viewing config files. This is the most important option for new users.
-----------	--

Connection	Panel for connecting to VR Juggler applications. Used for dynamic reconfiguration.
Descriptions	Panel used for editing and viewing Chunk Description files.
Org Tree	Panel used for editing and viewing Chunk Description files.
Messages	Panel used for logging error and informational messages from VjControl.
Performance	Panel used for viewing performance data.

In addition to these tabbed panels, the VjControl main window also has a fairly typical set of File, Help, and Options [Preferences.html] menus.

## Loading Files

VR Juggler uses two types of files: Config Files and ChunkDesc Files.

- Config Files are made of “Chunks” of information. For example, one chunk will describe the setup of a display device while another will describe the setup for a keyboard or tracking system. There will typically be several ready-made config files in the `$VJ_BASE_DIR/share/Data` directory, with names like `simulator.config` or `C2_4wall.config`.
- ChunkDesc Files define the different kinds of chunks (ie. what kind of information a Keyboard chunk should have, or a Display chunk). The main ChunkDesc file for VR Juggler is usually `$VJ_BASE_DIR/share/Data/chunksDesc`.

For more information, see (DEAD LINKS!):

- Working with Config Files [[ChunkDBPanel.html](#)]
- Working with ChunkDesc Files [[DescDBPanel.html](#)]

---

# Chapter 6. Command-Line Arguments

This chapter describes the various command line arguments for VjControl.

```
vjcontrol [[hostname] | [port] | [-noautoload] | [-o file] | [-c file] | [-d file]]
```

## Networking Options

If a hostname and (optionally) a port are specified, VjControl will try to establish a network connection to a running VR Juggler application on that machine.

## File Loading Options

-noautoload	VjControl can be configured to automatically load certain files at startup. This switch suppresses automatic loading so that only files given on the command line are loaded.
-o name	Loads the Chunk Organization tree name.
-d name	Loads the Chunk Description file name.
-p name	Loads the performance logging file name.

---

# Chapter 7. Working with Config Files

Clicking on VjControl's "Configure" tab lets you view and edit config files (see Figure 7.1). The display consists of two similar panels, side by side. The idea is that you can view one file on each side, making it easy to compare two files and to copy Chunks back and forth between them.

At the top of each panel is a ComboBox that lets you choose which of the loaded config files to display. The selected file will appear in a tree format in the middle of the panel.

Chunks in the display are organized by type and by broader categories. The specific organization of the tree is determined by VjControl's current ChunkOrgTree. See Chapter 9, *Working with Chunk Organization Trees* for more information.

You can click on a Chunk in the tree to select it. Double-clicking on a Chunk brings up a window that lets you view or edit the Chunk. Right-clicking on a Chunk brings up a small menu that lets you view help information on that Chunk (if available) or remove it from the file (see Figure 7.2). Similarly, right-clicking on folders that represent Chunk types brings up a small menu letting you insert new Chunks (see ).

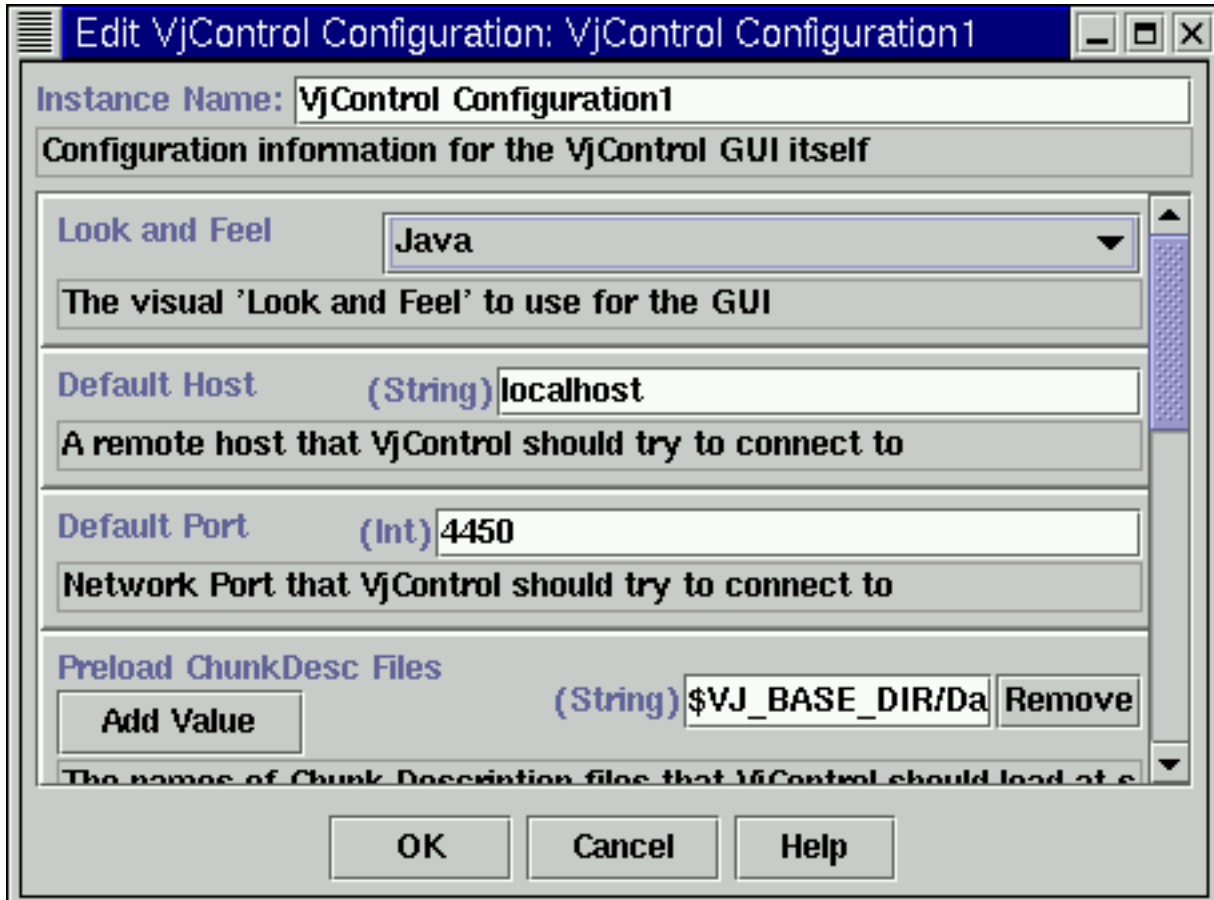
There are eight buttons along the side of each panel:

- *New* - creates a new (empty) config file.
- *Load* - loads a config file and views it in this panel.
- *Save* - save the config file currently being viewed. Note that this button always opens a file dialog, so it's technically Save As...
- *Close* - removes the currently viewed config file from memory. *WARNING!* VjControl does not currently warn you if you close a file without saving it.
- *>>* - copies all selected chunks in this file to the file being viewed in the other panel.
- *>All>* - copies every chunk in this file to the file being viewed in the other panel.
- *Remove* - deletes all selected chunks in the currently viewed file.
- *Duplicate* - adds duplicate copies of all selected chunks in the currently viewed file.
- *Insert* - inserts a new chunk. The type of chunk to insert is determined by the Insert Type combo box at the bottom of the panel.

## The ConfigChunk Editing Window

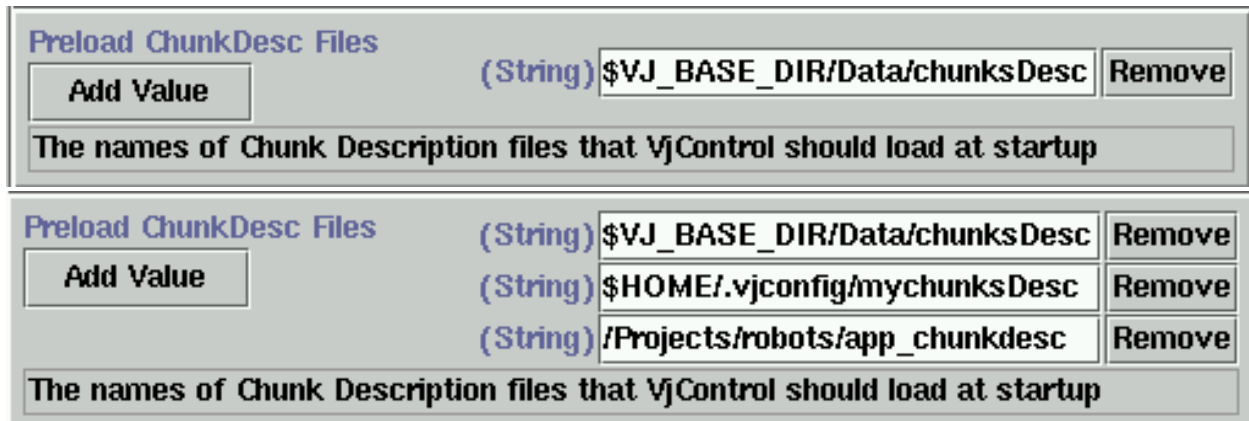
Double-click on one of the ConfigChunks in the tree to open the ConfigChunk editing window. This window displays the values for all the Chunk's properties, and lets you modify these values.

**Figure 7.4. Config Chunk Editing Window**



Some properties in a Chunk allow a variable number of values. These include an Add Value button next to the property name, and a Remove button next to each value (see Figure 7.5).

**Figure 7.5. Property Editors**



Click the OK button to exit the window, or Cancel to close the window without saving the changes you've made.

## Including Other Configuration Files

VR Juggler includes the idea of an "Include File" chunk, which has one property: the pathname of a VR Juggler config file. When a VR Juggler application loads a file with an "Include File" chunk, it will also load the file named in that chunk. This means that it is possible to make a config file that's just a list of other config files to be loaded.

Similarly, there are "Include Description File" chunks which will cause VR Juggler to automatically load a Chunk#Desc file. This is particularly useful for config files which contain application-specific kinds of ConfigChunks. If a config file with includes is loaded into VjControl, the included files are automatically loaded, though each file is viewed separately.

To include a file, simply add an "Include File" (or "Include Descriptions File") chunk to the config file. The name of the "Include File" chunk is the path name of the file to load. You can use the strings "\$HOME" and "\$VJ\_BASE\_DIR" at the start of the name to refer to your home directory and the VR Juggler install directory, respectively.

Relative path names in includes (e.g. paths that don't start with "/" or C:) are relative to the location of the file containing the "Include File" chunk. For example, if the config file `/home/vr/vjconfig/c2.config` includes the file `C2/displays.config`, VR Juggler will try to load a file named `/home/vr/vjconfig/C2/displays.config`.

---

# Chapter 8. Working with Chunk Description Files

VjControl includes the ability to view and edit the Chunk Description files, allowing you to create new kinds of ConfigChunks or modify existing Chunks. Typically, there are two reasons you might want to do this:

1. You're an application writer, and you want your application to be able to configure itself using the VR Juggler configuration mechanism. You could create a new kind of ConfigChunk that contains information specific to your application.
2. You're developing a new component for VR Juggler - a device driver, or something similar. You would create a ConfigChunk to define the driver's configuration info, which could include things like serial port name and speed, calibration files, and so on.

You can load and edit VR Juggler Chunk Description Files by clicking on the "Descriptions" tab of VjControl's main window. The Descriptions panel is similar to the Configure panel [ChunkDBPanel.html], without the side-by-side display of multiple files.

The ComboBox at the top of the panel lets you choose which of the loaded description files to display (in this case, the file is called ``chunksDesc"). Below that is a list of all the ChunkDescs defined in the file. Click on a name to select it. You can double-click on a name to view or edit the ChunkDesc.

There are six buttons along the side of the panel:

- New - creates a new (empty) descriptions file.
- Load - loads a descriptions file.
- Save - save the descriptions file currently being viewed. Note that this button always opens a file dialog, so it's technically ``Save As..."
- Close - removes the currently viewed description file from memory. *WARNING!* VjControl does not currently warn you if you close a file without saving it.
- Remove - deletes all selected descriptions in the currently viewed file.
- Insert - inserts a new description. Initially, a new description only has one property - the instance name.

## The ChunkDesc Editing Window

Double-clicking on the name of a ChunkDesc brings up a new window, shown in Figure 8.2.

The uppermost part of the window displays basic information about the ChunkDesc. Every ChunkDesc has two identifiers: a descriptive name which is used for GUI displays, and a shorter token, which is used in the file format and internally by VR Juggler. The name can be any short descriptive phrase; the token should be terse and may not contain spaces. There is also a help string, which can be used to provide a general one-line description of what the ChunkDesc is for.

The center of the ChunkDesc editing window displays a scrollable list of all the properties that ConfigChunks using this description should have. Every ChunkDesc includes a Name property, which is actually used to store the instance name for ConfigChunks of that type. The number and type of other properties can vary.

Users can add a new property by pressing the "Insert" button at the bottom of the window. Properties can be removed by selecting them (clicking on the background of the property's box) and then pressing the "Remove" button.

As with the ChunkDesc itself, each property has a descriptive name and short token, as well as a help string. Some properties can have multiple values (e.g. a position with separate x, y, and z values); this can be defined by entering the desired number of values in the "Number" box, or by selecting the "Var Arguments" value, for a property with a variable number of values.

Every property has a specific type of legal values (integers, strings, etc.); this is selected in the "Type" box. The meanings of the different types are explained in the ConfigChunk Primer [chunkprimer.html].

## Labeling Individual Values of a Property

If a property allows multiple values, it may be helpful to provide individual labels for each one. For example, a "Window Size" parameter may have values labeled "Width" and "Height". To define labels, click on the property's "Value Labels" button. A window such as the one shown in Figure 8.3 will be opened.

The window contains one line for each value the property allows; in this case we're labeling the individual values "width" and "height". If the property allows a variable number of values, the Value Label window will include Insert and Remove buttons.

## Properties with Enumerations

Sometimes, a property will have a fixed set of possible values (e.g. "Shift" "Control" "Alt" "None" for keypress modifiers). Selecting the "Edit Enumerations" button for a property description will open a window which allows you to define the set of allowed values. When VjControl is displaying a Property with a set of enumerated values, it can present the choices using a menu instead of requiring the user to type a value. The enumerations feature can also be used to provide text names for numeric or boolean values (e.g. assigning the string "NTSC" to the value 60, and "PAL" to 50).

## Properties that Reference or Embed ConfigChunks

Sometimes, a property of a ConfigChunk will be a reference to another ConfigChunk. For example, the ConfigChunk for a graphics window might refer to a position input device to use for view frustum calculations.

To create a property of this type, select "Chunk" as the value type for the property. In place of the "Edit Enumerations" button, you'll see a "Set Allowed Types" button. This opens a small window which lets you select what kinds of ConfigChunks this property should be allowed to point to.

Other times, it may be convenient to completely include a ConfigChunk inside of a property, instead of referring to an external Chunk. For example, VR Juggler config files frequently do this to handle complex data types such as keypresses with modifiers in the configuration of its simulated input devices. VjControl has special features to display a small "embedded" chunk as a single line inside a ConfigChunkFrame.

To create a property of this type, select "Embedded Chunk" as the value type. You'll see an "Inner Chunk Type" button in place of the "Edit Enumerations" button. This button opens a small window, similar to the set allowed types window mentioned above (the difference is that this one only allows you to select a single type of ConfigChunk).

## Using DescHelp files

The help strings for chunks and properties aren't always sufficient to explain the concept behind a given ConfigChunk, so VjControl includes a mechanism to include more verbose documentation as an external HTML file. These files can be automatically displayed by VjControl when a user presses the help button in the ConfigChunk

editor window, or through the help menus, context- sensitive help, etc.

Each "DescHelp" file should be named `token.html`, where `token` is the token of the `ChunkDesc` it describes. `VjControl` searches for `DescHelp` files in several places:

- Inside the `VjControl` binary itself
- In `$VJ_BASE_DIR/share/Data/DescHelp`
- In `$HOME/.vjconfig/DescHelp`

---

# Chapter 9. Working with Chunk Organization Trees

VjControl uses Chunk Organization Trees (OrgTrees) to put some order in the display of config files. The OrgTree defines the hierarchical tree view used in the "Configure" pane. To illustrate the point, here two snapshots of the same config file viewed with different OrgTrees are shown in Figure 9.1 and Figure 9.2.

As you can see, the same ConfigChunks exist in each view - just the way they're grouped together changes.

Clicking on the "Org Tree" tab displays a panel which shows the OrgTree currently in use and provides rudimentary editing capabilities. In Figure 9.3, we show the OrgTree panel with the (rather silly) example used in Figure 9.2:

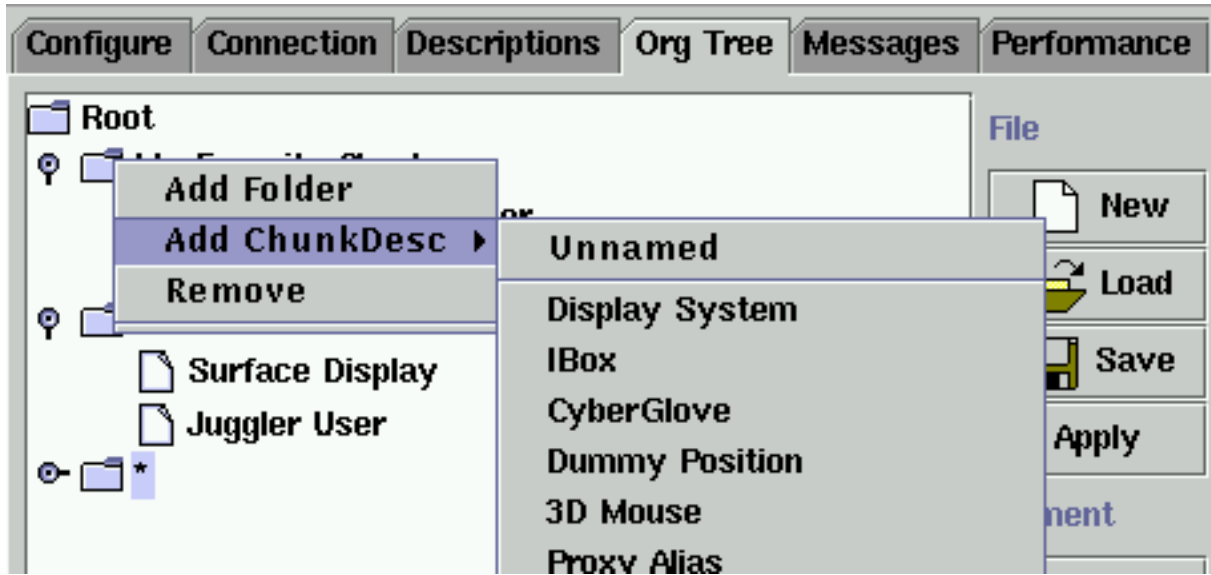
The bulk of the OrgTree pane is taken up by a view of the tree currently in use. The folders represent different categories of config data - "Input", "Displays", "My Favorite Chunks", etc. The leaves of the tree are labeled with the names of types of ConfigChunks. When viewing a config file in the "Configure" pane, the same tree structure will be used, but the leaves will be replaced with folders containing all the chunks in the file of a given type.

One additional note: a folder in the OrgTree with the name "\*" will include all ConfigChunks in a config file. It's useful to include a folder named "\*" in any OrgTree, so that you're sure to see all the chunks in any config file.

The New, Save, and Load buttons work in the same way as the other panels, but there are also a number of additional features:

- The *Apply* button is used to update the views in the Configure pane to use the tree shown in the OrgTree pane. Use this button after loading an OrgTree file or making changes to the structure.
- You can select elements in the tree by clicking on them. The *Remove* button deletes all selected elements from the tree.
- You can change the name of a tree element by double-clicking on it.
- Right-clicking on a tree element brings up a menu, shown in Figure 9.4.

## Figure 9.4. ChunkDesc Add Menu



This menu lets you insert folders or Chunk types, or remove the element.

## Missing Functionality

The major missing feature in the current OrgTree is the ability to drag around elements of the tree to rearrange them. This ability is forthcoming. For the time being, curious and adventurous readers are referred to Chapter 14, *ChunkOrgTree File Format*.

---

# Chapter 10. VjControl Preferences

VjControl has a number of user-configurable options. On startup, it will attempt to load these preferences from the file `$HOME/.vjconfig/vjcontrol.cfg`. If that file does not exist, VjControl falls back to its hard-coded defaults.

The preferences can be accessed from "Preferences" in the Options menu. "Edit Preferences" brings up a window to (surprise) edit the preferences. This is the same as the usual window used for editing ConfigChunks. The "Save Preferences" option saves the current preferences to `$HOME/.vjconfig/vjcontrol.cfg`, where they will be loaded from the next time VjControl starts.

## VjControl Preference Options

- Look and Feel

This option lets the user change the overall look-and-feel (L&F) of the GUI, using the Java Swing classes. The possible options are: Java (the default, also called "Metal"), Motif, Windows, and Macintosh. Note that the Windows and Mac L&Fs are only available when running on their respective platforms. VjControl will revert to Java/Metal if the requested L&F is unavailable.

Most of the differences between L&Fs are cosmetic, but you may wish to experiment. The Windows L&F performs slightly better than Java/Metal under W98/NT. The Java/Metal L&F has been most extensively tested.

- Default Host and Default Port

These options are the default host name and port number that will appear in the Connections panel. See Chapter 11, *Dynamically Reconfiguring VR Juggler Applications* for more information.

- Preload ChunkDesc Files

Here you can specify one or more Chunk Description files that will be loaded automatically when VjControl starts up. The arguments are simply pathnames to files. If the strings `$VJ_BASE_DIR` or `$HOME` are included in a path name, they will be replaced with the value of the `$VJ_BASE_DIR` environment variable or the user's home directory, respectively.

It is suggested that you include `$VJ_BASE_DIR/share/Data/chunksDesc`, the main Chunk Description file.

You can override the automatic loading of files at startup by using the `-noautoload` command-line argument (refer to Chapter 6, *Command-Line Arguments* for more information).

- Preload Configuration Files

This lets you specify one or more Configuration files to load at startup (The details are similar to the option above).

- Default ConfigChunk Files

This lets you specify one or more Configuration files which contain default values for new ConfigChunks. When you create a new ConfigChunk in VjControl, it checks the default files for a Chunk of that type. If found, it copies that value into the new Chunk.

- Font Name and Font Size

This specifies the name and size of font to use for all of VjControl's windows. Note that currently only a small

number of fonts are available (this is a limitation of JDK 1.1, and will be fixed at a later date).

---

# Chapter 11. Dynamically Reconfiguring VR Juggler Applications

One of VR Juggler's most interesting and unique features is *dynamic reconfiguration*: the ability to view and modify the configuration of the VR Juggler system without restarting an application. In addition to its role as a configuration file editor, VjControl is also the user interface for viewing and changing VR Juggler's configuration.

## Connecting to a VR Juggler application

VjControl communicates with VR Juggler applications over a TCP/IP network connection. This allows VjControl to control VR Juggler applications remotely - from another computer, another room, or another city. This feature is particularly valuable because many VR systems take over their host computer's display, limiting the use of traditional GUI programs.

To connect to a VR Juggler application, switch to the "Connections" panel in VjControl. You will need to enter the hostname and port number that the VR Juggler application is listening to. The hostname and port are in turn specified by one of the configuration files given to the application at startup time (in an Environment Manager ConfigChunk) and are printed to the console as part of the application's initialization.

Once the hostname and port number are in place, press the "Connect" button, and VjControl will open a communications channel to the VR Juggler application.

Alternately, you can specify the hostname and port with VjControl's command-line arguments (see Chapter 6, *Command-Line Arguments*) and it will connect automatically.

## Viewing and Changing the Active Configuration

Once the connection has been established, switch to the "Configure" panel in VjControl. The list of loaded configuration files which can be selected will contain a new entry called "Active Configuration". Selecting this will display the current configuration of the VR Juggler application - all of the active components displayed as if they were just another configuration file.

Unlike a simple configuration file, changes to this database do not just change VjControl's internal representation. If the user removes a ConfigChunk from the Active Configuration, or copies a Chunk into it, or edits a Chunk and presses the "OK" or "Apply" button, VjControl will send a reconfiguration command across the network connection to the VR Juggler application. The application will process the reconfiguration command and send an updated snapshot of its configuration to VjControl.

Adding a new ConfigChunk to the Active Configuration (as opposed to copying a Chunk into it) does not send a reconfiguration command to the application, since assumably the new Chunk still needs to be edited by the user. The new Chunk won't be sent to the VR Juggler application until the user edits it and presses the "OK" or "Apply" button in the ConfigChunk editor window.

---

## **Part III. Technical Specification**

---

---

# Table of Contents

- 12. Config File Format ..... 32
  - Overview ..... 32
  - General Rules ..... 32
  - Config Files ..... 33
  - ConfigChunks ..... 33
  - Property Lines ..... 33
  - Properties with Enumerations ..... 34
  - Example Property Lines ..... 34
- 13. ConfigDesc File Format ..... 35
  - Overview ..... 35
  - General Rules ..... 35
  - ChunkDesc Files ..... 36
  - ChunkDescs ..... 36
  - Property Description Lines ..... 36
- 14. ChunkOrgTree File Format ..... 38

---

# Chapter 12. Config File Format

## Overview

The config files used by VR Juggler and the VjControl program are ASCII text files based around the idea of "ConfigChunks" - logical groupings of configuration for particular parts of a system (e.g. displays, input devices, or simulators). The different kinds of ConfigChunks are defined in ChunkDesc files.

It's important to note that you should load the ChunkDesc files describing the available Chunks before loading a config file - otherwise the config file won't parse correctly.

Here is an example config file with two entries:

```
display
  Name { "simView" }
  projectiontype { "front" }
  origin { 0 0 }
  size { 600 600 }
  pipe { 0 }
  stereo { False }
  border { True }
  simulator { True }
  simCamera { "CameraProxy" }
end
flock
  Name { "flock 1" }
  Port { "/dev/tty52" }
  Baud { 38400 }
  num { 1 }
  translate { 0 0 0 }
  rotate { 180.0 0 0 }
end
End
```

As you can see, this file defines two chunks. The first is of type Display (i.e. it uses a ChunkDesc whose token is "Display"). The name of this chunk is "simView". All Chunks must have a unique name - if none is given, the parser will give it a geeky default name like "Display\_42" and all the other Chunks will make fun of it.

The remaining properties are defined by the ChunkDesc for Display. Each line gives the token for one of the properties followed by a list of values (the type and allowed number of values are also defined in the ChunkDesc).

## General Rules

1. Config files are generally non-case-sensitive. For example, "end", "End", and "eND" are all valid ways to end a chunk or a file. Similarly, "Display" in the above example could have been written "DISPLAY", and still would have used the same ChunkDesc. The main exception to this is the contents of string values, which may well be used in a case sensitive manner. An obvious example is a string containing the name of a file on a UNIX file system.
2. The following keywords are reserved in ChunkDesc files: end, and generally any word with the prefix vj\_.
3. Strings (names, help strings, etc.) should always be surrounded by quotes.
4. Strings cannot contain line breaks or quotes

5. In general, the ChunkDesc parser treats all whitespace the same, so it is possible for parts of a line to be broken up for greater readability.

## Config Files

The general format of a config file is:

```
ConfigChunk1
ConfigChunk2
...
End
```

That is, zero or more ConfigChunks followed by the token End.

Several forms of comments are allowed inside the files. C-style comments (*/\* comment \*/*) can be used to comment out blocks of text. Shell and C++ comments (*#* and *//*) can be used to comment out text to the next newline character.

Note that the VjControl program does not preserve comments in the files that it loads and saves, so this is mostly useful as a quick hack or as a debugging feature.

## ConfigChunks

The basic format of a ConfigChunk is:

```
token
  Name { "name" }
  propertytoken { value1 value2 ... }
  ...
end
```

The first line contains only the token of the ChunkDesc that this ConfigChunk uses (note that in general, where tokens of ChunkDescs or PropertyDescs are used, they are not put in quotes, though this would be allowed).

The second line should always be the Name property. This is followed by any other Property lines, and finally by end. Note that it is not strictly necessary to include all of the Properties defined in the Chunk's description. Those that are omitted will be filled in with default values (0, 0.0, false, ""). Omitted properties that allow variable numbers of values will start out with no values at all.

## Property Lines

The format for any property line is this:

```
token { value1 value2 ... }
```

Where token is the token for one of the properties defined in the ChunkDesc. Values is a list of 0 or more values separated by whitespace. The surrounding curly braces should be considered mandatory.

The format for a value varies depending upon type. Integers and Floats are simply represented as numbers: 0, 42, 44.556, etc. (Warning: VjControl doesn't parse scientific notation for floats). String values are simply represented as quoted text strings. Boolean values are represented as the strings "True" and "False" (case is irrelevant, and quotes unnecessary).

## Properties with Enumerations

Some Properties are defined to have a set of named values. If this is the case, the values in the property line can be either names defined in the enumeration or actual values (the example below of an int Property with an enumeration includes one value of each type).

## Example Property Lines

```
stringproperty { "a string value" "a second string value" }
intproperty { 42 }
floatproperty { 42.0 43.0 }
booleanproperty { True }
intproperty_with_enumeration { "a name from the enumeration" 42 }
```

---

# Chapter 13. ConfigDesc File Format

## Overview

A ChunkDesc file includes descriptions of different kinds of ConfigChunks. It is a way to configure the configuration system itself. Here's a simple example, a ChunkDesc file with two entries.

```
chunk flockobirds
  port string 1 port "serial port address"
  offset double 3 offset "tracker coord offset"
  baudrate int 1 baud "serial port rate"
end

chunk display
  name string 1 name "descriptive name"
  origin double 3 origin "other help text"
  size int 2 size ""
  SGIpipeline int 1 SGIpipeline ""
  xdisplay string 1 xdisplay ""
  stereo bool 1 stereo ""
  lowerleft double 3 lowerleft ""
  lowerright double 3 lowerright ""
  upperleft double 3 upperleft ""
end

End
```

The first entry defines a chunk type called "flockobirds" that contains three properties ("port", "offset", and "baudrate"). The second defines a chunktype called "display" which contains 9 properties.

Each property has a type - for example, the values that "port" can take will be strings, such as "/dev/ttyd45", while baudrate will always have integer values.

It is also possible for a property to have a set of values. For example, the "display" chunk's "lowerleft", "lowerright", and "upperleft" properties are meant to represent positions in 3d space. These can be represented by a sequence of three doubles for the x, y, and z coordinates. The number of values a property supports is listed directly to the right of the type - the "3" in the line for "lowerleft", or the "1" for "stereo." If a property might have a variable number of entries (perhaps as a list of active displays, or such), use the number -1.

The fourth word on each property line is the actual token to be used to represent this property when reading/writing config files. Most often this will be the same as the name of the property (which is used internally for querying a ConfigChunk), but the options is provided to make the config files more flexible or readable.

The last thing on a property line is a help string. If the help string contains spaces or is empty, it must be in quotes.

## General Rules

1. ChunkDesc files are non-case-sensitive. For example, "end", "End", and "eND" are all valid ways to end a chunk description or a file. Similarly, the ChunkDesc system would make no distinction between ChunkDescs with the tokens "flock" and "FLOCK".
2. The following keywords are reserved in ChunkDesc files: `end`, `string`, `int`, `boolean`, `chunk`, `float`, `vj_valuelabels`, `vj_enumerations`, and generally any word with the prefix `vj_`.
3. Strings (names, help strings, etc.) should always be surrounded by quotes.

4. Strings cannot contain line breaks or quotes (the latter is considered a parser bug for now)
5. In general, the ChunkDesc parser treats all whitespace the same, so it is possible for parts of a line to be broken up for greater readability.

## ChunkDesc Files

The general format of a ChunkDesc file is:

```
[ ChunkDesc ]  
...  
End
```

That is, zero or more ChunkDescs followed by the token End.

Several forms of comments are allowed inside the files. C-style comments (`/* comment */`) can be used to comment out blocks of text. Shell and C++ comments (`#` and `//`) can be used to comment out text to the next newline character.

Note that the VjControl program does not preserve comments in the files that it loads and saves, so this is mostly useful as a quick hack or as a debugging feature.

## ChunkDescs

The basic format of a ChunkDesc is:

```
chunk "token" "longer name" "help info"  
  Name String 1 "Name" "  
  [other Property Descriptions]  
  ...  
end
```

The first line contains the token chunk, followed by three strings. The token is used in programs to refer to the ChunkDesc. The name is a longer name that is used in graphical interfaces instead of the token. (The name can contain spaces, the token must not). The help info is a string shown in the graphic interface. All three strings must be defined, though the help string can be empty ("").

The first line is followed by one or more Property Descriptions, followed by the token end. Note that the first property description must always be for the "Name" property - this is used to give a unique name to every ConfigChunk using this ChunkDesc. If there is no "Name" PropertyDesc, one will be added by the loader.

## Property Description Lines

The format of a Property Description line is:

```
"token" type num "Name" [optional stuff] "help"
```

The Token, Name, and Help serve the same purposes as in the Chunk line described above.

Type refers to the type of value stored in this property. Valid values are `String`, `Int`, `Float`, `Boolean`, and `Chunk` (a property that stores the names of other ConfigChunks).

Num refers to the number of values that this Property can store. Usually this is 1. A coordinate in 3d space might be stored with 3 float values. A value of -1 in this field means that the property can have a variable number of values

(useful for lists of available components, directory search paths, etc.).

There are also two optional fields in a PropertyDesc line, either or both of which can be placed between the Name and Help fields of the line.

- Enumerations

Sometimes a given Property will have a small number of possible values. For example, in configuring a C2-like device, we might have a description for a Property called "ActiveWalls":

```
ActiveWalls String -1 "Active Walls" ""
```

Since cubes have six sides, that's probably the total number of possible values we have. We can build that information into the Property definition like so:

```
ActiveWalls String -1 "Active Walls"
  vj_enumerations { "front" "back"
    "floor" "ceiling" "left"
    "right" } ""
```

The advantage of doing this is that the GUI can prevent someone with a list of the valid options for this property.

For int and double Properties, enumerations can match descriptive names with numbers, like so (assuming that our application identifies walls with integers instead of strings):

```
ActiveWalls Int -1 "Active Walls"
  vj_enumerations { "front=1" "back=3"
    "floor=2" } ""
```

Note that the entries can be put in any order, and that it is possible for multiple names to refer to the same value.

If you are editing a ChunkDesc file by hand, you can leave off the "=x" part of the enumeration entries. In this case the loader will automatically assign the values 0, 1, 2... (or 0.0, 1.0, 2.0...) to the entries as they are read.

Frankly, you're better off just using the editor.

- Value Labels

Consider this Property definition:

```
Orientation Float 3 "Orientation" ""
```

It apparently defines an orientation in 3D space, but how? Euler angles? Azimuth, elevation and roll? Looking at this in the VjControl GUI, it would be nice to have individual labels next to each of the values so we know what's what. Value labels let us do that, like so:

```
Orientation Float 3 "Orientation"
  vj_valuelabels { "azimuth"
    "elevation" "roll" } ""
```

Now we know that the first value in Orientation represents the azimuth, the second is elevation, etc. The GUI can display the value labels next to the values' fields.

---

# Chapter 14. ChunkOrgTree File Format

A Chunk Organization Tree (OrgTree) is used to put a logical tree structure onto a ConfigChunkDB. VjControl uses the OrgTree to provide a file-folder-like view of a config file. The OrgTree file is composed of OrgTreeElements, which look like this:

```
begin "name"  
    contents  
end
```

where *contents* is a list of OrgTreeElements and chunk type names, separated by whitespace (Note that that's chunk type *names*, not tokens). The entire file is a single OrgTreeElement (usually called "root", though the GUI never actually uses its name) containing all the other elements. Here's a real example:

```
Begin "Zelda"  
    Begin "Input"  
        Begin "Proxies and Aliases"  
            "ProxyAlias"  
        End  
        Begin "Input Devices"  
            "Keyboard"  
            "flock"  
            "ibox"  
            "Cyberglove"  
            "ThreeDMouse"  
            "DummyPosition"  
        End  
    End  
    Begin "Expert Only"  
        Begin "Performer API"  
            "ApiPerformer"  
        End  
        Begin "Environment Manager"  
            "Environment Manager"  
            "EM File Output"  
        End  
    End  
    Begin "*"   
    End  
End
```

This OrgTree will create a tree view where, for example, the top level will have a subfolder called "Input" containing a folder called "Input Devices" which will list all chunks of types Keyboard, flock, ibox, etc.

There are two important things to note: first, it's entirely ok for chunk type names to appear in multiple folders. Second, the last OrgTreeElement has the name "\*" and no children. Any OrgTreeElement named "\*" will represent a folder that lists all available chunks of any type.