Open RV

Release 2.0.0

Open RV contributors

May 16, 2024

BUILDING OPEN RV

1 Building Open RV on CentOS 7 **1**
- 1.1 Summary . . . 1
- 1.2 Install Basics . . . 1
- 1.3 Install devtoolset-9 . . . 1
- 1.4 Install tools and build dependencies . . . 2
- 1.5 Install CMake . . . 2
- 1.6 Install nasm . . . 2
- 1.7 Install Qt . . . 3

2 Building Open RV on Rocky 8 **5**
- 2.1 Summary . . . 5
- 2.2 Install Basics . . . 5
- 2.3 Install tools and build dependencies . . . 5
- 2.4 Install CMake . . . 6
- 2.5 Install Qt . . . 6

3 Building Open RV on Rocky 9 **7**
- 3.1 Summary . . . 7
- 3.2 Install Basics . . . 7
- 3.3 Install tools and build dependencies . . . 7
- 3.4 Install CMake . . . 8
- 3.5 Install Qt . . . 8

4 Building Open RV on macOS **9**
- 4.1 Summary . . . 9
- 4.2 Install XCode . . . 9
- 4.3 Install Homebrew . . . 9
- 4.4 Install tools and build dependencies . . . 9
- 4.5 Install the python requirements . . . 10
- 4.6 Install Qt . . . 10

5 Building Open RV on Windows **11**
- 5.1 Summary . . . 11
- 5.2 1. Install Microsoft Visual Studio . . . 11
- 5.3 2. Install Qt . . . 11
- 5.4 3. Install Strawberry Perl . . . 12
- 5.5 4. Install MSYS2 . . . 12
- 5.6 5. Install required MSYS2 pacman packages . . . 12

6 Build Errors and Solutions **15**
- 6.1 Mu::Parse(char const*, Mu::NodeAssembler*) . . . 15

6.2 Numerous undeclared identifier ‘YYMUFlexLexer’ Errors 15
6.3 (macOS) from ranlib tool: XYZ has no symbols 15
6.4 Linker Error, Undefined symbols “Mu::Parse(char const*, Mu::NodeAssembler*)” 16
6.5 Multiple ‘error: unknown type name ‘string’ reported in system files 16
6.6 (Linux) /usr/bin/env: ‘python’: No such file or directory 17
6.7 (Linux) pyconfig.h: No such file or directory 17
6.8 (Linux) error: libaio.h: No such file or directory 17
6.9 (Rocky Linux) 18
6.10 (Linux any Distro) 18
6.11 Windows 10 & Windows 11 (Windows 10+) : PIP REQUIREMENTS: py7zr fails to install 18
6.12 Windows : System-wide vcpkg 19
6.13 macOS, Linux, Windows 19

7 Chapter 1 - Introduction 21
7.1 1.1 Overview 21
7.2 1.2 Getting Started With Open RV 21
7.3 1.2.10 Customizing Open RV 24
7.4 1.3 Getting Started with RVIO 24

8 Chapter 2 - Installation 27
8.1 Proxy Configuration 27
8.2 About Self-Signed Certificates and Open RV 27

9 Chapter 3 - Command Line Usage 29
9.1 Troubleshooting Open RV 30
9.2 3.1 Image Sequence Notation 33
9.3 3.2 Source Layers from the Command Line 35
9.4 3.3 Directories as Input 37

10 Chapter 4 - User Interface 39
10.1 4.1 Feedback 41
10.2 4.2 Main Window Tool Bars 42
10.3 4.3 Loading Images, Sequences, Movies and Audio 43
10.4 4.4 Examining an Image 45
10.5 4.5 Playing Image Sequences, Movie Files, and Audio Files 55
10.6 4.6 Audio 60
10.7 4.7 Caching 63
10.8 4.8 Color, LUTs, and CDLs 64
10.9 4.9 Stereo 64
10.10 4.10 Key and Mouse Bindings 64
10.11 4.11 Preferences File 71
10.12 4.12 Audio Settings 71

11 Chapter 5 - The Session and the Session Manager 73
11.1 5.1 Open RV Session 73
11.2 5.1.1 What’s in a Session 73
11.3 5.2 Session Manager 75
11.4 5.3 Creating, Adding to, and Removing from a View 78
11.5 5.4 Navigating Between Views 79
11.6 5.5 Source Views 80
11.7 5.5.1 Source Media Containing Multiple Images (Subcomponents) 80
11.8 5.6 Group Views 81
11.9 5.7 Retime View 85
11.10 5.8 Folders 87

12 Chapter 6 - Presentation Mode and Video Devices **89**
12.1 6.1 Video Device Configuration 89
12.2 6.2 Display Profiles 91
12.3 6.3 Video Device Command Line Arguments 93
12.4 6.4 Presentation Mode Settings 93
12.5 6.5 Platform Specific Considerations 93
12.6 6.6 HDMI Frame Packed Mode 94

13 Chapter 7 - How a Pixel Gets from a File to the Screen **99**
13.1 7.1 Image Layers 101
13.2 7.2 Image Attributes 101
13.3 7.3 Image Channels 102
13.4 7.4 Crop and Uncrop 104
13.5 7.5 Conversion to Linear Color Space 106
13.6 7.6 Color Correction 109
13.7 7.7 Display Simulation and Correction 114
13.8 7.8 Final Display Filters 116

14 Chapter 8 - Using LUTs in Open RV **119**
14.1 8.1 Channel (1D) versus 3D LUTs 120
14.2 8.2 Input Matrix and Pre-LUT 120
14.3 8.3 The Pre-Cache LUT 121
14.4 8.4 LUT File Formats 121

15 Chapter 9 - Using CDLs in Open RV **125**
15.1 9.1 CDL File Formats 125

16 Chapter 10 - Packages **127**
16.1 10.1 Package Support Path 127
16.2 10.2 Installation 128
16.3 10.3 Package Dependencies 129

17 Chapter 13 - Networking **131**
17.1 13.1 The Network Dialog 131
17.2 13.2 Synchronizing Multiple RVs 134
17.3 13.3 “Streaming” Movie Media from Online Sources 137

18 Chapter 11 - OpenColorIO **139**
18.1 11.1 The ocio_source_setup Package 139

19 Chapter 12 - Stereo Viewing **143**
19.1 12.1 Anaglyph or Luminance Anaglyph 143
19.2 12.2 Side-by-Side and Mirror 147
19.3 12.3 DLP Checker and Scanline 149
19.4 12.4 HDMI 1.4a 149
19.5 12.5 Hardware Stereo Support 149
19.6 12.6 Additional Stereo Operations 149

20 Chapter 14 - Maximizing Performance **153**
20.1 Internal Software Operations 154
20.2 Steps to Improve Performances 155
20.3 System Requirements 156
20.4 Cache Settings 156
20.5 Reader Threads 157
20.6 I/O Methods 157

20.7 Real-Time RV on Linux . . . 159
20.8 Refresh Rate and V-Sync . . . 159
20.9 Testing Playback Performances and the Filesystem Cache . . . 160

21 Chapter 15 - File Formats 163
21.1 15.1 Movie File Formats . . . 163
21.2 15.2 Image File Formats . . . 166
21.3 15.3 Audio File Formats . . . 170
21.4 15.4 Simple ASCII EDL Format . . . 171

22 Chapter 16 - RVIO 173
22.1 16.1 Basic Usage . . . 175
22.2 16.2 Advanced Usage . . . 175

23 Chapter 17 - RVLS 183

24 Chapter 18 - RVPUSH 185

25 A - Tuning Platform Audio for Linux 187
25.1 K.1 Suggestions for resolving audio static issues with PulseAudio for Linux . . . 190

26 B - Stereo Setup 191
26.1 B.1 Linux . . . 191
26.2 B.2 macOS and Windows . . . 193

27 C - The RVLINK Protocol: Using Open RV as a URL Handler 195
27.1 C.1 Using rvlink URLs . . . 195
27.2 C.2 Installing the Protocol Handler . . . 196
27.3 C.3 Custom Environment Variables . . . 197
27.4 C.4 Testing the Protocol Handler . . . 197
27.5 C.5 One-click sync . . . 198

28 D - Using Open RV as Nuke's Flip Book Player 199
28.1 D.1 Setting up a custom plugins area for Nuke . . . 199
28.2 D.2 Adding Open RV support in the custom plugins area . . . 199

29 E - Open RV Audio on Linux 201
29.1 E.1 Overview . . . 201
29.2 E.2 ALSA (Pre-1.0.14) . . . 202
29.3 E.3 ALSA (Safe) . . . 202
29.4 E.4 Platform Audio . . . 203

30 F - Troubleshooting Networking 205
30.1 Configure RV's network settings . . . 205
30.2 Connections only work from one direction or are always refused . . . 205

31 G - Rising Sun Research CineSpace .csp File Format 207
31.1 Header . . . 207
31.2 Metadata . . . 207
31.3 1D preLUT data . . . 208
31.4 LUT data . . . 208
31.5 Notes: . . . 209

32 H - Crash Reporting 211

33 I - PySide example usage 213

34 J - Supported Multichannel Audio Layouts **217**

35 K - Localizing media paths with RV_OS_PATH or RV_PATHSWAP **219**
35.1 RV_OS_PATH Variables . . . 219
35.2 Open RV PATHSWAP Variables . . . 220
35.3 Hand-written Session Files or RVLINK URLs . . . 221
35.4 Remote Sync . . . 221

36 Chapter 1 - Overview **223**
36.1 1.1 The Big Picture . . . 223
36.2 1.2 Drawing . . . 224
36.3 1.3 Menus . . . 224

37 Chapter 2 - Image Processing Graph **225**
37.1 2.1 Top-Level Node Graph . . . 225

38 Chapter 3 - Writing a Custom GLSL Node **243**
38.1 3.1 Node Definition Files . . . 243
38.2 3.2 Fields in the IPNodeDefinition . . . 244
38.3 3.3 Alternate File URL . . . 247
38.4 3.4 Augmented GLSL Syntax . . . 247
38.5 3.5 Testing the Node Definition . . . 249
38.6 3.6 Publishing the Node Definition . . . 249

39 Chapter 4 - Python **251**
39.1 Open RV Python quickstart . . . 251
39.2 4.1 Calling Mu From Python . . . 252
39.3 4.2 Calling Python From Mu . . . 253
39.4 4.3 Python Mu Type Conversions . . . 255
39.5 4.4 PySide Example . . . 256
39.6 4.5 Open RV Python Implementation FAQ . . . 257
39.7 Why does my external Python process (which I call from Open RV) now behave differently? . . . 257

40 Chapter 5 - Event Handling **259**
40.1 5.1 Binding an Event . . . 261
40.2 5.2 Keyboard Events . . . 262
40.3 5.3 Pointer (Mouse) Events . . . 262
40.4 5.4 The Render Event . . . 263
40.5 5.5 Remote Networking Events . . . 264
40.6 5.6 Internal Events . . . 264

41 Chapter 6 - Open RV File Format **267**
41.1 6.1 How Open RV Uses GTO . . . 267
41.2 6.2 Naming . . . 268
41.3 6.3 A Simple Example . . . 268
41.4 6.4 Per-Source and Display Color Settings and LUT Files . . . 269
41.5 6.5 Information Global to the Session . . . 273
41.6 6.6 The Graph . . . 274
41.7 6.7 Creating a Session File for Custom Review . . . 275
41.8 6.8 Limitations on Number of Open Files . . . 282
41.9 6.9 What's the Best Way to Write a .rv (GTO) File? . . . 282

42 Chapter 7 - Using Qt in Mu **283**
42.1 7.1 Signals and Slots . . . 284
42.2 7.2 Inheriting from Qt Classes . . . 285

43 Chapter 8 - Modes and Widgets **287**
43.1 8.1 Outline of a Mode . . . 288
43.2 8.2 Outline of a Widget . . . 289

44 Chapter 9 - Package System **291**
44.1 9.1 rvpkg Command Line Tool . . . 291
44.2 9.2 Package File Contents . . . 294
44.3 9.3 PACKAGE Format . . . 294
44.4 9.4 Package Management Configuration Files . . . 297
44.5 9.5 Developing a New Package . . . 298
44.6 9.6 Loading Versus Installing and User Override . . . 299

45 Chapter 10 - A Simple Package **301**
45.1 10.1 How Menus Work . . . 304
45.2 10.2 A Menu in MyStuffMode . . . 305
45.3 10.3 Finishing up . . . 306

46 Chapter 11 - The Custom Matte Package **307**
46.1 11.1 Creating the Package . . . 307
46.2 11.2 The Custom Matte File . . . 307
46.3 11.3 Parsing the Matte File . . . 307
46.4 11.4 Adding Bindings and Menus . . . 310
46.5 11.5 Handling Settings . . . 313
46.6 11.6 The Finished custom_mattes.py File . . . 315

47 Chapter 12 - Automated Color and Viewing Management **319**
47.1 12.1 The source-group-complete Event . . . 320
47.2 12.2 The default source-group-complete behavior . . . 320
47.3 12.3 Breakdown of sourceSetup() in the source_setup Package . . . 321
47.4 12.4 Setting up 3D and Channel LUTs . . . 325
47.5 12.5 Setting CDL Values From File . . . 325
47.6 12.6 Building a Package For Color Management . . . 326

48 Chapter 13 - Network Communication **327**
48.1 13.1 Example Code . . . 327
48.2 13.2 TwkQtChat Library . . . 329
48.3 13.3 The Protocol . . . 330

49 Chapter 14 - Webkit JavaScript Integration **335**
49.1 14.1 Executing Mu or Python from JavaScript . . . 335
49.2 14.2 Getting Event Call Backs in JavaScript . . . 335
49.3 14.3 Using the webview Example Package . . . 337

50 Chapter 15 - Hierarchical Preferences **339**

51 Chapter 16 - Node Reference **341**
51.1 RVCache . . . 341
51.2 RVCacheLUT and RVLookLUT . . . 341
51.3 RVCDL . . . 342
51.4 RVChannelMap . . . 342
51.5 RVColor . . . 342
51.6 RVDispTransform2D . . . 343
51.7 RVDisplayColor . . . 343
51.8 RVDisplayGroup and RVOutputGroup . . . 344
51.9 RVDisplayStereo . . . 345

51.10 RVFileSource . . . 345
51.11 RVFolderGroup . . . 346
51.12 RVFormat . . . 346
51.13 RVImageSource . . . 347
51.14 RVLayoutGroup . . . 349
51.15 RVLensWarp . . . 349
51.16 RVLinearize . . . 351
51.17 OCIO (OpenColorIO), OCIOFile, OCIOLook, and OCIODisplay . . . 352
51.18 RVOverlay . . . 353
51.19 RVPaint . . . 354
51.20 RVPrimaryConvert . . . 356
51.21 PipelineGroup, RVDisplayPipelineGroup, RVColorPipelineGroup, RVLinearizePipelineGroup, RVLookPipelineGroup and RVViewPipelineGroup . . . 356
51.22 RVRetime . . . 357
51.23 RVRetimeGroup . . . 357
51.24 RVSequence . . . 358
51.25 RVSequenceGroup . . . 358
51.26 RVSession . . . 358
51.27 RVSoundTrack . . . 359
51.28 RVSourceGroup . . . 359
51.29 RVSourceStereo . . . 359
51.30 RVStack . . . 359
51.31 RVStackGroup . . . 360
51.32 RVSwitch . . . 360
51.33 RVSwitchGroup . . . 361
51.34 RVTransform2D . . . 361
51.35 RVViewGroup . . . 361

52 Chapter 17 - Additional GLSL Node Reference **363**
52.1 17.1 Color Nodes . . . 364
52.2 17.2 Transition Nodes . . . 368
52.3 17.2.1 CrossDissolve . . . 369
52.4 17.2.2 Wipe . . . 369

53 GTO: The Kitchen Sink of Data **371**
53.1 Overview . . . 371
53.2 New in Version 4 . . . 372
53.3 Overview . . . 373
53.4 Binary Format . . . 373
53.5 Text Format . . . 375
53.6 Types of Property Data . . . 380
53.7 Interpretation Strings . . . 382
53.8 Object Protocols . . . 384
53.9 Naming Conventions . . . 397
53.10 Issues and Questionable Aspects of the Format . . . 399
53.11 C++ Library . . . 400
53.12 Python Module . . . 406
53.13 Utilities . . . 412

54 Using LUTs in Open RV **419**
54.1 Channel (1D) versus 3D LUTs . . . 419
54.2 Input Matrix and Pre-LUT . . . 420
54.3 The Pre-Cache LUT . . . 421
54.4 LUT File Formats . . . 421

55 Using Multiple Media Representations in Open RV **423**
55.1 Introduction . . . 423
55.2 Swap between source media representations . . . 423
55.3 Exporting MMR clips to OTIO . . . 427
55.4 Multiple Media Representations: Technical Details . . . 427

56 Mu Programming Language **431**
56.1 1. Overview . . . 431
56.2 2. Primitive Types . . . 433
56.3 3. Built-in Types . . . 437
56.4 4. User Defined Types: Structs, Classes, and Unions . . . 442
56.5 5. Functions and the Function Type. . . . 445
56.6 6. Constants and Initialization . . . 448
56.7 7. Polymorphic and Parameterized Types . . . 451
56.8 8. Variables . . . 451
56.9 9. Pattern Matching . . . 453
56.10 10. Flow Control . . . 455
56.11 11. Namespace (Scoping) Rules . . . 462
56.12 12. Symbol Aliasing . . . 464
56.13 13. Separate Parse and Compilation Modules . . . 467
56.14 14. Documenting Source Code . . . 467
56.15 15. Memory Management . . . 468
56.16 16. Closures and Partial Evaluation . . . 469
56.17 17. Phases . . . 471
56.18 18. Muc: Mu Compiler . . . 472
56.19 19. Standard Library . . . 472
56.20 20. Mu Compared to Similar Languages . . . 473
56.21 21. Example Usage . . . 473
56.22 Appendix A Reference . . . 473

57 Align Start Frames Package (to slide sources in time so that they can be compared) **475**

58 RV/Maya Integration (v1.4) **477**
58.1 Requirements . . . 477
58.2 Installation . . . 477
58.3 Build a Session in Open RV . . . 478
58.4 Organizing and Comparing your Renders . . . 478
58.5 Creating New Views . . . 479
58.6 Rendering into Context . . . 479

59 Nuke style file syntax in Open RV **481**

60 Nuke/Open RV integration **483**
60.1 Introduction . . . 483
60.2 Note to Users . . . 484
60.3 Updating an Existing Installation . . . 484
60.4 Installation . . . 484
60.5 Getting Started . . . 485
60.6 Read/Write Nodes . . . 486
60.7 Checkpoints and Renders . . . 486
60.8 Working with Media in Open RV . . . 488
60.9 Modifying the Nuke Project from Open RV . . . 488

61 OTIO Reader Package Overview **489**
61.1 Imported and exported nodes . . . 489

61.2 Package contents 489
61.3 Using the OTIO hook system 490
61.4 Media Multi-Reference Support 492

CHAPTER ONE

BUILDING OPEN RV ON CENTOS 7

1.1 Summary

1. *Install Basics*
2. *Install devtoolset-9*
3. *Install tools and build dependencies*
4. *Install CMake*
5. *Install nasm*
6. *Install Qt5*

1.2 Install Basics

Make sure we have some basic tools available on the workstation:

```
sudo yum install sudo wget git
```

1.3 Install devtoolset-9

By default the CentOS 7.9 built-in tools won't match the requirements we have to build RV. Install devtoolset-9 to remedy the situation.

```
sudo yum install centos-release-scl
sudo yum-config-manager --enable rhel-server-rhscl-7-rpms
sudo yum install devtoolset-9

# Note that you have to activate this all of the time (or add it to shell user␣
↪initialization (e.g.: .bashrc))
scl enable devtoolset-9 $SHELL
```

1.4 Install tools and build dependencies

Most of the build requirements can be installed using the following command:

```
sudo yum install alsa-lib-devel autoconf automake avahi-compat-libdns_sd-devel bison␣
↪bzip2-devel cmake-gui curl-devel flex glew-devel libXcomposite libXi-devel libaio-
↪devel libffi-devel ncurses-devel libtool libxkbcommon openssl-devel patch pulseaudio-
↪libs pulseaudio-libs-glib2 mesa-libOSMesa mesa-libOSMesa-devel ocl-icd opencl-headers␣
↪python3 python3-devel qt5-qtbase-devel readline-devel sqlite-devel tcl-devel tk-devel␣
↪yasm zlib-devel
```

1.4.1 Install the python requirements

Some of the RV build scripts requires extra python packages. They can be installed using the requirements.txt at the root of the repository.

```
python3 -m pip install -r requirements.txt
```

1.5 Install CMake

You need CMake version 3.24+ to build RV. The yum-installable version is not quite recent enough, you'll to build and install CMake from sources.

```
wget https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0.tar.gz
tar -zxvf cmake-3.24.0.tar.gz
cd cmake-3.24.0
./bootstrap --parallel=32  # 32 or whatever your machine allows
make -j 32  # 32 or whatever your machine allows
sudo make install

cmake --version  # confirm the version of you're newly installed version of CMake
cmake version 3.24.0
```

1.6 Install nasm

The `nasm` tool is required to build the `libdav1d` and `ffmpeg` dependencies. The yum-provided version is slightly too old and you'll required building from sources. Fortunately, building `nasm` from source is as easy as it gets:

```
wget https://www.nasm.us/pub/nasm/releasebuilds/2.15.05/nasm-2.15.05.tar.gz
tar xf nasm-2.15.05.tar.gz
cd nasm-2.15.05
scl enable devtoolset-9 "$SHELL -c './configure'"
scl enable devtoolset-9 "$SHELL -c 'make -j'"
sudo make install
```

1.7 Install Qt

Download the last version of Qt 5.15.x that you can get using the online installer on the Qt page. Logs, Android, iOS and WebAssembly are not required to build OpenRV.

WARNING: If you fetch Qt from another source, make sure to build it with SSL support, that it contains everything required to build PySide2, and that the file structure is similar to the official package.

CHAPTER

TWO

BUILDING OPEN RV ON ROCKY 8

2.1 Summary

1. *Install Basics*
2. *Install tools and build dependencies*
3. *Install CMake*
4. *Install Qt5*

2.2 Install Basics

Make sure we have some basic tools available on the workstation:

```
sudo dnf install wget git
```

2.3 Install tools and build dependencies

Some of the build dependencies come from outside the main AppStream repo. So first we will enable those and then install our dependencies:

```
sudo dnf install epel-release
sudo dnf config-manager --set-enabled crb devel
sudo dnf install alsa-lib-devel autoconf automake avahi-compat-libdns_sd-devel bison␣
↪bzip2-devel cmake-gui curl-devel flex gcc gcc-c++ libXcomposite libXi-devel libaio-
↪devel libffi-devel nasm ncurses-devel nss libtool libxkbcommon libXcomposite␣
↪libXdamage libXrandr libXtst libXcursor mesa-libOSMesa mesa-libOSMesa-devel meson␣
↪ninja-build openssl-devel patch pulseaudio-libs pulseaudio-libs-glib2 ocl-icd ocl-icd-
↪devel opencl-headers python3 python3-devel qt5-qtbase-devel readline-devel sqlite-
↪devel tcl-devel tcsh tk-devel yasm zip zlib-devel
```

You can disable the devel repo afterwards since dnf will warn about it:

```
sudo dnf config-manager --set-disabled devel
```

2.3.1 GLU

You may or may not have libGLU on your system depending on your graphics driver setup. To know you can check if you have the lib /usr/lib64/libGLU.so.1. If it's there you can skip this step, there's nothing further to do in this section. If not, you can install the graphics drivers for your system. As as last resort, you can install a software (mesa) libGLU version, but you may not have ideal performance with this option. To do so:

```
sudo dnf install mesa-libGLU mesa-libGLU-devel
```

2.3.2 Install the python requirements

Some of the RV build scripts requires extra python packages. They can be installed using the requirements.txt at the root of the repository.

```
python3 -m pip install -r requirements.txt
```

2.4 Install CMake

You need CMake version 3.24+ to build RV. The dnf-installable version is not quite recent enough, you'll to build and install CMake from sources.

```
wget https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0.tar.gz
tar -zxvf cmake-3.24.0.tar.gz
cd cmake-3.24.0
./bootstrap --parallel=32  # 32 or whatever your machine allows
make -j 32  # 32 or whatever your machine allows
sudo make install

cmake --version  # confirm the version of your newly installed version of CMake
cmake version 3.24.0
```

2.5 Install Qt

Download the last version of Qt 5.15.x that you can get using the online installer on the Qt page. Logs, Android, iOS and WebAssembly are not required to build OpenRV.

WARNING: If you fetch Qt from another source, make sure to build it with SSL support, that it contains everything required to build PySide2, and that the file structure is similar to the official package.

CHAPTER

THREE

BUILDING OPEN RV ON ROCKY 9

3.1 Summary

1. *Install Basics*
2. *Install tools and build dependencies*
3. *Install CMake*
4. *Install Qt5*

3.2 Install Basics

Make sure we have some basic tools available on the workstation:

```
sudo dnf install wget git
```

3.3 Install tools and build dependencies

Some of the build dependencies come from outside the main AppStream repo. So first we will enable those and then install our dependencies:

```
sudo dnf install epel-release
sudo dnf config-manager --set-enabled crb devel
sudo dnf install alsa-lib-devel autoconf automake avahi-compat-libdns_sd-devel bison␣
↪bzip2-devel cmake-gui curl-devel flex gcc gcc-c++ libXcomposite libXi-devel libaio-
↪devel libffi-devel nasm ncurses-devel nss libtool libxkbcommon libXcomposite␣
↪libXdamage libXrandr libXtst libXcursor mesa-libOSMesa mesa-libOSMesa-devel meson␣
↪ninja-build openssl-devel patch perl-FindBin pulseaudio-libs pulseaudio-libs-glib2 ocl-
↪icd ocl-icd-devel opencl-headers python3 python3-devel qt5-qtbase-devel readline-devel␣
↪sqlite-devel tcl-devel tcsh tk-devel yasm zip zlib-devel
```

You can disable the devel repo afterwards since dnf will warn about it:

```
sudo dnf config-manager --set-disabled devel
```

3.3.1 GLU

You may or may not have libGLU on your system depending on your graphics driver setup. To know you can check if you have the lib /usr/lib64/libGLU.so.1. If it's there you can skip this step, there's nothing further to do in this section. If not, you can install the graphics drivers for your system. As as last resort, you can install a software (mesa) libGLU version, but you may not have ideal performance with this option. To do so:

```
sudo dnf install mesa-libGLU mesa-libGLU-devel
```

3.3.2 Install the python requirements

Some of the RV build scripts requires extra python packages. They can be installed using the requirements.txt at the root of the repository.

```
python3 -m pip install -r requirements.txt
```

3.4 Install CMake

You need CMake version 3.24+ to build RV. The dnf-installable version is not quite recent enough, you'll to build and install CMake from sources.

```
wget https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0.tar.gz
tar -zxvf cmake-3.24.0.tar.gz
cd cmake-3.24.0
./bootstrap --parallel=32  # 32 or whatever your machine allows
make -j 32  # 32 or whatever your machine allows
sudo make install

cmake --version  # confirm the version of your newly installed version of CMake
cmake version 3.24.0
```

3.5 Install Qt

Download the last version of Qt 5.15.x that you can get using the online installer on the Qt page. Logs, Android, iOS and WebAssembly are not required to build OpenRV.

WARNING: If you fetch Qt from another source, make sure to build it with SSL support, that it contains everything required to build PySide2, and that the file structure is similar to the official package.

CHAPTER

FOUR

BUILDING OPEN RV ON MACOS

4.1 Summary

1. *Install XCode*
2. *Install Homebrew*
3. *Install tools and build dependencies*
4. *Install the python requirements*
5. *Install Qt*

4.2 Install XCode

From the App Store, download XCode 14.3.1. Make sure that it's the source of the active developer directory. Note that using an XCode version more recent than 14.3.1 will result in an FFmpeg build break.

`xcode-select -p` should return `/Applications/Xcode.app/Contents/Developer`. If it's not the case, run `sudo xcode-select -s /Applications/Xcode.app`

4.3 Install Homebrew

Homebrew is the one stop shop providing all the build requirements. You can install it following the instructions on the Homebrew page.

Make sure Homebrew's binary directory is in your PATH and that `brew` is resolved from your terminal.

4.4 Install tools and build dependencies

Most of the build requirements can be installed by running the following brew install command:

```
brew install cmake ninja readline sqlite3 xz zlib tcl-tk autoconf automake libtool␣
↪python yasm clang-format black meson nasm pkg-config glew
```

Make sure `python` resolves in your terminal. In some case, depending on how the python formula is built, there's no `python` symbolic link. In that case, you can create one with this command `ln -s python3 $(dirname $(which python3))/python`.

4.5 Install the python requirements

Some of the RV build scripts requires extra python packages. They can be installed using the requirements.txt at the root of the repository.

```
python3 -m pip install -r requirements.txt
```

4.6 Install Qt

Download the last version of Qt 5.15.x that you can get using the online installer on the Qt page. Logs, Android, iOS and WebAssembly are not required to build OpenRV.

WARNING: If you fetch Qt from another source, make sure to build it with SSL support, that it contains everything required to build PySide2, and that the file structure is similar to the official package.

Note: Qt5 from homebrew is known to not work well with OpenRV.

CHAPTER

FIVE

BUILDING OPEN RV ON WINDOWS

5.1 Summary

1. *Install Microsoft Visual Studio*
2. *Install Qt*
3. *Install Strawberry Perl*
4. *Install MSYS2*
5. *Install required MSYS2 pacman packages (from an MSYS2-MinGW64 shell)*

5.2 1. Install Microsoft Visual Studio

Install Microsoft Visual Studio 2022.

Any edition of Microsoft Visual Studio 2022 should do, even the Microsoft Visual Studio 2022 Community Edition. Download it from the Microsoft older downloads page.

Make sure to select the “Desktop development with C++” and the latest SDK for Windows 10 or Windows 11 features when installing Microsoft Visual Studio.

You select the Windows SDK based on the target Windows version you plan on running the compiled application on.

5.3 2. Install Qt

Download the last version of Qt 5.15.x that you can get using the online installer on the Qt page. Logs, Android, iOS and WebAssembly are not required to build OpenRV.

Note: You will also need `jom`, and it is included with Qt Creator (available from the Qt online installer). If you do not want to install Qt Creator, you can download it from here and copy the executable and other files into the QT installation root directory under the Tools/QtCreator/bin/jom folder.

WARNING: If you fetch Qt from another source, make sure to build it with SSL support, that it contains everything required to build PySide2, and that the file structure is similar to the official package.

Note: Qt from MSYS2 is missing QtWebEngine.

5.4 3. Install Strawberry Perl

Download and install the 64-bit version of Strawberry Perl

5.5 4. Install MSYS2

Download and install the latest MSYS2.

Note that RV is NOT a mingw64 build. It is a Microscoft Visual Studio 2022 build.

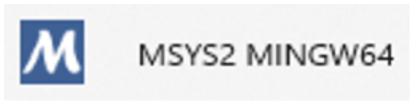
RV is built with Microsoft Visual Studio 2022 via the cmake "Visual Studio 17 2022" generator.

msys2 is only used for convenience as it comes with a package manager with utility packages required for the RV build such as cmake, git, flex, bison, nasm, unzip, zip, etc.

NOTE: The Windows' WSL2 feature conflict with MSYS2. For simplicity, it is highly recommended to disable Windows' WSL or WSL2 feature entirely.

Additional information can be found on the MSYS2 github.

5.6 5. Install required MSYS2 pacman packages



From an MSYS2-MinGW64 shell, install the following packages which are required to build RV:

```
pacman -Sy --needed \
        mingw-w64-x86_64-autotools \
        mingw-w64-x86_64-cmake \
        mingw-w64-x86_64-cmake-cmcldeps \
        mingw-w64-x86_64-glew \
        mingw-w64-x86_64-libarchive \
        mingw-w64-x86_64-make \
        mingw-w64-x86_64-meson \
        mingw-w64-x86_64-python-pip \
        mingw-w64-x86_64-python-psutil \
        mingw-w64-x86_64-toolchain \
        autoconf  \
        automake \
        bison \
        flex \
        git \
        libtool \
        nasm \
        p7zip \
        patch \
        unzip \
        zip
```

While installing the MSYS packages, review the list for any missing package. Some packages might not be installed after the first command.

Note: To confirm which version/location of any tool used inside the MSYS shell, `where` can be used e.g. `where python`. If there's more than one path return, the top one will be used.

5.6.1 Setting the PATH

The path to MSYS2's mingw64/bin folder must be added to the path. To set your PATH correctly: edit the MSYS2's ~/.bashrc file and add the following line: `PATH=/c/msys64/mingw64/bin:${PATH}`

Also add the following line to your ~/.bashrc file: `export ACLOCAL_PATH=/c/msys64/usr/share/aclocal/`

5.6.2 Python

You must use Python from mingw64 (msys64/mingw64/bin/python.exe) or your own. Therefore, you must set your PATH EnvVar correctly. Python must be BEFORE **msys64/usr/bin**.

Reminder: you must install, via pip, the requirements which are contained at the root of the project in the file requirements.txt. You must start pip from your mingw64 python executable. If there's any errors while installing via pip, see build_system/build_errors.md.

Building DEBUG

To successfully build Open RV in debug on Windows, you must also install a Windows-native python3 (download page) as it is required to build the opentimelineio python wheel in debug.

This step is not required if you do not intend to build the debug version of RV.

5.6.3 Ninja

A large part of the RV build on Windows uses Ninja. To use Ninja on MSYS, you must add **msys64/mingw64/bin** to your PATH. Ninja is installed as a dependency for `meson` hence it doesn't need to be manually installed.

5.6.4 CMake

If you have your own install of CMake on your computer, your PATH will need to pick mingw's CMake and not your own. `cmake --help` must return that the default is Ninja e.g. : `* Ninja`. Usually CMake on Windows will default to **Visual Studio** but mingw's CMake defaults to **Ninja which OpenRV is dependent on**.

Symptoms of using Windows' (and not mingw64) CMake:

- During the build, Python scripts such as quoteFile.py, make_openssl.py and make_python.py fail: either they don't work or they aren't found.
- `build.ninja not found` during the build of a dependency. (cmake/dependencies) Reason: CMake in ExternalProject_Add *configured* the Project with the Visual Studio Generator whereas OpenRV's ExternalProject_Add build command expects that Ninja will be used.

5.6.5 NOTE: Before starting a build

To maximize your chances of successfully building RV, you must:

- Fully update your code base to the latest version (or the version you want to use) with a command like `git pull`.
- Fix all conflicts due to updating the code.
- Revisit all modified files to ensure they aren't using old code that changed during the update such as when the Visual Studio version changes.

5.6.6 NOTE: Path Length

Even as of Windows 11, for legacy reason, a default system path length is still limited to 254 bytes long. For that reason we strongly suggest cloning `OpenRV` into drive's root directory e.g.: `C:\`

CHAPTER

SIX

BUILD ERRORS AND SOLUTIONS

6.1 Mu::Parse(char const*, Mu::NodeAssembler*)

```
Undefined symbols for architecture x86_64:
  "Mu::Parse(char const*, Mu::NodeAssembler*)", referenced from:
      Mu::MuLangContext::evalText(char const*, char const*, Mu::Process*, std::__
↪1::vector<Mu::Module const*, gc_allocator<Mu::Module const*> > const&) in libMuLang.
↪a(MuLangContext.cpp.o)
      Mu::MuLangContext::evalFile(char const*, Mu::Process*, std::__1::vector<Mu::Module␣
↪const*, gc_allocator<Mu::Module const*> > const&) in libMuLang.a(MuLangContext.cpp.o)
      Mu::MuLangContext::parseType(char const*, Mu::Process*) in libMuLang.
↪a(MuLangContext.cpp.o)
      Mu::MuLangContext::parseStream(Mu::Process*, std::__1::basic_istream<char, std::__
↪1::char_traits<char> >&, char const*) in libMuLang.a(MuLangContext.cpp.o)
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

6.2 Numerous undeclared identifier ‘YYMUFlexLexer’ Errors

```
FlexLexer.cpp:2429:10: error: use of undeclared identifier 'YYMUFlexLexer'
    void yyFlexLexer::yy_load_buffer_state()
         ^
```

Solution: Most likely, the YY prefix wasn’t set properly or at all

6.3 (macOS) from ranlib tool: XYZ has no symbols

```
[ 85%] Building CXX object src/lib/files/Gto/CMakeFiles/Gto.dir/Parser.cpp.o
[ 85%] Linking CXX static library libGto.a
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/
↪ranlib: file: libGto.a(FlexLexer.cpp.o) has no symbols
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/
↪ranlib: file: libGto.a(FlexLexer.cpp.o) has no symbols
[100%] Built target Gto
```

Solution: Most likely an empty source file was compiled.

6.4 Linker Error, Undefined symbols "Mu::Parse(char const*, Mu::NodeAssembler*)"

```
Undefined symbols for architecture x86_64:
  "Mu::Parse(char const*, Mu::NodeAssembler*)", referenced from:
      Mu::MuLangContext::evalText(char const*, char const*, Mu::Process*, std::__
↪1::vector<Mu::Module const*, gc_allocator<Mu::Module const*> > const&) in libMuLang.
↪a(MuLangContext.cpp.o)
      Mu::MuLangContext::evalFile(char const*, Mu::Process*, std::__1::vector<Mu::Module␣
↪const*, gc_allocator<Mu::Module const*> > const&) in libMuLang.a(MuLangContext.cpp.o)
      Mu::MuLangContext::parseType(char const*, Mu::Process*) in libMuLang.
↪a(MuLangContext.cpp.o)
      Mu::MuLangContext::parseStream(Mu::Process*, std::__1::basic_istream<char, std::__
↪1::char_traits<char> >&, char const*) in libMuLang.a(MuLangContext.cpp.o)
ld: symbol(s) not found for architecture x86_64
```

Solution: Check that Lexer/Parser compiled source aren't empty! Compiling an empty source file won't generate an error but would endup causing linker to complain du to missing code. This situation did occurred with sed command being to write-back to same file. The mitigation solution was to sed to a temporary file then rename the temporary file to the expected output filename.

6.5 Multiple 'error: unknown type name 'string' reported in system files

```
In file included from /OpenRV/src/lib/graphics/TwkGLF/GLVideoDevice.cpp:5:
In file included from /OpenRV/src/lib/graphics/TwkGLF/GLVideoDevice.h:7:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/string:519:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/__debug:14:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/iosfwd:98:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/__mbstate_t.h:29:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/wchar.h:123:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/wchar.h:91:
In file included from /OpenRV/src/lib/base/TwkMath/time.h:7:
In file included from /OpenRV/src/lib/base/TwkMath/math.h:12:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/algorithm:653:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/functional:500:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/__functional/function.h:20:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/__memory/shared_ptr.h:22:
In file included from /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.
↪platform/Developer/SDKs/MacOSX12.3.sdk/usr/include/c++/v1/__memory/allocator.h:18:
```

(continues on next page)

(continued from previous page)

```
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
↪MacOSX12.3.sdk/usr/include/c++/v1/stdexcept:83:32: error: unknown type name 'string'
    explicit logic_error(const string&);
                               ^
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
↪MacOSX12.3.sdk/usr/include/c++/v1/stdexcept:106:34: error: unknown type name 'string'
    explicit runtime_error(const string&);
                                 ^
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
↪MacOSX12.3.sdk/usr/include/c++/v1/stdexcept:126:59: error: unknown type name 'string'
    _LIBCPP_INLINE_VISIBILITY explicit domain_error(const string& __s) : logic_error(__
↪s) {}
                                                          ^
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
↪MacOSX12.3.sdk/usr/include/c++/v1/stdexcept:139:63: error: unknown type name 'string'
    _LIBCPP_INLINE_VISIBILITY explicit invalid_argument(const string& __s) : logic_
↪error(__s) {}
```

6.6 (Linux) /usr/bin/env: 'python': No such file or directory

```
env: 'python': No such file or directory
CMake Error at cmake/macros/rv_quote_file.cmake:103 (message):
  Couldn't create file from 'rv_about.html'
Call Stack (most recent call first):
  src/lib/app/RvCommon/CMakeLists.txt:58 (quote_file2)
```

Solution: We need a `python` command available.

The newer Ubuntu, Centos & Rocky system usually won't have `python` out of the box, they have a `python3` interpreter.

One solution is to instyall `pipenv` and start a local virtual environment. That will give you a `python` command.

6.7 (Linux) pyconfig.h: No such file or directory

Should be in `/home/nmontmarquette/local/cache/RV_build/RV_DEPS_DESKTOP_PACKAGE/src/Python3/Release/include/python3.9/pyconfig.h`

/home/nmontmarquette/local/cache/RV_build/RV_DEPS_DESKTOP_PACKAGE/src/Python3/Release/bin/python3.9

6.8 (Linux) error: libaio.h: No such file or directory

```
/OpenRV/src/lib/base/TwkUtil/FileStream.cpp:170:10: fatal error: libaio.h: No such file␣
↪or directory
 #include <libaio.h>
          ^~~~~~~~~~
```

Solution:

```
sudo dnf install libaio-devel
```

6.9 (Rocky Linux)

```
/usr/bin/ld: ../../../lib/mu/Mu/libMu.a(Thread.cpp.o): undefined reference to symbol
↪'pthread_attr_getstacksize@@GLIBC_2.2.5'
//usr/lib64/libpthread.so.0: error adding symbols: DSO missing from command line
collect2: error: ld returned 1 exit status
make[2]: *** [src/bin/apps/rv/CMakeFiles/rv.dir/build.make:228: src/bin/apps/rv/rv]␣
↪Error 1
make[1]: *** [CMakeFiles/Makefile2:2525: src/bin/apps/rv/CMakeFiles/rv.dir/all] Error 2
make: *** [Makefile:91: all] Error 2
```

6.10 (Linux any Distro)

```
/opt/rh/devtoolset-8/root/usr/libexec/gcc/x86_64-redhat-linux/8/ld: ../../../lib/
↪graphics/TwkGLF/libTwkGLF.a(GLFBO.cpp.o): undefined reference to symbol
↪'glBindFramebufferEXT'
//lib64/libGL.so.1: error adding symbols: DSO missing from command line
collect2: error: ld returned 1 exit status
make[3]: *** [src/bin/apps/rv/CMakeFiles/rv.dir/build.make:236: src/bin/apps/rv/rv]␣
↪Error 1
make[2]: *** [CMakeFiles/Makefile2:2497: src/bin/apps/rv/CMakeFiles/rv.dir/all] Error 2
make[1]: *** [CMakeFiles/Makefile2:2504: src/bin/apps/rv/CMakeFiles/rv.dir/rule] Error 2
make: *** [Makefile:254: rv] Error 2
[nmontmarquette@CentOS79 rv_build]$
```

Solution:

Ref.: Resolve "DSO missing from command line" error Ref.: error adding symbols: DSO missing from command line

6.11 Windows 10 & Windows 11 (Windows 10+) : PIP REQUIREMENTS: py7zr fails to install

```
  [end of output]

        note: This error originates from a subprocess, and is likely not a problem with␣
↪pip.

      See above for output.

      note: This is an issue with the package mentioned above, not pip.
      hint: See above for details.
      [end of output]

  note: This error originates from a subprocess, and is likely not a problem with pip.
```

(continues on next page)

(continued from previous page)

```
error: subprocess-exited-with-error

× pip subprocess to install build dependencies did not run successfully.
```

Note: There's an issue with the latest version of MSYS2 and Python3.xx versions. Enter the following command from the same MSYS2-MinGW64 shell if you encounter an error when installing the **py7zr** python requirement:

```
SETUPTOOLS_USE_DISTUTILS=stdlib pip install py7zr
```

6.12 Windows : System-wide vcpkg

Another installation of vcpkg can conflict with the one installed by the build system. Currently, there are only two possibles solutions:

- Option A: Remove the automatic integration of the system-wide vcpkg into cmake using :

  ```
  vcpkg integrate remove
  ```

- Option B: Make sure that the right dependencies (and version) are installed in the system-wide vcpkg.

6.13 macOS, Linux, Windows

```
[ 99%] Linking CXX executable rv
CMakeFiles/rv.dir/main.cpp.o: In function `utf8Main(int, char**)':
/OpenRV/src/bin/apps/rv/main.cpp:419: undefined reference to `qInitResources_rv()'
collect2: error: ld returned 1 exit status
make[2]: *** [src/bin/apps/rv/rv] Error 1
make[1]: *** [src/bin/apps/rv/CMakeFiles/rv.dir/all] Error 2
make: *** [all] Error 2
```

Solution:

Most likely missing a `qt5_add_resources` CMake statement. Ref.: https://forum.qt.io/topic/88959/undefined-reference-to-qinitresources/2

CHAPTER

SEVEN

CHAPTER 1 - INTRODUCTION

7.1 1.1 Overview

RV and its companion tools, RVIO and RVLS have been created to support digital artists, directors, supervisors, and production crews who need reliable, flexible, high-performance tools to review image sequences, movie files, and audio. RV is clean and simple in appearance and has been designed to let users load, play, inspect, navigate and edit image sequences and audio as simply and directly as possible. RV's advanced features do not clutter its appearance but are available through a rich command-line interface, extensive hot keys and key-chords, and smart drag/drop targets. RV can be extensively customized for integration into proprietary pipelines. The RV Reference Manual has information about RV customization.

This chapter provides quick-start guides to RV and RVIO. If you already have successfully installed RV, and want to get going right away, this chapter will show you enough to get started.

7.2 1.2 Getting Started With Open RV

7.2.1 1.2.1 Loading Media and Saving Sessions

There are four basic ways to load media into RV,

1. Command-line,
2. File open dialogs,
3. Drag/Drop, and
4. rvlink: protocol URL

RV can load individual files or multiple files (i.e. a sequence) and it can also read directories and figure out the sequences they contain; you can pass RV a directory on the command-line or drag and drop a folder onto RV. RV's ability to read directories can be particularly useful. If your shots are stored as one take per directory you can get in the habit of just dropping directories into RV or loading them on the command line. Or you can quickly load multiple sequences or movies that are stored in a single directory.

Some simple RV command line examples are:

```
shell> rv foo.mov
shell> rv [ foo.#.exr foo.aiff ]
shell> rv foo_dir/
shell> rv .
```

and of course:

```
shell> rv -help
```

The output of the -help flag is reproduced in this manual in the chapter on command-line usage, Chapter *3* .

RV sessions can be saved out as .rv files using the File->Save menus. Saved sessions contain the default views, user-defined views, color setup, compositing setup, and other settings. This is useful for reloading and sharing sessions, and also for setting up image conversion, compositing, or editing operations to be processed by RVIO.

7.2.2 1.2.2 Caching

If your image sequences are too large to play back at speed directly from disk, you can cache them into system memory using RV's *region cache* . If you are playing compressed movies like large H.264 QuickTime movies, you can use RV's *lookahead cache* to smooth out playback without having to cache the entire movie. If your IO subsystems can provide the bandwidth, RV can be used to stream large uncompressed images from disk. You can set the RV cache options from the Tools menu, using the hot keys ``Shift+C" and ``Ctrl-L" (``Command+L" on Mac) for the region cache and lookahead cache respectively, or from the command line using ``-c" or ``-l" flags. Also see the Caching tab of the Preferences dialog.

7.2.3 1.2.3 Sources and Layers

RV gives you the option to load media (image sequences or audio) as a *Source* or a *Layer* . A source is a new sequence or movie that gets added to the end of the default sequence of the RV session. Adding sources is the simplest way to build an edit in RV. Layers are the way that RV associates related media, e.g. an audio clip that goes with an EXR sequence can be added as a layer so that it plays back along with the sequence. Layers make it very simple to string together sequences with associated audio clips–each movie or image sequence can be added as source with a corresponding audio clip added as a layer (see soundfile commandline example above). RV's stereoscopic display features can interpret the first two image layers in a source as left and right views.

7.2.4 1.2.4 Open RV Views

RV provides three default views, and the ability to make views of your own. The three that all sessions have are the Default Sequence, which shows you all your sources in order, the Default Stack, which shows you all your sources stacked on top of one another, and the Default Layout, which has all the sources arranged in a grid (or a column, row, or any other custom layout of your own design). In addition to the default views, you can create any number of Sources, Sequences, Stacks, and Layouts of your own. See *5* for information about the process of creating and managing your own views.

7.2.5 1.2.5 Marking and Navigating

RV's timeline (hit TAB or F2 to bring it up) can be *marked* to make it easy to navigate around an RV session. RV can mark sequence boundaries automatically, but you can also use the ``m" key to place marks anywhere on the timeline. Once a session is marked you can use hot keys to quickly navigate the timeline, e.g. ``control+right arrow" (``command+right arrow" on Mac) will set the in/out points to the next pair of marks so you can loop over that part of the timeline. If no marks are set, many of these navigation options interpret the boundaries between sources as "virtual marks", so that even without marking you can easily step from one source to the next, etc.

7.2.6 1.2.6 Color

RV provides fine grained control over color management. Subsequent sections of this manual describe the RV color pipeline and options with a fair amount of technical detail. RV supports file LUTs and CDLs per source and an overall display LUT as well as a completely customizable 'source setup' function (described in the RV Reference Manual). For basic operation, however, you may find that the built in hardware conversions can do everything you need. A common example is playing a QuickTime movie that has baked-in sRGB together with an EXR sequence stored as linear floating point. RV can bring the QuickTime into linear space using the menu command Color->sRGB and then the whole session can be displayed to the monitor using the menu command View->sRGB.

LUTs can be loaded into RV by dragging and dropping them onto the RV window, through the File->Import menus and on the command line using the -flut, -llut, -dlut, and -pclut options. Similarly CDLs can be loaded into RV by dragging and dropping them onto the RV window, through the File->Import menus and on the command line using the -fcdl and -lcdl options.

7.2.7 1.2.7 Menus, Help and Hot Keys

Help->Show Current Bindings will print out all of RV's current key bindings to the shell or console (these are also included in chapter X of this manual). RV's menus can be reached through the menu bar or by using the right mouse button. Menus items with hot keys will display the hot key on the right side of the menu item. Some hot keys worth learning right away are:

- Space - Toggle playback
- Tab or F2 - Toggle Show Timeline
- 'i' - Toggle Show Info Widget
- '`' - (back-tick) Toggle Full Screen
- F1 - Toggle Show Menu
- Shift + Left Click - Open Pixel Inspector at pointer
- "q" to quit RV (or close the current session)

7.2.8 1.2.8 Parameter Editing and Virtual Sliders

Many settings in RV, like exposure, volume, or frame rate, can be changed quickly using Parameter Edit Mode. This mode lets you use virtual sliders, the mouse wheel or the keyboard to edit RV parameters. Hot keys and Parameter Editing Mode allow artists to easily and rapidly interact with images in RV. It is worth a little practice to get comfortable using these tools. For example, to adjust the exposure setting of a sequence you can use any of the following techniques:

1. Hit the 'e' key to enter exposure editing mode Then:
2. Click and drag left or right to vary the exposure, and then release the mouse button to leave the mode,
3. OR: Roll the mouse wheel to vary the exposure and then hit return to leave the mode.
4. OR: Hit return, type the new exposure value at the prompt, and hit return again (typing '.' or any digit also starts this text-entry mode)
5. OR: Use the '+' and '-' keys to vary the exposure and then hit return to leave the mode.

Some advanced usage:

- Use the 'r' 'g' 'b' keys to edit individual color channels. ('c' to return to editing all 3 channels.) Parameters that can be "unganged" in this way will display a 3-color glyph in the display feedback when you start editing.
- Hit the 'l' to lock (or unlock) slider mode, so that you can repeatedly set the same parameter ('ESC' to exit).

- The 'DEL' or 'BackSpace' key will reset the parameter to it's default value.
- When multiple Sources are visible, as in a Layout view, parameter sliders will affect all Sources. Or you can use 's' to select only the source under the pointer for editing.

Some parameters in RV don't use virtual sliders; you can edit these directly by entering the new value, e.g. to change the playback frames per second:

1. Hit Shift+F
2. Type in the new frame rate at the prompt and hit return

Some other useful parameters and their hot keys:

- Gamma - 'y'
- Hue - 'h'
- Contrast - 'k'
- Audio Volume - ctrl-'v' (command-'v' on Mac)

7.2.9 1.2.9 Preferences and Command Line Parameters

RV's Preferences can be opened with the RV->Preferences menu item. These are worth exploring in some detail. They give you fine control over how RV loads and displays images, handles color, manages the cache, handles audio, etc. RV's preferences map to RV's command line options, so almost any option available at the command line can be set to a preferred default value in the Preferences. RV also has a -noPrefs command line flag so that you can temporarily ignore the preferences, and a -resetPrefs flag that will reset all preferences to their default values. The quickest way to take a look at all of RV's command line options is:

```
shell> rv -help
```

7.3 1.2.10 Customizing Open RV

RV is built to be customized. For many users, this may be completely ignored or be limited to sharing startup scripts and packages created by other users. A package is a collection of script code (Mu or Python) and interface elements which can be automatically loaded into RV. A package can be installed site wide, per-show, or per-user and a command line tool (rvpkg) is included for package administration tasks.

RV Packages are discussed in Chapter *10* .

Customization is discussed in detail in the RV Reference Manual. The RV command API technical documentation can be browsed from Help->Mu Command API Browser.

7.4 1.3 Getting Started with RVIO

7.4.1 1.3.1 Converting Sequences and Audio

RVIO is a powerful pipeline tool. Like RV, the basic operation of RVIO is very simple, but advanced (and complex) operations are possible. RVIO can be used with a command line very similar to RV's, with additional arguments for specifying the output. Any number of sources and layers can be given to RVIO using the same syntax as you would use for RV. Some basic RVIO command line examples are:

```
shell> rvio foo.exr -o foo.mov
shell> rvio [ foo.#.exr foo.aiff ] -o foo.mov
shell> rvio [ foo_right.#.exr  foo_left.#.exr foo.aiff ] \
       -outstereo -o foo.mov
```

And of course:

```
shell> rvio -help
```

RVIO usage is more fully described in Chapter *16* .

7.4.2 1.3.2 Processing Open RV Session Files

RVIO can also take RV session files (.rv files) as input. RV session files can contain composites, color corrections, LUTs, CDLs, edits, and other information that might be easier to specify interactively in RV than by using the command line. RV session files can be saved from RV and then processed with RVIO. For example

```
shell> rvio foo.rv -o foo_out.#.exr
```

When rvio operates on a session file, any of the Views defined in the session file can be selected to provide rvio's output, so a single session could generate any number of different output sequences or movies, depending on which of the session's views you choose.

7.4.3 1.3.3 Slates, Mattes, Watermarks, etc.

RVIO uses Mu scripts to create slates, frame burn-in and other operations that are useful for generating dailies, client reviews and other outputs. These scripts are usable as is, but they can also be modified or replaced by users. Some examples are:

```
shell> rvio foo.#.exr -overlay watermark "For Client Review" 0.5 \
       -o foo.mov
shell> rvio foo.#.exr -leader simpleslate \
       "Tweak Films" \
       "Artist=Jane Doe" \
       "Shot=SC101_vfx_01" \
       "Notes=Lighter/Darker" \
       -o foo.mov
```

CHAPTER

EIGHT

CHAPTER 2 - INSTALLATION

Refer to the Open RV README to learn how to build and install Open RV.

If you're using network proxies or self-signed certificates, read on.

8.1 Proxy Configuration

If your network uses a proxy server, you can configure Open RV to use it by setting the following environment variables.

Environment variable	Type	Required?	Description
RV_NETWORK_PROXY_HOST	string	required	Proxy host name or IP address
RV_NETWORK_PROXY_PORT	int	required	Proxy port number
RV_NETWORK_PROXY_USER	string	optional	Username for proxy service
RV_NETWORK_PROXY_PASSWORD	string	optional	Password for proxy service

Note: If you do not set a proxy, Open RV always uses the operating system's proxy. To have Open RV to ignore all proxies (including the system's own), set the environment variable `RV_NETWORK_PROXY_DISABLE`.

8.2 About Self-Signed Certificates and Open RV

Open RV uses QtWebEngine as a browser backend, built on top of Chromium. Because of the Chromium backend, Open RV uses the system certificate store to approve SSL certificates. If your studio uses a custom certificate on the network path between Open RV and your media, you must make sure that the Certificate Authority (CA) is in the system's centralized store. Adding the certificate to Google Chrome is the easy method, but you can use any method supported by your operating system.

The Python version embedded with Open RV uses its own built-in certificates file, which stores root certificates for the most common certificate authorities. But you can have Python use your self-signed certificate by setting the path to the file in the `SSL_CERT_FILE` environment variable.

CHAPTER

NINE

CHAPTER 3 - COMMAND LINE USAGE

In this chapter, the emphasis will be on using the software from a Unix-like shell. On Windows, this can be done via Cygwin's bash shell or tcsh. If you choose to use the command.com shell, the command syntax will be the same, but some features (like pattern matching, etc) which are common with Unix shells may not work.

To use RV from a shell, you will need to have the binary executable in your path. On Linux and Windows the RV executable is located in the bin directory of the install tree.

On macOS this can be done by including /Applications/RV64.app/Contents/MacOS in your path (assuming you installed RV there). The executable on macOS is called RV64; you can either type the name in as is or create an alias or symbolic link from rv to RV.

There are a number of ways to start RV from the command line:

```
rv options
```

```
rv options source1 source2 source3 ...
```

```
rv options [ source1 source2 ... ] [ source4 source5 ... ]
```

```
rv options [ source-options source1 source2 ... ] [ source-options source4 source5 ... ]
```

```
rv file.rv
```

The command line options are not required and may appear throughout the command line. Sources are individual images, QuickTime .mov files, .avi files, audio files, image sequences or directory names. When specifying a .rv file, no other sources should be on the command line. (See Table *cap:RV-Command-Line*)

The third example above uses square brackets around groups of sources. When sources appear between brackets they are called layers. These will be discussed in more detail below.

If RV is started with no arguments, it will launch a blank window; you can later add source material to the window via file browser or drag and drop.

Options are all preceded by a dash (minus sign) in the Unix tradition, even on windows. Some of them take arguments and some of them are flags which toggle the associated feature on or off. For example:

```
shell> rv -fullscreen foo.mov
```

plays back a movie file in full screen mode. In this case -fullscreen is a toggle which takes no arguments and foo.mov is one of the source material. If an option takes arguments, you supply them directly after the option:

```
shell> rv -fps 23.97 bar.#.exr
```

Here the -fps option (frames per second) requires a single floating point number

1

A floating point number in this context means a number which may or may not have a decimal point. E.g., 10 and 10.5 are both floating point numbers.

. With rare exception, RV's options are either toggles or take a single argument.gti

The most important option to remember is -help. The help option causes RV to print out all of the options and command line syntax and can be anywhere in a command line. When unsure of what the next argument is or whether you can add more options to a long command line, you can always add -help onto the end of your command and immediately hit enter. At that point, RV will ignore the entire command and print the help out. You can always use your shell's history to get the command back, remove the -help option, and continue typing the rest of the command.

```
shell> rv -fps 30 -fullscreen -l -lram .5 -help
(rv shows help)
Usage: RV movie and image viewer
...
(hit up arrow in shell, back up over -help and continue typing)
shell> rv -fps 30 -fullscreen -l -lram .5 -play foo.mov
```

Finally, you can do some simple arithmetic on option arguments. For example if you know you want to apply an inverse gamma of 2.2 to an image to view it you could do this:

```
shell> rv -gamma 1/2.2
```

which is identical to this:

```
shell> rv -gamma 0.454545454545
```

9.1 Troubleshooting Open RV

Launching RV from the command line is the best way to troubleshoot RV by giving you access to error messages or crash dumps.

You can also access the logs at the following locations:

OS	Logs location
Linux	~/.local/share/Autodesk/RV/RV.log
macOS	~/Library/Logs/Autodesk/RV.log
Windows	%AppData%\Roaming\Autodesk\RV\RV.log

These logs contain anything that is written in the RV Console.

You can control the size and number of log files kept by RV with the following environment variables:

- `RV_FILE_LOG_SIZE`: Sets the maximum size of each log file. When that maximum size is reached, RV creates a new log file. Default value is 5 MB.
- `RV_FILE_LOG_NUM_FILES`: Sets the number of log files to keep. If there are a number of log files equal to `RV_FILE_LOG_NUM_FILES`, then RV deletes the oldest log file before creating a new one. Default value is 2.

9.1.1 Command-Line Options

-c	Use region frame cache
-l	Use look-ahead cache
-nc	Use no caching
-s float	Image scale reduction
-stereo *string*	Stereo mode (hardware, checker, scanline, anaglyph, left, right, pair, mirror, hsqueezed, vsqueezed)
-vsync int	Video Sync (1 = on, 0 = off, default = 0)
-comp *string*	Composite mode (over, add, difference, replace, default=replace)
-layout *string*	Layout mode (packed, row, column, manual)
-over	Same as -comp over -view defaultStack
-diff	Same as -comp difference -view defaultStack
-tile	Same as -comp tile -view defaultStack
-wipe	Same as -over with wipes enabled
-view *string*	Start with a particular view
-noSequence	Don't contract files into sequences
-inferSequence	Infer sequences from one file
-autoRetime *int*	Automatically retime conflicting media fps in sequences and stacks (1 = on, 0 = off, default = 1)
-rthreads int	Number of reader threads (default = 1)
-renderer *string*	Default renderer type (Composite or Direct)
-fullscreen	Start in fullscreen mode
-present	Start in presentation mode (using presentation device)
-presentAudio int	Use presentation audio device in presentation mode (1 = on, 0 = off)
-presentDevice string	Presentation mode device
-presentVideoFormat string	Presentation mode override video format (device specific)
-presentDataFormat string	Presentation mode override data format (device specific)
-screen *int*	Start on screen (0, 1, 2, . . .)
-noBorders	No window manager decorations
-geometry *int int *[_ int int _]	Start geometry x, y, w, h
-init *string*	Override init script
-nofloat	Turn off floating point by default
-maxbits *int*	Maximum default bit depth (default=32)
-gamma *float*	Set display gamma (default=1)
-sRGB	Display using linear -> sRGB conversion
-rec709	Display using linear -> Rec 709 conversion
-floatLUT *int*	Use floating point LUTs (requires hardware support, 1=yes, 0=no, default=*platform-dependant*)
-dlut *string*	Apply display LUT
-brightness *float*	Set display relative brightness in stops (default=0)
-resampleMethod *string*	Resampling method (area, linear, cube, nearest, default=area)
-eval *string*	Evaluate expression at every session start
-nomb	Hide menu bar on start up
-play	Play on startup
-fps float	Overall FPS
-cli	Mu command line interface
-vram *float*	VRAM usage limit in Mb, default = 64.000000
-cram *float*	Max region cache RAM usage in Gb
-lram *float*	Max look-ahead cache RAM usage in Gb
-noPBO	Prevent use of GL PBOs for pixel transfer
-prefetch	Prefetch images for rendering
-bwait *float*	Max buffer wait time in cached seconds, default 5.0
-lookback float	Percentage of the lookahead cache reserved for frames behind the playhead, default 25

Table 1 – continued from previous page

-yuv	Assume YUV hardware conversion
-volume float	Overall audio volume
-noaudio	Turn off audio
-audiofs *int*	Use fixed audio frame size (results are hardware dependant … try 512)
-audioCachePacket *int*	Audio cache packet size in samples (default=512)
-audioMinCache *float*	Audio cache min size in seconds (default=0.300000)
-audioMaxCache *float*	Audio cache max size in seconds (default=0.600000)
-audioModule string	Use specific audio module
-audioDevice *int*	Use specific audio device (default=-1)
-audioRate float	Use specific output audio rate (default=ask hardware)
-audioPrecision int	Use specific output audio precision (default=16)
-audioNice *int*	Close audio device when not playing (may cause problems on some hardware) default=0
-audioNoLock int	Do not use hardware audio/video synchronization (use software instead default=0)
-audioGlobalOffset int	Global audio offset in seconds
-bg string	Background pattern (default=black, grey18, grey50, checker, crosshatch)
-formats	Show all supported image and movie formats
-cmsTypes	Show all available Color Management Systems
-debug *string*	Debug category
-cinalt	Use alternate Cineon/DPX readers
-exrcpus *int*	EXR thread count (default=2)
-exrRGBA	EXR use basic RGBA interface (default=false)
-exrInherit	EXR guesses channel inheritance (default=false)
-exrIOMethod int [int]	EXR I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=As
-jpegRGBA	Make JPEG four channel RGBA on read (default=no, use RGB or YUV)
-jpegIOMethod int [int]	JPEG I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=As
-cinpixel *string*	Cineon/DPX pixel storage (default=RGB8_PLANAR)
-cinchroma	Cineon pixel storage (default=RGB8_PLANAR)
-cinIOMethod int [int]	ineon I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=As
-dpxpixel string	DPX pixel storage (default=RGB8_PLANAR)
-dpxchroma	Use DPX chromaticity values (for default reader only)
-dpxIOMethod int [int]	DPX I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=As
-tgaIOMethod int [int]	TARGA I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=
-tiffIOMethod int [int]	TIFF I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=As
-noPrefs	Ignore preferences
-resetPrefs	Reset preferences to default values
-qtcss *string*	Use QT style sheet for UI
-qtstyle *string*	Use QT style, default=””
-qtdesktop	QT desktop aware, default=1 (on)
-xl	Aggressively absorb screen space for large media
-mouse int	Force tablet/stylus events to be treated as a mouse events, default=0 (off)
-network	Start networking
-networkPort int	Port for networking
-networkHost *string*	Alternate host/address for incoming connections
-networkConnect *string *[int]	Start networking and connect to host at port
-networkPerm int	Default network connection permission (0=Ask, 1=Allow, 2=Deny, default=0)
-reuse int	Try to re-use the current session for incoming URLs (1 = reuse session, 0 = new session, default = 1
-nopackages	Don’t load any packages at startup (for debugging)
-encodeURL	Encode the command line as an rvlink URL, print, and exit
-bakeURL	Fully bake the command line as an rvlink URL, print, and exit
-flags *string*	Arbitrary flags (flag, or ‘name=value’) for use in Mu code

Table 1 – continued from previous page

-prefsPath *string*	Alternate path to preferences directory
-registerHandler	Register this executable as the default rvlink protocol handler (macOS only)
-scheduler *string*	Thread scheduling policy (may require root, Linux only)
-priorities int int	Set display and audio thread priorities (may require root, Linux only)
-version	Show RV version number
-pa float	Set the Pixel Aspec Ratio
-ro int	Shifts first and last frames in the source range (range offset)
-rs int	Sets first frame number to argument and offsets the last frame number
-fps float	FPS override
-ao float	Audio Offset. Shifts audio in seconds (audio offset)
-so float	Set the Stereo Eye Relative Offset
-volume float	Audio volume override (default = 1)
-fcdl filename	Associate a file CDL with the source
-lcdl filename	Associate a look CDL with the source
-flut filename	Associate a file LUT with the source
-llut filename	Associate a look LUT with the source
-pclut *filename*	Associate a pre-cache software LUT with the source
-cmap *channels*	Remap color channels for this source (channel names separated by commas)
-select *selectType selectName*	Restrict loaded channels to a single view/layer/channel.* selectType* must be one of view, layer, or
-crop x0 y0 x1 y1	Crop image to box (all integer arguments)
-uncrop width height x y	Inset image into larger virtual image (all integer arguments)
-in int	Cut-in frame for this source in default EDL
-out int	Cut-out frame for this source in default EDL
-noMovieAudio	Turn off source movie's baked-in audio (aka "-nma")

Table 3.1: Per-Source Command Line Options

9.2 3.1 Image Sequence Notation

RV has a special syntax to describe image sequences as source movies. Sequences are assumed to be files with a common base name followed by a frame number and an image type extension. For example, the filenames foo.1.tif and foo.0001.tif would be interpreted as frame 1 of the TIFF sequence foo. RV sorts images by frame numbers in numeric order. It sorts image base names in lexical order. What this means is that RV will sort images into sequences the way you expect it to. Padding tricks are unnecessary for RV to get the image order correct; image order will be interpreted correctly.

```
foo.0001.tif foo.0002.tif foo.0003.tif foo.0004.tif
foo.0005.tif foo.0006.tif foo.0007.tif foo.0008.tif
foo.0009.tif foo.0010.tif
```

To play this image sequence in RV from the command line, you could start RV like this:

```
shell> rv foo.\*.tif
```

and RV will automatically attempt to group the files into a logical movie. (**Note** : this will only work on Linux or Mac OS, or some other Unix-like shell, like cygwin on Windows.)

When you want to play a subset of frames or audio needs to be included, you can specify the sequence using the ``#" or ``@" notation (similar to Shake's) or the printf-like notation using ``%" similar to Nuke.

```
rv foo.#.tif
rv foo.2-8#.tif
rv foo.2-8@@@@.tif
rv foo.%04d.tif
rv foo.%04d.tif 2-8
rv foo.#.tif 2-8
```

The first example above plays all frames in the foo sequence, the second line plays frames starting at frame 2 through frame 8. The third line uses the ``@” notation which forces RV to assume 0 padded frame numbers–in this case, four ``@” characters indicate a four character padding.

The next two examples use the printf-like syntax accepted by nuke. In the first case, the entire frame range is specified with the assumption that the frame numbers will be padded with 0 up to four characters (this notation will also work with 6 or other amounts of padding). In the final two examples, the range is limited to frames 2 through 8, and the range is passed as a separate argument.

Sometimes, you will encounter or create an image sequence which is purposefully missing frames. For example, you may be rendering something that is very expensive to render and only wish to look at every tenth frame. In this case, you can specify the increment using the ``x” character like this:

```
rv foo.1-100x10#.tif
```

or alternately like this using the ``@” notation for padding to four digits:

```
rv foo.1-100x10@@@@.tif
```

or if the file was padded to three digits like foo.001.tif:

```
rv foo.1-100x10@@@.tif
```

In these examples, RV will play frames 1 through 100 by tens. So it will expect frames 1, 11, 21, 31, 41, on up to 91.

If there is no obvious increment, but the frames need to be group into a sequence, you can list the frame numbers with commas:

```
rv foo.1,3,5,7,8,9#.tif
```

In many cases, RV can detect file types automatically even if a file extension is not present or is mis-labeled.

NOTE : Use the same format for exporting multiple annotated frames.

9.2.1 3.1.1 Negative Frame Numbers

RV can handle negative frames in image sequences. If the frame numbers are zero padded, they should look like so:

```
foo.-012.tif
foo.-001.tif
```

To specify in and out points on the command line in the presence of negative frames, just include the minus signs:

```
foo.-10-20#.tif
foo.-10--5#.tif
```

The first example uses frames -10 to +20. The second example uses frames -10 to -5. Although the use of the ``-” character to specify ranges can make the sequence a bit visually confusing, the interpretation is not ambiguous.

9.2.2 3.1.2 Stereo Notation

RV can accept stereo notation similar to Nuke's ``%v" and ``%V" syntax. By default, RV can only recognize left, right, Left, and Right for %V and for %v it will try L, R, or l and r. You can change the substitutions by setting the environment variables RV_STEREO_NAME_PAIRS and RV_STEREO_CHAR_PAIRS. These should be set to a colon separated list of values (even on windows). For example, the defaults would look like this:

```
RV_STEREO_NAME_PAIRS = left:right:Left:Right
RV_STEREO_CHAR_PAIRS = L:R:l:r
```

So for example, if you have two image sequences:

```
foo.0001.left.exr
foo.0002.left.exr
foo.0001.right.exr
foo.0002.right.exr
```

you could refer to the entire stereo sequence as:

```
foo.%04d.%V.exr
```

9.3 3.2 Source Layers from the Command Line

You can create source material from multiple audio, image, and movie files from the command line by enclosing them in square brackets. The typical usage looks something like this:

```
shell> rv [ foo.#.exr soundtrack.aiff ]
```

Note that there are spaces around the brackets: they must be completely separated from subsequent arguments to be interpreted correctly. You cannot nest brackets and the brackets must be matched; for every begin bracket there must be an end bracket.

9.3.1 3.2.1 Associating Audio with Image Sequences or Movie Files

Frequently a movie file or image sequence needs to be viewed with one or more separate audio files. When you have multiple layers on the command line and one or more of the layers are audio files, RV will play back the all of the audio files mixed together along with the images or movies. For example, to play back a two wav files with an image sequence:

```
shell> rv [ foo.#.exr first.wav second.wav ]
```

If you have a movie file which already has audio you can still add additional audio files to be played:

```
shell> rv [ movie_with_audio.mov more_audio.aiff ]
```

9.3.2 3.2.2 Dual Image Sequences and/or Movie Files as Stereo

It's not unusual to render left and right eyes separately and want to view them as stereo together. When you give RV multiple layers of movie files or image sequences, it uses the first two as the left and right eyes.

```
shell> rv [ left.#.exr right.#.exr ]
```

It's OK to mix and match formats with layers:

```
shell> rv [ left.mov right.100-300#.jpg ]
```

if you want audio too:

```
shell> rv [ left.#.exr right.#.exr soundtrack.aiff ]
```

As with the mono case, any number of audio files can be added: they will be played simultaneously.

9.3.3 3.2.3 Per-Source Arguments

There are a few arguments which can be applied with in the square brackets (See Table *3.1*). The range start sets the first frame number to its argument; so for example to set the start frame of a movie file with or without a time code track so that it starts at frame 101:

```
shell> rv [ -rs 101 foo.mov ]
```

You must use the square brackets to set per-source arguments (and the square brackets must be surrounded by spaces).

The -in and -out per-source arguments are an easy way to create an EDL on the command line, even when playing movie files.

9.3.4 3.2.4 A note on the -fps per-source argument

The point of the -fps arg is to provide a scaling factor in cases where the frame rate of the media cannot be determined and you want to play an audio file with it. For example, if you want to play "[foo.#.dpx foo.wav]" in the same session with "[bar.#.dpx bar.wav]" but the "native frame rate of foo is 24fps and the native frame rate of bar is 30fps, then you might want to say:

```
shell> rv [foo.#.dpx foo.wav -fps 24 ] [ bar.#.dpx bar.wav -fps 30 ]
```

This will ensure that the video and audio are synced properly no matter what frame rate you use for playback. To clarify further, the per-source -fps flag has no relation to the frame rate that is used for playback, and in general RV plays media (all loaded media) at whatever single frame rate is currently in use.

9.3.5 3.2.5 Source Layer Caveats and Capabilities

There are a number of things you should be aware of when using source layers. In most cases, RV will attempt to do something with what you give it. However, if your input is logically ambiguous the result may be different than what you expect. Here are some things you should avoid using in layers of a single source:

- Images or movies with differing frame rates
- Images or movies with different image or pixel aspect ratios
- Images or movies which require special color correction for only one eye

- Images or movies stored with differing color spaces (e.g. cineon log images + jpeg)

Here are some things that are OK to do with layers:

- Images with different resolutions but the same image aspect ratio
- Images with different bit depths or number of channels or chroma sampling
- Audio files with different sample rates or bit depths
- Mixing movies with audio and separate audio files
- An image sequence for one eye and a movie for the other
- A movie with audio for one eye and a movie without audio for the other
- Audio files with no imagery

9.4 3.3 Directories as Input

If you give RV the name of a directory instead of a single file or an image sequence it will attempt to interpret the contents of the directory. RV will find any single images, image sequences, or single movie files that it can and present them as individual source movies. This is especially useful with the directory ``.” on Linux and macOS. If you navigate in a shell to a directory that contains an image sequence for example, you need only type the following to play it:

```
rv .
```

You don’t even need to get a directory listing. If RV finds multiple sequences or a sequence and movie files, it will sort and organize them into a playlist automatically. RV will attempt to read files without extensions if they look like image files (for example the file ends in a number). If RV is unable to parse the contents of a directory correctly, you will need to specify the image sequences directly to force it to read them.

CHAPTER

TEN

CHAPTER 4 - USER INTERFACE

The goal of RV's user interface is to be minimal in appearance, but complete in function. By default, RV starts with no visible interface other than a menu bar and the timeline, and even these can be turned off from the command line or preferences. While its appearance is minimal, its interaction is not: almost every key on the keyboard does something and it's possible to use key-chords and prefix-keys to extend this further.

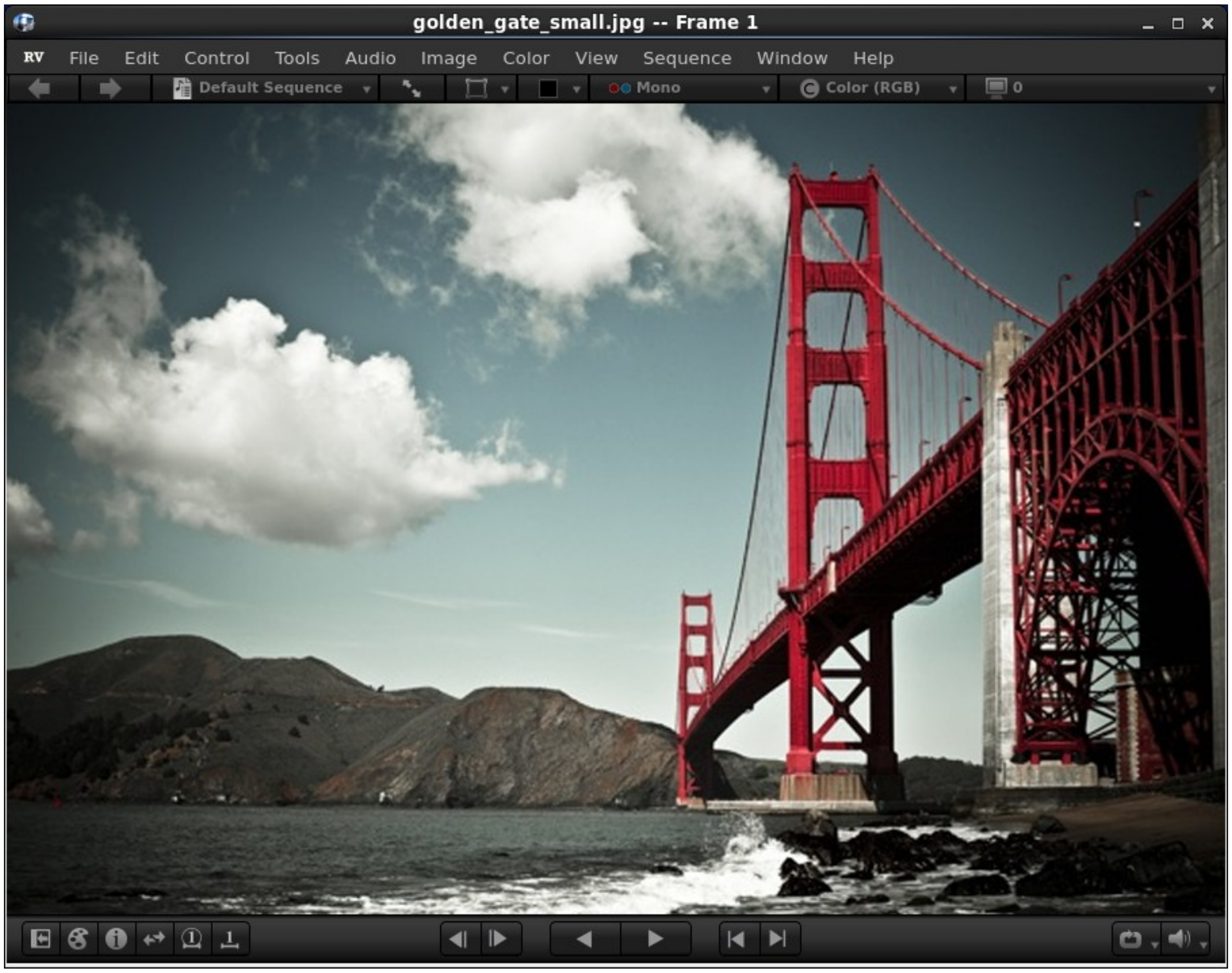
Emacs users will find this feature familiar. RV can have prefix-keys that when pressed remap the entire keyboard or mouse bindings or both.

The main menu and pop-up menus allow access to most functions and provide hot keys where available.

RV makes one window per session. Each window has two main components: the viewing area—where images and movies are shown— and the menu bar. On macOS, the menu bar will appear at the top of the screen (like most native macOS applications). On Linux and Windows, each RV window has its own attached menu bar.

A single RV process can control multiple independent sessions on all platforms. On the Mac and Windows, it is common that there be only a single instance of RV running. On Linux it is common to have multiple separate RV processes running.

Many of the tools that RV provides are heads up widgets. The widgets live in the image display are or are connected to the image itself. Aside from uniformity across platforms, the reason we have opted for this style of interface was primarily to make RV function well when in full screen mode.


golden_gate_small.jpg -- Frame 1
RV
File
Edit
Control
Tools
Audio
Image
Color
View
Sequence
Window
Help
Default Sequence
Mono
Color (RGB)
0

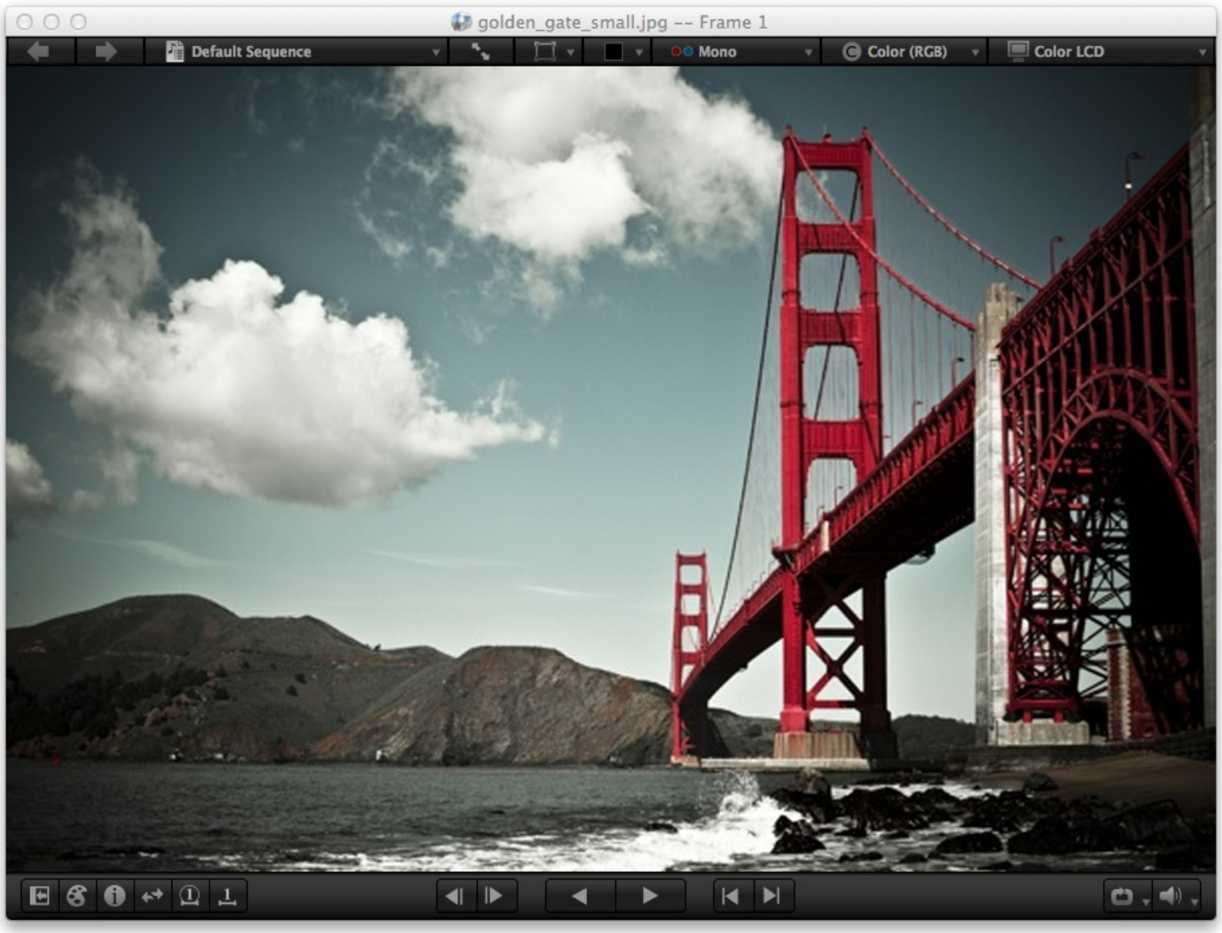


Table 4.1: RV on Linux and on macOS.

10.1 4.1 Feedback

RV provides feedback about its current state near the top left corner of the window.

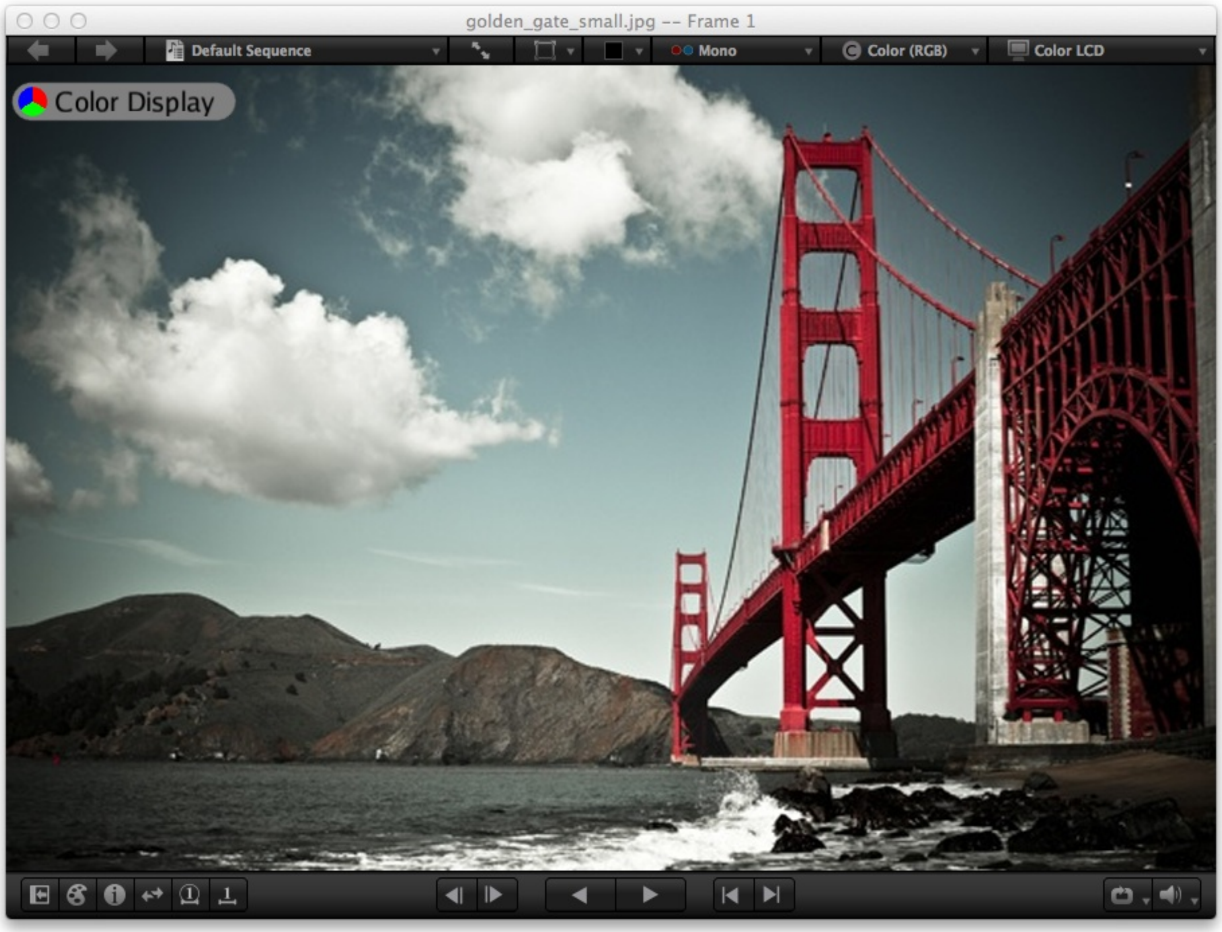


Figure 4.1: Feedback widget indicating full-color display

10.2 4.2 Main Window Tool Bars

The main RV window has two toolbars which are visible by default. The upper toolbar controls which view is displayed, viewing options, and current display device settings. The lower toolbar control play back, has tool buttons to show more functions, and audio controls.

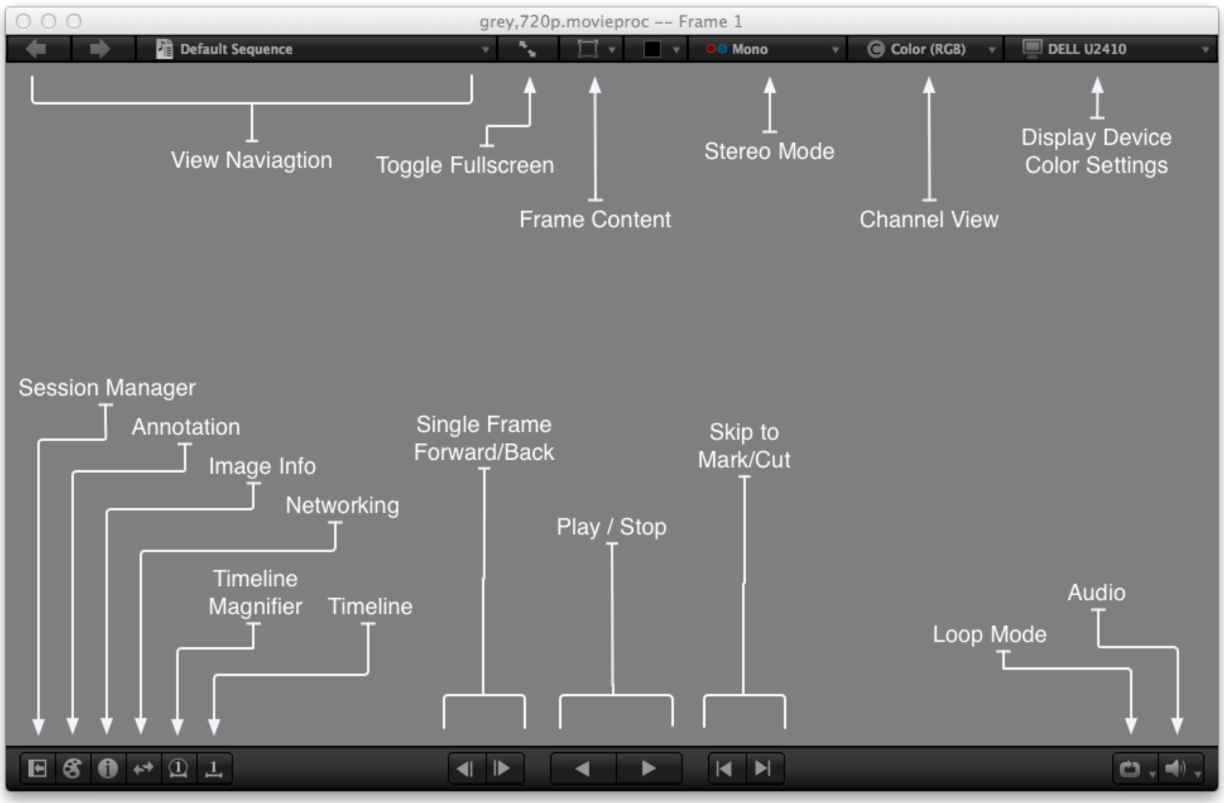


Figure 4.2:

Tool Bar Controls

The lower toolbar is in three sections from left to right: tool launch buttons, play controls, and audio/loop mode. The tool launch buttons toggle rv's main user interface components like the session manager or the heads-up timeline. The play controls control play back in the current view. These are similar to the heads-up play controls available from the timeline configuration. The loop mode determines what happens at the end of the timeline and the audio controls modify the volume and mute.

The frame content, display device settings, channel view, and stereo mode on the top tool bar are also available under the View menu. See Chapter *7* for more information on what these settings do.

The full screen toggle is also under the Window menu.

You can toggle the visibility for each of the tool bars under the Tools menu.

10.3 4.3 Loading Images, Sequences, Movies and Audio

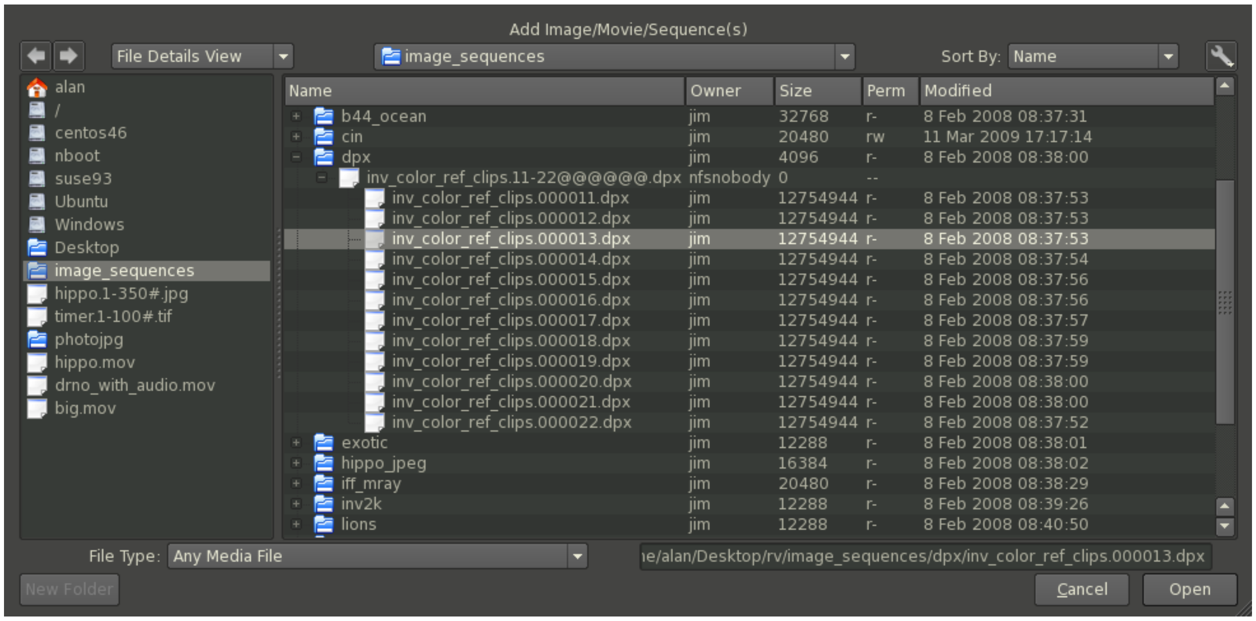
10.3.1 4.3.1 Using the File Browser

There are two options for loading images, sequences, and movies via the file browser: you can add to the existing session by choosing File → Open (or File → Open into Layer) or you can open images in a new window by choosing File → Open In New Session.

In the file browser, you may choose multiple files. RV tries to detect image sequences from the names of image files (movie files are treated as individual sequences). If RV detects a pattern, it will create an image sequence for each unique pattern. If no pattern is found, each individual image will be its own sequence.

Audio files can be loaded into RV using the file browser. To associate an audio file with its corresponding image sequence or movie open the audio file as a layer using File → Open into Layer. The first sequence in the layer will determine the overall length of the source, e.g. a longer audio file loaded as a layer after a sequence will be truncated to the duration of the sequence. Any number of Audio files can be added as layers to the same source and they will be mixed together on playback.

The file browser has three file display modes: column view, file details view, and media details view. Sequences of images appear as virtual directories in the file browser: you can select the entire sequence or individual files if you open the sequence up. **Note:** You can multi-select in File Details and Media Details, but not in Column View. In general the File Details view will be the fastest.



File Details View | image_sequences | Sort By: Name

alan
/
centos46
nboot
suse93
Ubuntu
Windows
Desktop
image_sequences
hippo.1-350#.jpg
timer.1-100#.tif
photojpg
hippo.mov
drno_with_audio.mov
big.mov

Name	Owner	Size	Perm	Modified
b44_ocean	jim	32768	r-	8 Feb 2008 08:37:31
cin	jim	20480	rw	11 Mar 2009 17:17:14
dpx	jim	4096	r-	8 Feb 2008 08:38:00
inv_color_ref_clips.11-22@@@@@@.dpx	nfsnobody	0	--	
inv_color_ref_clips.000011.dpx	jim	12754944	r-	8 Feb 2008 08:37:53
inv_color_ref_clips.000012.dpx	jim	12754944	r-	8 Feb 2008 08:37:53
inv_color_ref_clips.000013.dpx	jim	12754944	r-	8 Feb 2008 08:37:53
inv_color_ref_clips.000014.dpx	jim	12754944	r-	8 Feb 2008 08:37:54
inv_color_ref_clips.000015.dpx	jim	12754944	r-	8 Feb 2008 08:37:56
inv_color_ref_clips.000016.dpx	jim	12754944	r-	8 Feb 2008 08:37:56
inv_color_ref_clips.000017.dpx	jim	12754944	r-	8 Feb 2008 08:37:57
inv_color_ref_clips.000018.dpx	jim	12754944	r-	8 Feb 2008 08:37:59
inv_color_ref_clips.000019.dpx	jim	12754944	r-	8 Feb 2008 08:37:59
inv_color_ref_clips.000020.dpx	jim	12754944	r-	8 Feb 2008 08:38:00
inv_color_ref_clips.000021.dpx	jim	12754944	r-	8 Feb 2008 08:38:00
inv_color_ref_clips.000022.dpx	jim	12754944	r-	8 Feb 2008 08:37:52
exotic	jim	12288	r-	8 Feb 2008 08:38:01
hippo_jpeg	jim	16384	r-	8 Feb 2008 08:38:02
iff_mray	jim	20480	r-	8 Feb 2008 08:38:29
inv2k	jim	12288	r-	8 Feb 2008 08:39:26
lions	jim	12288	r-	8 Feb 2008 08:40:50

File Type: Any Media File

ıe/alan/Desktop/rv/image_sequences/dpx/inv_color_ref_clips.000013.dpx

Figure 4.3: File Browser Show File Details

Favorite locations can be remembered by dragging directories from the main part of the file browser to the side bar on the left side of the dialog box. Recent items and places can be found under the path combo box. You can configure the way the browser uses icons from the preferences drop down menu on the upper right of the browser window.

10.3.2 4.3.2 Dragging and Dropping

On all platforms, you can drag and drop file and folder icons into the RV window. RV will correctly interpret sequences that are dropped (either as multiple files or inside of a directory folder that is dropped). LUT and CDL files can also be dropped.

RV uses smart drop targets to give you control over how files are loaded into RV. You can drop files as a *source* or as a *layer* . As you drag the icons over the RV window, the drop targets will appear. Just drop onto the appropriate one.

On Linux RV should be compatible with the KDE and Gnome desktops. It is possible with these desktops (or any that supports the XDnD protocol) to drag file icons onto an RV window.

If multiple icons are dropped onto RV at the same time, the order in which the sequences are loaded is undefined.

To associate an audio file with an image sequence or movie, drop the audio file as a layer, rather than as a source.

10.4 4.4 Examining an Image

RV normalizes image geometry to fit into its viewing window. If you load two files containing the same image but at different resolutions, RV will show you the images with the same apparent ``size”. So, for example, if these images are viewed as a sequence — one after another — the smaller of the two images will be scaled to fit the larger. Of course, if you zoom in on a high-resolution image, you will see detail compared to a lower-resolution image. When necessary you can view the image scaled so that one image pixel is mapped to each display pixel.

On startup, RV will attempt to size the window to map each pixel to a display pixel, but if that is not possible, it will settle on a smaller size that fits. You can always set the scale to 1:1 with the ‘1’ hotkey, and if it is possible to resize the window to contain the entire image at 1:1 scaling, you can do so via the Window → Fit or Window → Center Fit menu items.

10.4.1 4.4.1 Panning, Zooming, and Rotating

You can manipulate the Pan and Zoom of the image using the mouse or the row of number keys (or the keypad on an extended keyboard if numlock is on). By holding down the control key (apple key on a mac) and the left mouse button you can zoom the image in or out by moving to the left and right. By holding down the alt key (or option key on a mac) you can pan the image in any direction. If you are accustomed to Maya camera bindings, you can use those as well.

Rotating the image is accessible from the Image → Rotation menu. By selecting Image → Rotation → Arbitrary it’s possible to use the mouse to scrub the rotation as a parameter.

To frame the image — automatically pan and zoom it to fit the current window dimensions — hit the ‘f’ key. If the image has a rotation, it will remain rotated.

To precisely scale the image you can use the menu Image → Scale to apply one of the preset scalings. Selected 1:1 will draw one image pixel at every display pixel. 2:1 will draw 1 image pixel as 4 display pixels, etc.

10.4.2 4.4.2 Inspecting Pixel Values

The pixel inspector widget can be accessed from the Toools menu or by holding down Shift and clicking the left mouse button. The inspector will appear showing you the source pixel value at that point on the image (see Figure *cap:Color-Inspector*). If you drag the mouse around over the image while holding down the shift key the inspector widget will also show you an average value (see Figure *cap:Average-Color*). You can move the widget by clicking on it and dragging.

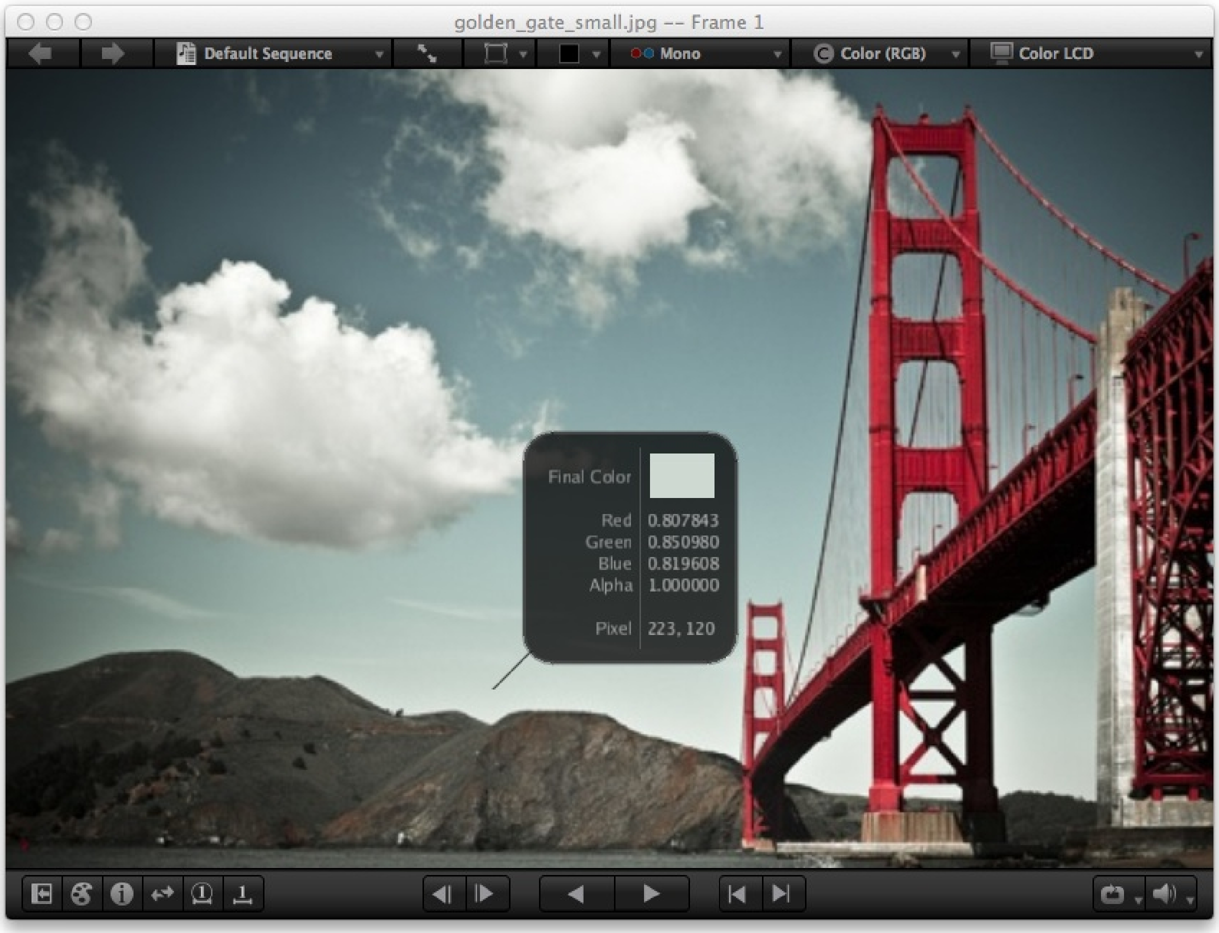
To remove inspector widget from the view either movie the mouse to the top left corner of the widget and click on the close button that appears or toggle the display with the Tools menu item or hot key.

The inspector widget is locked to the image. If you pan, zoom, flip, flop, or rotate the image, the inspector will continue to point to the last pixel read.

If you play a sequence of images with the inspector active, it will average the pixel values over time. If you drag the inspector while playback occurs it will average over time and space.

RV shows either the source pixel values or the final rendered values. The source value represents the value of the pixel just after it is read from the file. No transforms have been done on the pixel value at that point. You can see the final pixel color (the value after rendering) by changing the pixel view to the final pixel value from the right-click popup menu.

The value is normalized if the image is stored as non-floating point — so values in these types of images will be restricted to the [0,1] range. Floating point images pass the value through unchanged so pixels can take values below zero or above one. Table *7.3* shows the range of each of the channel data types.

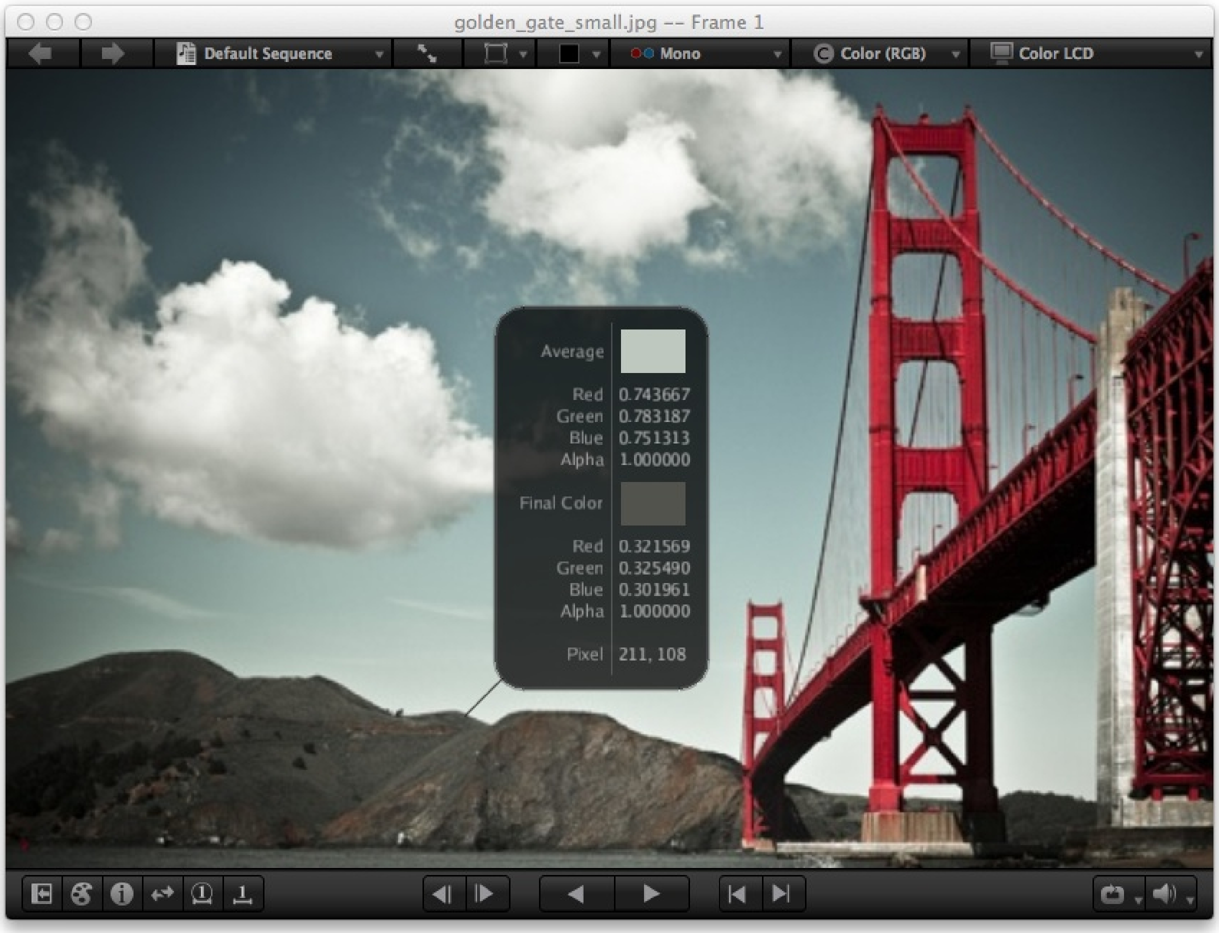


Final Color

Red 0.807843
Green 0.850980
Blue 0.819608
Alpha 1.000000

Pixel 223, 120

Figure 4.6: Color inspector



Average	
Red	0.743667
Green	0.783187
Blue	0.751313
Alpha	1.000000
Final Color	
Red	0.321569
Green	0.325490
Blue	0.301961
Alpha	1.000000
Pixel	211, 108

Figure 4.7: Average Color

From the right-click popup menu it's possible to view the pixel values as normalized to the [0,1] range or as 8, 16, 10, or 12 bit integer values.

10.4.3 4.4.3 Comparing Images with Wipes

Wipes allow you to compare two or more images or sequences when viewing a stack. Load the images or sequences that you wish to compare into RV as sources (not as layers). Put RV into stack mode or create a stack view from the Session Manager, and enable wipes from the Tools menu or with the "F6" hot key. Now you can grab on the edges of the top image and wipe them back to reveal the image below. You can grab any edge or corner, and you can move the entire wipe around by grabbing it in the exact center. Also, by clicking on the icon that appears at the center or corner of a wipe or via the Wipes menu, you can enable the wipe information mode, that will indicate which edge you are about to grab.

Wipes can be used with any number of sources. The stack order can be cycled using the "(" and ")" keys.

10.4.4 4.4.4 Parameter Edit Mode and Virtual Sliders

RV has a special UI mode for editing the parameters such as color corrections, volume, and image rotation. For example, hitting the ``y" key enters gamma edit mode. When editing parameters, the mouse and keyboard are bound to a different set of functions. On exiting the editing mode, the mouse and keyboard revert to the usual bindings. (See Table *4.2*)

To edit the parameter value using the mouse you can either scrub (like a virtual slider) or use the wheel. If you want to eyeball it, hold the left mouse down and scrub left and right. By default, when you release the button, the edit mode will be finished, so if you want to make further changes you need to re-enter the edit mode. If you want to make many changes to the same parameter, you can "lock" the mode with the 'l' key. The scroll wheel increments and decrements the parameter value by a predefined amount. Unlike scrubbing with the left mouse button, the scroll wheel will not exit the edit mode. When multiple Sources are visible, as in a Layout view, parameter sliders will affect all Sources. Or you can use 's' to select only the source under the pointer for editing. You can exit the edit mode by hitting the escape key or space bar (or most other keys).

To change the parameter value using the keyboard, hit the Enter (or Return) key; RV will prompt you for the value. For interactive changes from the keyboard, use the ``+" and ``-" keys (with or without shift held down). The parameter is incremented and decremented. To end the keyboard interactive edit, hit the Escape or Spacebar keys.

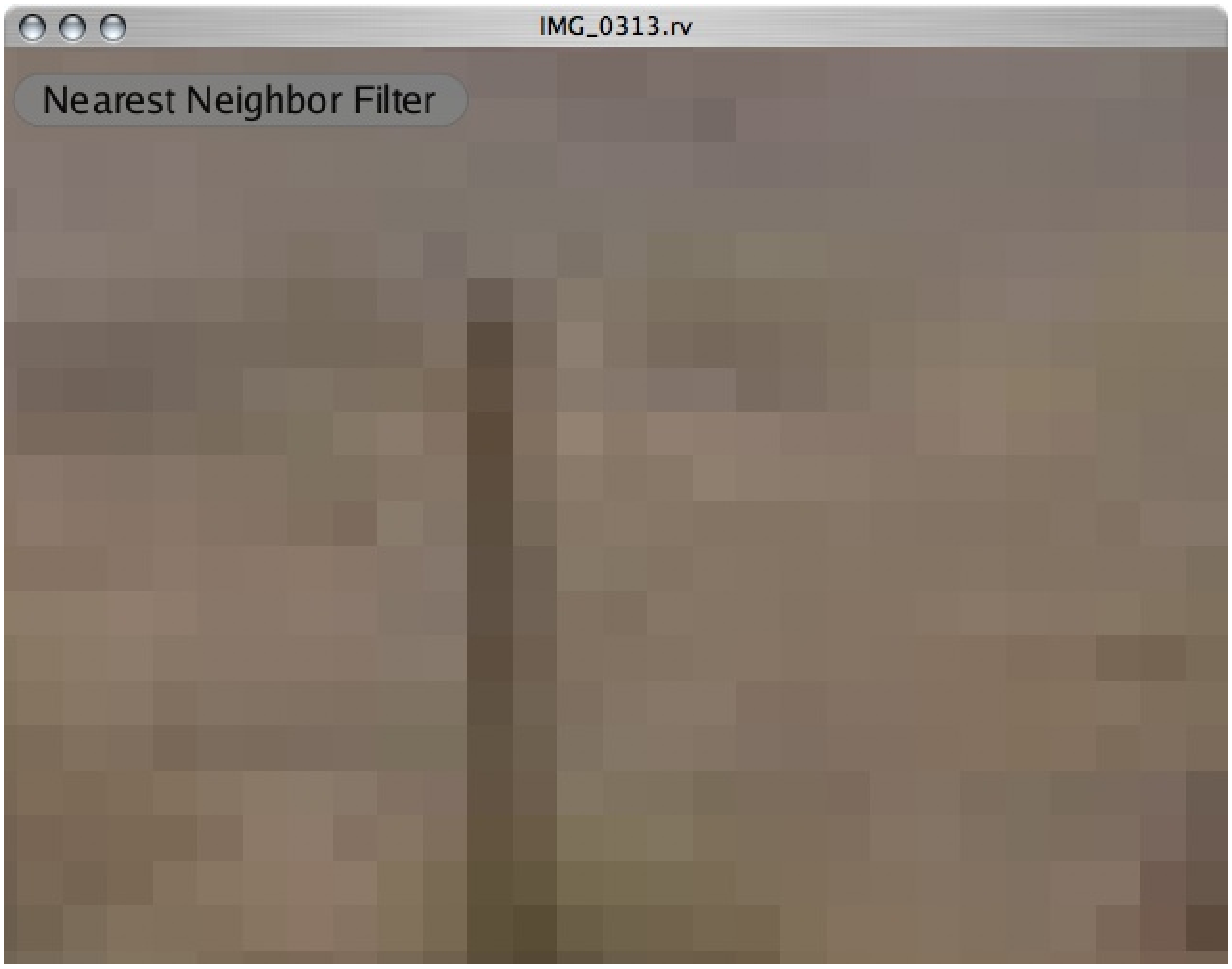
Key/Mouse Sequence	Action
Mouse Button #1 Drag	Scrub parameter
Mouse Button #1 Up	Finish parameter edit
Wheel	Increment or decrement parameter
Enter	Enter parameter numerically
0 through 9	Enter parameter numerically
ESC	Cancel parameter edit mode
+ or =	Increment parameter value
- or _	Decrement parameter value
BACKSPACE or DEL	Reset parameter value to default
r or g or b	Edit single channel of color parameter
c	Edit all channels of color parameter
l	Lock or unlock editing mode
s	Select single Source for editing

Table 4.2: Parameter Edit Mode Key and Mouse Bindings

10.4.5 4.4.5 Image Filtering

When image pixels are scaled to be larger or smaller than display pixels, resampling occurs. When the image is scaled (zoomed) RV provides two resampling methods (filters): nearest neighbor and linear interpolation (the default).

You can see the effects of the resampling filters by making the scale greater than 1:1. This can be done with any of the hot keys ``2" through ``8" or by zooming the image interactively. When the image pixels are large enough, you can switch the sampling method via View → Linear Filter or by hitting the ``n" key. Figure *4.3* shows an example of an image displayed with nearest neighbor and linear filtering.


IMG_0313.rv
Nearest Neighbor Filter

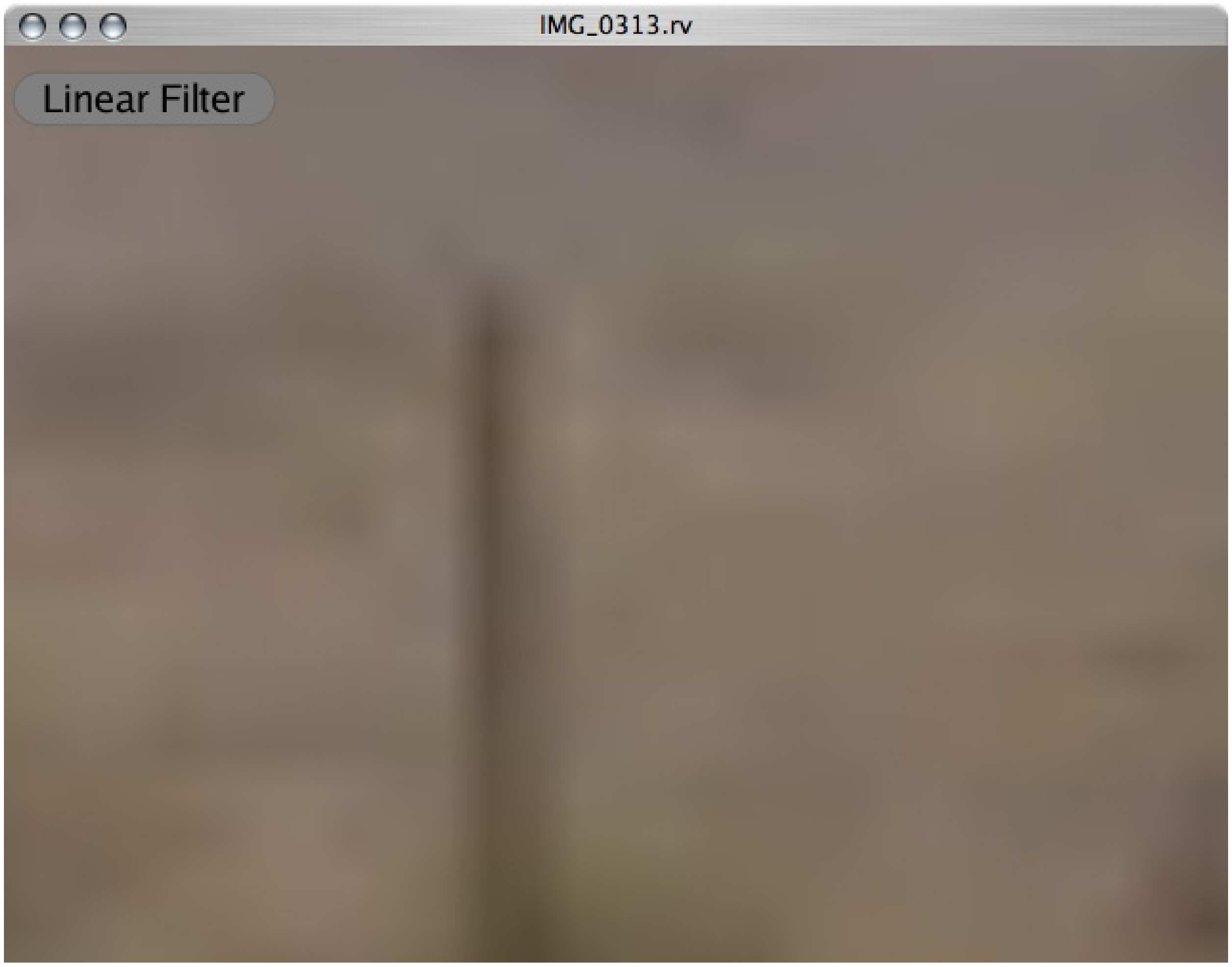


Table 4.3: Nearest Neighbor and Linear Interpolation Filtering. Nearest neighbor filtering makes pixels into blocks (helpful in trying to determine an exact pixel value).

Digression on Resampling

It's important to know about image filtering because of the way in which RV uses the graphics hardware. When an image is resampled—as it is when zoomed in—and the resampling method produces interpolated pixel values, correct results are really only obtained if the image is in linear space. Because of the way in which the graphics card operates, image resizing occurs before operations on color. This sequence can lead to odd results with non-linear space images if the linear filter is used (e.g., cineon files).

If you want to put a positive spin on it you could say you're using a non-linear resampling method on purpose. The results are only incorrect if you meant to do something else!

There are two solutions to the problem: use the nearest neighbor filter or convert the image to linear space before it goes to the graphics card. The only downside with the second method is that the transform must happen in software which is usually not as fast. Of course this only applies to images that are not already in linear space.

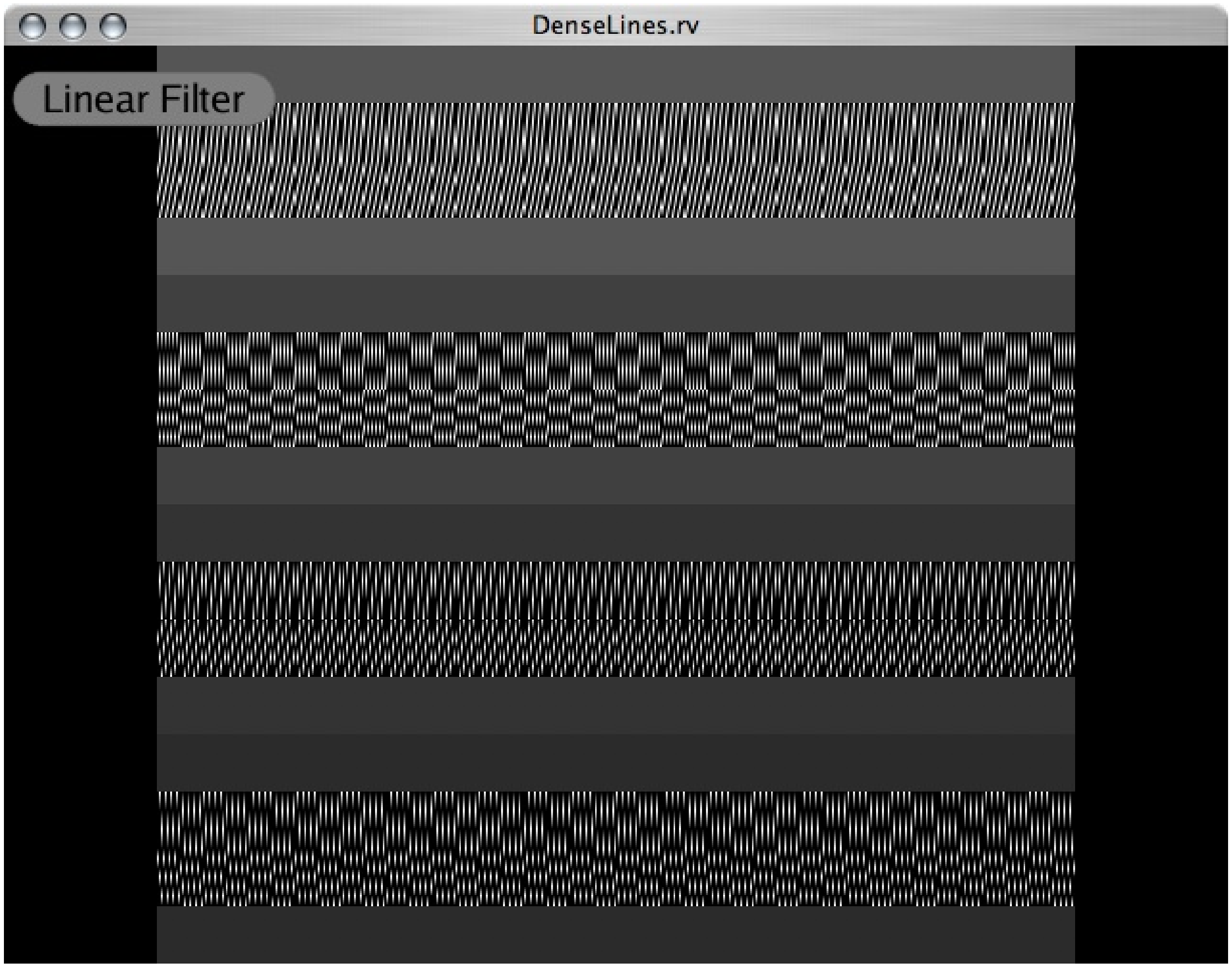
Why does RV default to the linear filter? Most of the time, images and movies come from file formats that store pixel values in linear (scene-referred) space so this default is not an issue. It also looks better.

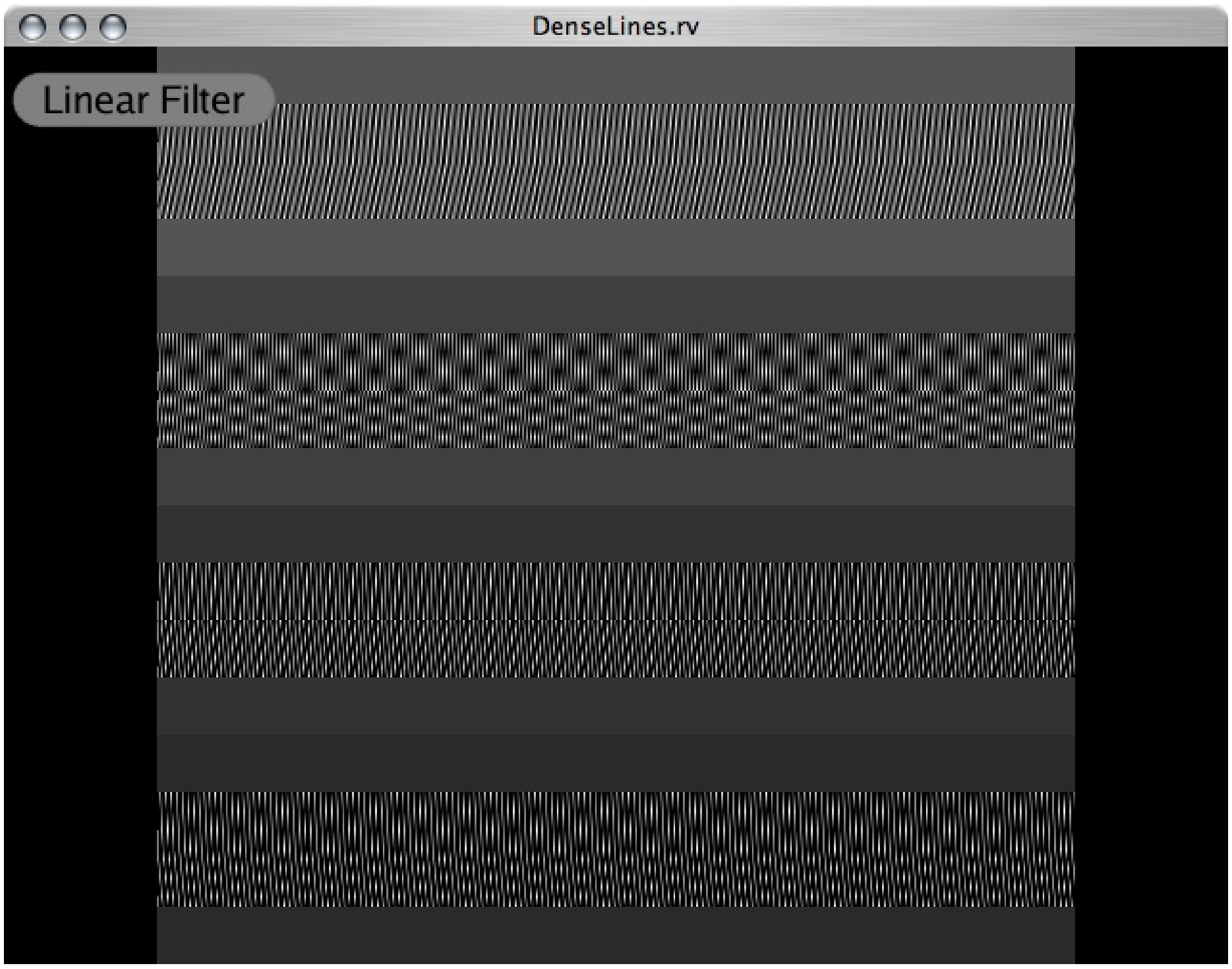
The important thing is to be aware of the issue.

Floating Point Images

If RV is displaying floating point data directly, linear filtering may not occur even though it is enabled. This is a limitation of some graphics cards that will probably be remedied (via driver update or new hardware) in the near future. In this case You can make the filter work by disallowing floating point values via Image → Color Resolution → Allow Floating Point. Many graphics can do filtering on 16 bit floating point images but cannot do filtering on 32 bit floating point images. RV automatically detects the cards capabilities and will turn off filtering for images if necessary.

Figure *4.4* shows an example of a floating point image with linear filtering enabled versus equivalent 8-bit images.




DenseLines.rv
Linear Filter

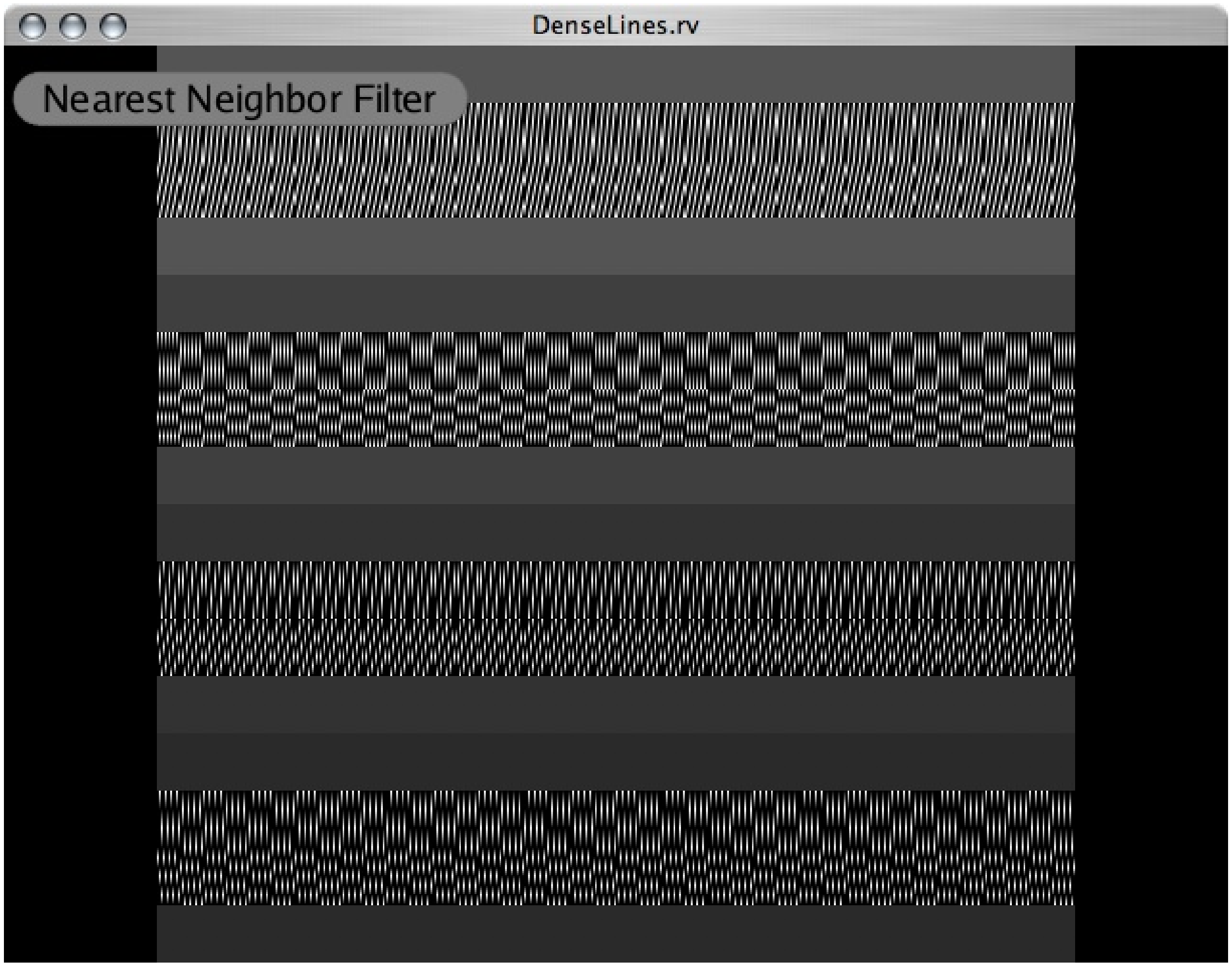


Table 4.4: Floating point linear, 8 bit linear, and 8 bit nearest neighbor filtering.

Graphics hardware does not always correctly apply linear filtering to floating point images. Filtering can dramatically change the appearance of certain types of images. In this case, the image is composed of dense lines and is zoomed out (scaled down).

10.4.6 4.4.6 Big Images

RV can display any size image as long as it can fit into your computer's memory. When an image is larger than the graphics card can handle, RV will tile the image display. This makes it possible to send all the pixels of the image to the card for display. The downside is that all of the pixels are sent to the display even though you probably can't see them all. However, if you zoom in (for example hit ``1" for 1:1 scale) when a large image is loaded, RV will only draw pixels that are visible.

One of the constraints that determines how big an image can be before RV will tile it is the amount of available memory in your graphics card and limitations of the graphics card driver. On most systems, up to 2k by 2k images can be displayed without tiling (as long as the image has 8-bit integer channels). In some cases (newer cards) the limit is 4k by 4k. However, there are other factors that may reduce the limit.

If your window system uses the graphics card (like macOS or Linux with the X.org X server) or there are other graphics-intensive applications running, the amount of available memory may be dependent on these processes as well. Alternately, because RV wants to use as much graphics memory as it can, RV may cause graphics resource depletion that affects other running applications that should have higher priority. Because of this, RV has the capability to limit its graphics memory usage. You can specify this in RV's Preferences by editing the *Maximum VRAM Usage Per Tile* or on the command line with the -vram option.

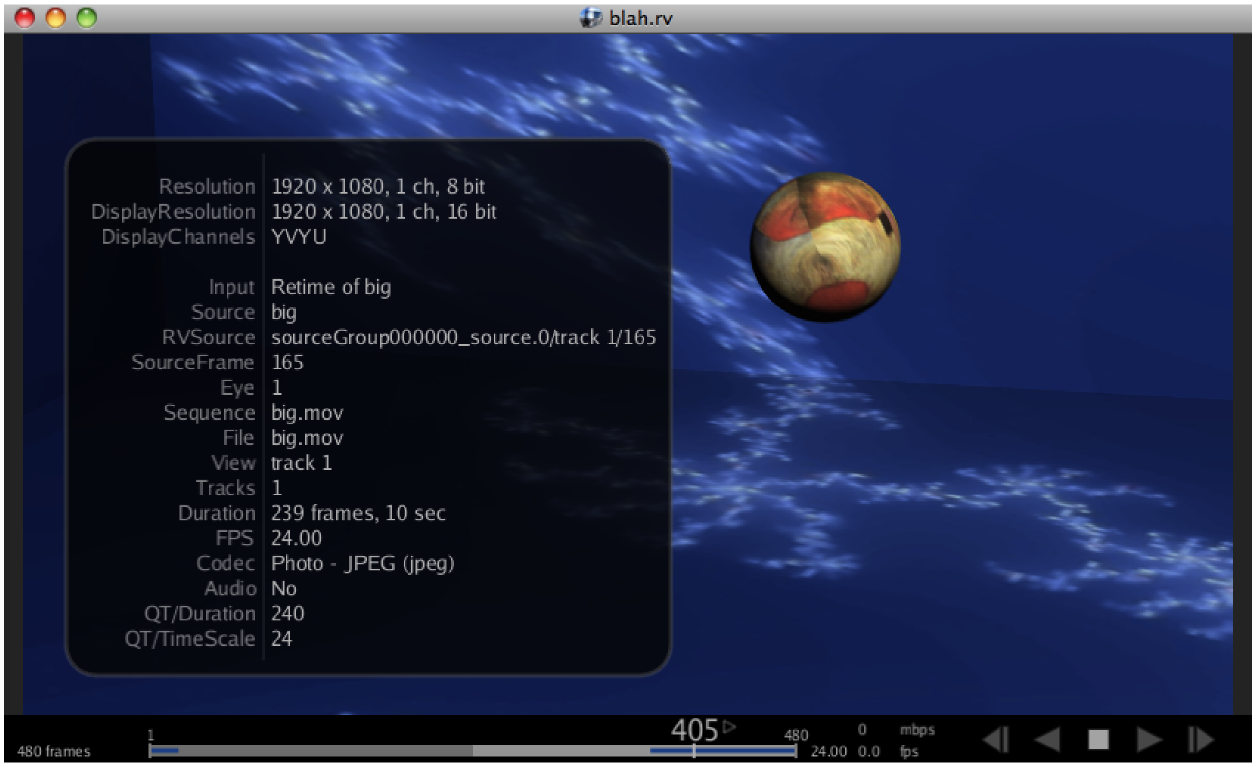
Over time, these problems will go away as drivers and operating systems become smarter about graphics resource allocation.

If you reduce the VRAM usage, RV will tile images of smaller size. For sequences, this may affect playback speed since tiling is slightly less efficient than not tiling. Tiling also affects interactive speed on single images; if tiling is not on, RV can keep all of the image pixels on the graphics card. If tiling is on, RV has to send the pixels every time it redraws the image.

You can determine if RV is tiling the image by looking the image info widget under Tools → Image Info. If tiling is on there will be an entry called ``DisplayTiling" showing the number of tiles in X and Y.

10.4.7 4.4.7 Image Information

The image information widget, can be shown or hidden via the Tools → Image Info menu item or using the hot key: ``i". You can move the widget by clicking and dragging. The widget shows the geometry and data type of the image as well as associated meta-data (attributes in the file). Figure *4.9* shows an example of the information widget.



Resolution	1920 x 1080, 1 ch, 8 bit
DisplayResolution	1920 x 1080, 1 ch, 16 bit
DisplayChannels	YVYU
Input	Retime of big
Source	big
RVSource	sourceGroup000000_source.0/track 1/165
SourceFrame	165
Eye	1
Sequence	big.mov
File	big.mov
View	track 1
Tracks	1
Duration	239 frames, 10 sec
FPS	24.00
Codec	Photo - JPEG (jpeg)
Audio	No
QT/Duration	240
QT/TimeScale	24

Figure 4.9: Image Information Widget.

Channel map information—the current mapping of file channels to display channels—is displayed by the info widget as well as the names of channels available in the image file; this display is especially useful when viewing an image with non-RGBA channels.

If the image is part of a sequence or movie the widget will show any relevant data about both the current image as well as the sequence it is a part of. For movie files, the codecs used to compress the movie are also displayed. If the movie file has associated audio data, information about that will also appear.

To remove the image information widget from the view either move the mouse to the top left corner of the widget and click on the close button that appears or toggle the display with the Tools menu item or hot key ('i').

10.5 4.5 Playing Image Sequences, Movie Files, and Audio Files

RV can play multiple images, image sequences and movie files as well as associated audio files. Play controls are available via the menus, keyboard, and mouse. Timing information and navigation is provided by the timeline widget which can be toggled via the Tools → Timeline menu item or by hitting the TAB key.

10.5.1 4.5.1 Timeline

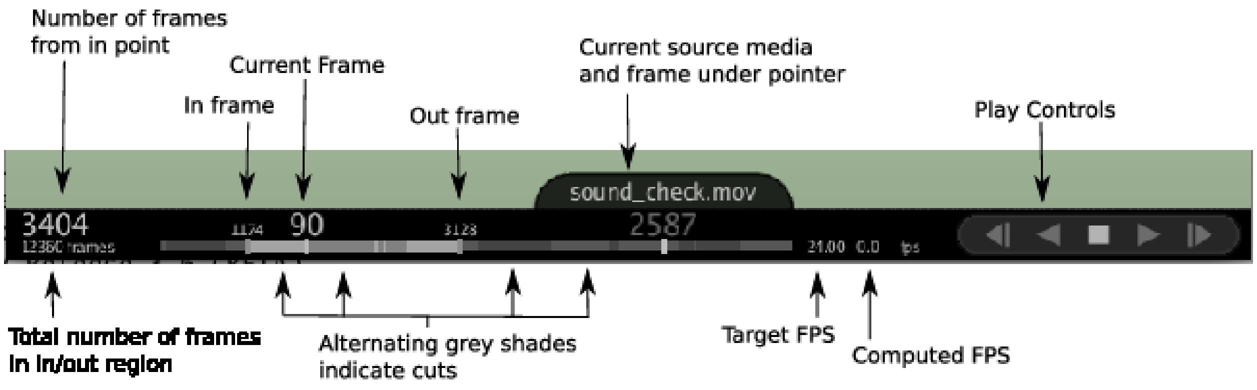

Number of frames from in point
Current Frame
In frame
Out frame
Current source media and frame under pointer
Play Controls
sound_check.mov
3404
12360 frames
1174
90
3128
2587
24.00
0.0
fps
Total number of frames in in/out region
Alternating grey shades indicate cuts
Target FPS
Computed FPS

Figure 4.10: Timeline With Labelled Parts

This timeline shows in and out points, frame count between in and out points, total frames, target fps and current fps. In addition, if there are frame marks, these will appear on the timeline as seen in Figure *4.5.5* .

The current frame appears as a number positioned relative to the start frame of the session. If in and out points are set, the relative frame number will appear at the left side of the timeline — the total number of frames between the in and out points is displayed below the relative frame number.

Tip: To change the start frame of a Version (with or without Slate): Toggle **Movie Has Slate** (or **Frames have Slate**) to on, and then set the First Frame and Last Frame Fields on the Version. If your Movie (or Frames) are 201 to 299 and 201 is the slate, you need to set First Frame to 202.

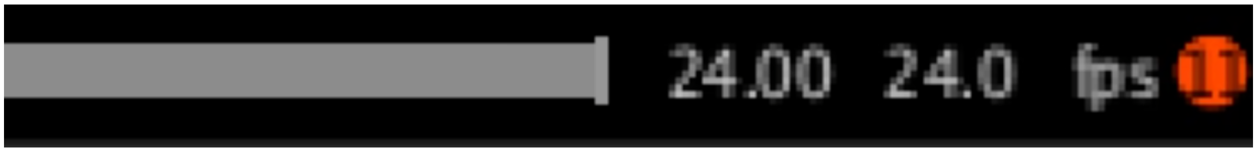
By clicking anywhere on the timeline, the frame will change. Clicking and dragging will scrub frames, as will rolling the mouse-wheel. Also note that you can shift-click drag to set an in/out range.

The in/out range can also be manipulated with the mouse. You can grab and drag either end, or grab in the middle to drag the whole range.

There are two FPS indicators on the timeline. The first indicates the target FPS, the second the actual measured playback FPS.

[	Set in point
]	Set out point
\	Clear in/out points
right-arrow	Step one frame to the right.
left-arrow	Step on frame to the left.
alt-right-arrow	Move current frame to next mark (or source boundary, if there are no marks)
alt-left-arrow	Move current frame to previous mark (or source boundary, if there are no marks)
ctrl(mac: cmd)-left-arrow	Set in/out to next pair of marks (or source boundaries, if there are no marks)
ctrl(mac: cmd)-right-arrow	Set in/out to previous pair of marks (or source boundaries, if there are no marks)
down-arrow, spacebar	Toggle playback
up-arrow	Reverse play direction

Table 4.5: Useful Timeline Hotkeys



Note: A red dot with a number indicates how many frames RV has lost since the last screen refresh.

10.5.2 4.5.2 Timeline Configuration

The timeline can be configured from its popup menu. Use the right mouse button anywhere on the timeline to show the menu. If you show the popup menu by pointing directly at any part of the timeline, the popup menu will show that frame number, the source media there, and the operations will all be relative to that frame. For example, without changing frames you can set the in and out point or set a mark via the menu.

By default the timeline will show the ``source" frame number, the native number of the media. Alternately you can show the global frame number, global time code, or even the ``Footage" common in traditional animation (16 frames per foot).

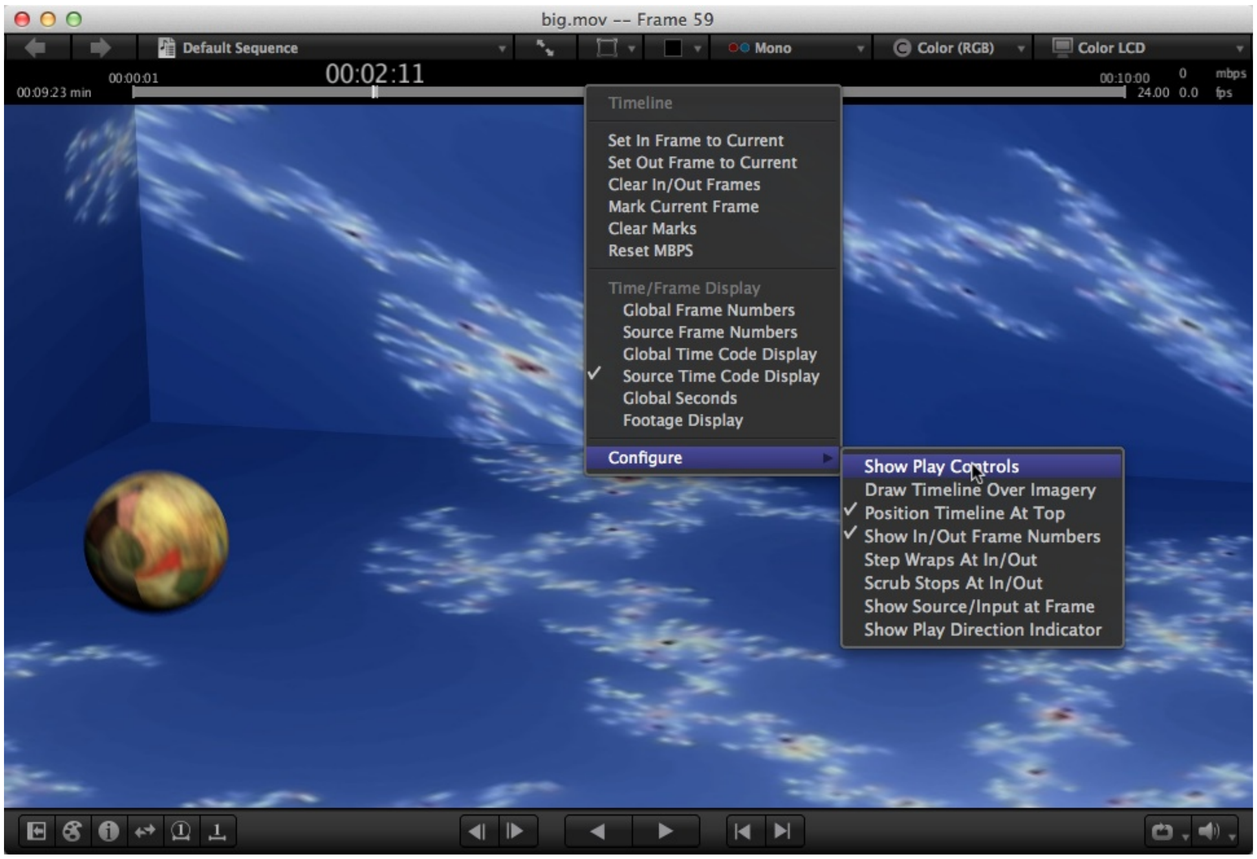


Figure 4.12: Timeline Configuration Popup Menu

The Configuration menu has a number of options:

Show Play Controls	Hide or Show the playback control buttons on the right side of the timeline
Draw Timeline Over Imagery	This was the default behavior in previous versions of RV. The timeline is now drawn in the margin by default
Position Timeline At Top	Draw the timeline at the top of the view. The default is to draw it at the bottom of the view.
Show In/Out Frame Numbers	When selected, the in and out points will be labeled using the current method for display the frame (global, source, or time code).
Step Wraps At In/Out	This controls how the arrow keys behave at the in and out point. When selected, the frame will wrap from in to out or vice versa.
Show Source/Input at Frame	When selected, the main media file name for the frame under the pointer (not the current frame) will be shown just above or below the timeline.
Show Play Direction Indicator	When selected, a small triangle next to the current frame indicates the direction playback will occur, when started.

10.5.3 4.5.3 Realtime versus Play All Frames

Control → Play All Frames determines whether RV should skip frames or not if it is unable to render fast enough during playback. Realtime mode (when play all frames is not selected) uses a realtime clock to determine which frame should be played. When in realtime mode, audio never skips, but the video can. When play all frames is active, RV will never skip frames, but will adjust the audio if the target fps cannot be reached.

When the timeline is visible, skipped frames will be indicated by a small red circle towards the right hand side of the display. The number in the circle is the number of frames skipped.

10.5.4 4.5.4 In and Out Points

There are two frame ranges associated with each view in an RV Session: the start and end frames and the in and out frames (also known as in and out points). The in and out points are always within the range of the start and end frames. RV sets the start and end frames automatically based on the contents of the view. The in and out points are set to the start and end frames by default. However, you can set these points using the ``[" and ``]" keys, or by right-clicking on the timeline. You can also set an in/out range by shift-dragging with the left-button in the Timeline or the Timeline Magnifier. The in/out range displayed in the timeline can also be changed with the mouse, either by dragging the whole range (click down in the middle of the range), or by dragging one of the endpoints (click down on the endpoint).To reset the in and out points, use the ``" key.

If frames have been marked, RV can automatically set the in and out points for you based on them (use the ctrl-right/left-arrow keys, or command-right/left-arrow on Mac).

10.5.5 4.5.5 Marks

A mark in RV is nothing more than a frame number which can be stored in an RV file for later use. To toggle a frame as being marked, use Mark → Mark Frame (or use the ``m" hotkey). The timeline will show marks if any are present.

While not very exciting in and of themselves, marks can be used to build more complex actions in RV. For example, RV has functions to set the in and out points based on marks. By marking shot boundaries in a movie file, you can quickly loop individual shots without selecting the in and out points for each shot. By selecting Mark → Next Range From Marks and Mark → Previous Range From Marks or using the associated hot keys ``control+right arrow" or ``control+left arrow" the in and out points will shift from one mark region to the next.

Marking and associated hot keys for navigating marked regions quickly becomes indispensable for many users. These features make it very easy to navigate around a movie or sequence and loop over part of the timeline. Producers and coordinators who often work with movie files of complete sequences (for bidding or for client reviews) find it useful to mark up movie at the shot boundaries to make it easy to step through and review each shot.

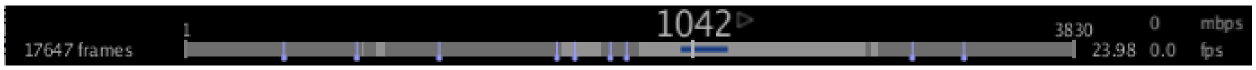


Figure 4.13: Timline with Marks

10.5.6 4.5.6 Timeline Magnifier

The Timeline Magnifier tool (available from the **Tools** menu, default hotkey F3) brings up a special timeline that is ``zoomed" to the region bounded by the In/Out Points. In addition to showing only the in/out region, the timeline magnifier differs from the standard timeline in that it shows frame ticks and numbers on every frame, if possible. If there is not enough room for frames/ticks on every frame, the magnifier will fall back to frames/ticks every 5 or 10 frames. The frame numbers of the in/out points are displayed at either end of the magnified timeline.

Audio Waveform Display

The timeline magnifier can display the audio waveform of any loaded audio. Note that this is the normalized sum of all audio channels loaded for the given frame range. To preserve interactive speed, the audio data is not rendered into the timeline until that section of the frame range is played. You can turn on **Scrubbing** , in the **Audio** menu, to force the entire frame range to be loaded immediately. Also, if **Scrubbing** is on, audio will play during scrubbing, and during single frame stepping. See Section *4.6* for further details on Audio in RV.

In/Out Range Manipulation

Note that on each end of the timeline magnifier, there are two triangular ``arrow" buttons. These are the in/out nudge buttons, and clicking on them will move the in or out point by one frame in the indicated direction. The in/out range displayed in the timeline can also be changed with the mouse, either by dragging the whole range (click down in the middle of the range), or by dragging one of the endpoints (click down on the endpoint). All these manipulations can be performed during playback. You can also set an in/out range by shift-dragging with the left-button in timeline magnifier.

Configuration

All the hotkeys mentioned in Table *4.5* are also relevant to the timeline magnifier. The timeline magnifier configuration menu is also a subset of the regular timeline menu (see Figure *4.12*), with additional items for setting the height of the audio waveform display.

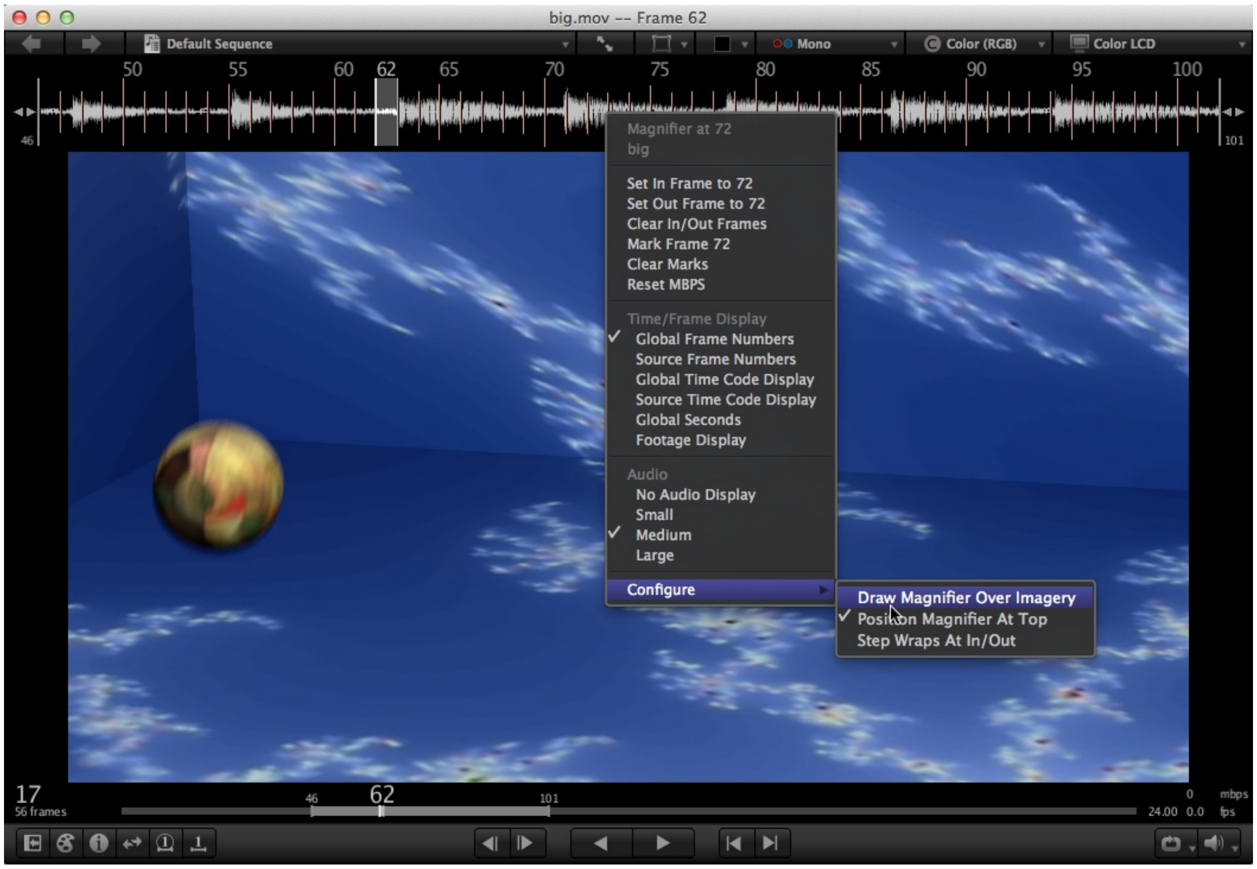


Figure 4.14:

Timeline Magnifier Configuration Popup Menu

10.6 4.6 Audio

When playing back audio with an image sequence or movie file, RV can be in one of two modes: video locked to audio or audio locked to video.

When a movie with audio plays back at its native speed, the video is locked to the audio stream. This ensures that the audio and video are in sync.

If you change the frame rate of the video, the opposite will occur: the audio will be locked to the video. When this happens, RV will synthesize new audio based on the existing audio in an attempt to either stretch or compress the playback in time. When pushed to the limits, the audio synthesis can create artifacts (e.g. when slowing down or speeding up by a factor of 2 or more).

RV can handle audio files with any sample rate and can re-sample on the fly to match the output sample rate required by the available audio hardware. The recommended formats are AIFF or WAV. Use of mp3 and audio-only AAC files is not supported.

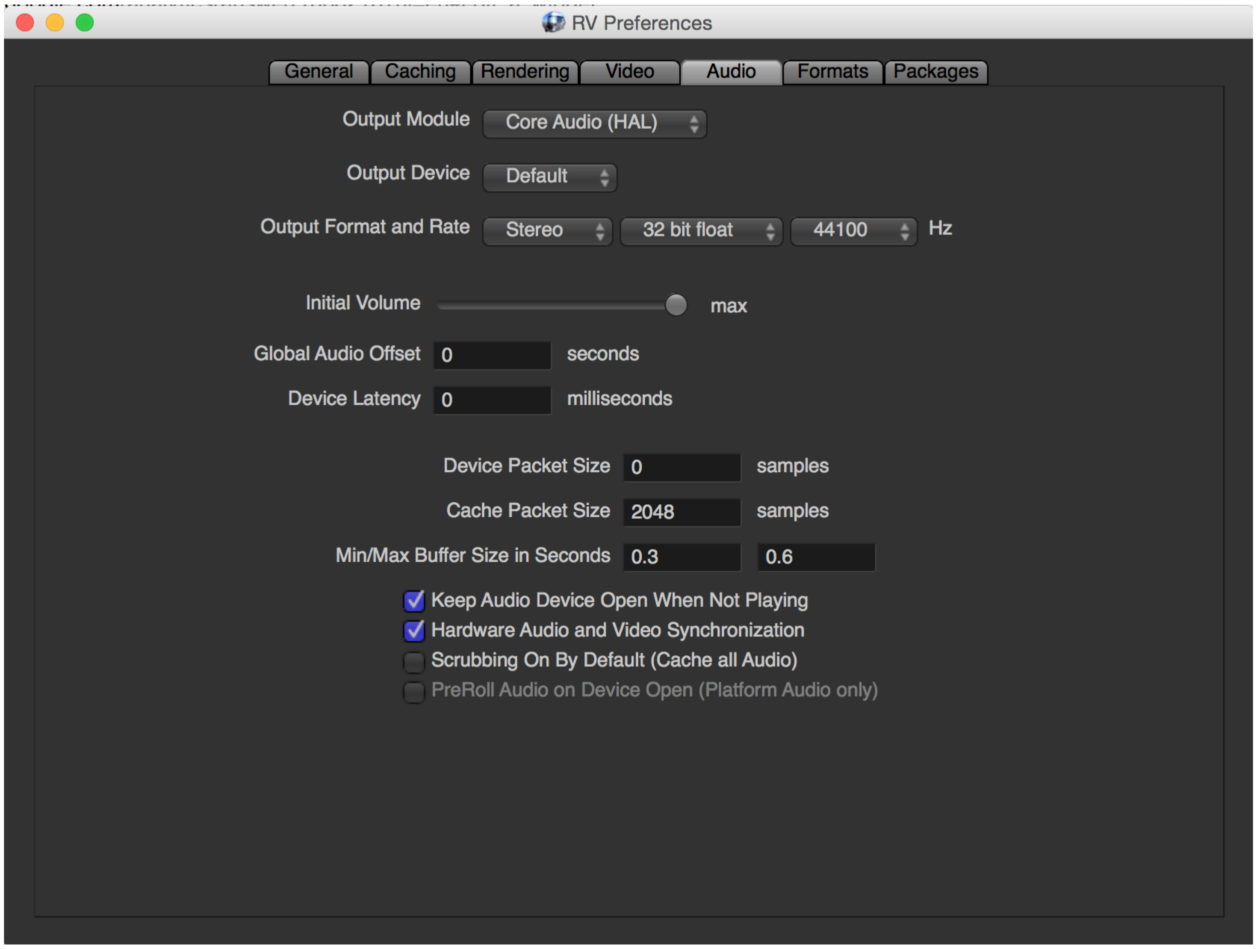
Audio settings can be changed using the items on the Audio Menu. Volume, time Offset, and Balance can be edited per source or globally for the session. The RV Preferences Audio tab lets you choose the default audio device and set the initial volume (as well as some other technical options that are rarely changed).

For visualizing the audio waveform see Section *4.5.1* .

10.6.1 4.6.1 Audio Preferences

RV provides audio preferences in the Preferences dialog. The most important audio preference is the choice of the output device from those listed. In practice this will rarely change. Preferences also let you set the initial volume for RV. The option to hold audio open is for use on Linux installations where audio system support is poor (see the next section on Linux Audio.) The other preferences are there for fine tuning performance in extreme cases of marginal audio hardware or support - they will almost never change.

RV offers a cross-platform output module choice called “Platform Audio”. This is based on Qt audio. “Platform Audio” does support the use USB based audio peripherals for playback (e.g. Behringer UCA 202) on all platforms. These usb audio devices would typically appear as “USB Audio CODEC” (“front:CARD=CODEC,DEV=0” on Linux) in the “Output Device” pull down menu when “Platform Audio” is selected.



Output Module: Core Audio (HAL)
Output Device: Default
Output Format and Rate: Stereo, 32 bit float, 44100 Hz
Initial Volume: max
Global Audio Offset: 0 seconds
Device Latency: 0 milliseconds
Device Packet Size: 0 samples
Cache Packet Size: 2048 samples
Min/Max Buffer Size in Seconds: 0.3, 0.6
✓ Keep Audio Device Open When Not Playing
✓ Hardware Audio and Video Synchronization
☐ Scrubbing On By Default (Cache all Audio)
☐ PreRoll Audio on Device Open (Platform Audio only)

Figure 4.15: Audio Preferences (macOS)

On the macOS and Windows there is only a single entry in this menu. On Linux, however, there may be many. (See Appendix *E* for details about Linux Audio).

The Output Device, Output Channel Layout, Output Format, and Output Rate determine the sound quality and speakers (e.g. mono, stereo, 5.1 etc) to use. Typical output rates are 44100 or 48000 Hz and 32 bit float or 16 bit integer output format. This produced Audio CD or DAT quality audio.

Global Audio Offset is the means by which audio data can be time shifted backwards or forwards in time. The effect of this preference is observable in the audio waveform display. For example, setting the value to 0.5 seconds will shift the audio data by 0.5 seconds.

Device Latency allows you to correct for audio/video sync differences measured during playback. It is measured in milliseconds, and defaults to zero. The audio waveform rendered in RV is not affected by the value of this preference since it does not offset the audio data that is cached.

The Device Packet Size and Cache Packet Size can be changed, but not all output modules support arbitrary values. The default values are recommend. The Min/Max Buffer Size determines how much audio RV will cache ahead of the display frame. Ideally these numbers are low.

Keep Audio Device Open When Not Playing should usually be set to ON. There are very few circumstances in which it's a good idea to turn this off. When the value is OFF, RV may skip frames and audio when playback starts and can become unstable. On some linux distributions turning this OFF will result in no audio at all after the first play.

Hardware Audio and Video Synchronization determines which clock RV will use to sync video and audio. When on, RV will use the audio hardware clock if one is available otherwise it will use a CPU timer in software. In most cases this should be left ON. RV can usually detect when the audio clock is unstable or inaccurate and switch to the CPU timer automatically. However if playback with audio appears jerky (even when caching is on) it might be worth turning it off.

Scrubbing On By Default (Cache all Audio) determines if audio scrubbing is on when RV starts.

PreRoll Audio on Device Open generally improves the consistency of RV's playback across different Linux machines and audio devices. It influences the overall AV sync lag, so expect to see different in AV sync readings when the feature is enabled versus turned off. In either case, the AV sync lag can be corrected via the Device Latency preference. Note that this feature is Linux only and available only for the Platform Audio module. It defaults to turned off.

10.6.2 4.6.2 Audio on Open RV for Linux

Linux presents special challenges for multimedia applications and audio is perhaps the worst case. RV audio works well on Linux in many cases, but may be limited in others. RV supports special configuration options so that users can get the best audio functionality possible within the limitations of the vintage and flavor of Linux being used. See the Appendix *E* for complete details.

10.6.3 4.6.3 Multichannel Audio

In RV, the audio output module "Platform Audio" will support multichannel channel audio playback for devices that allow it. This would include six, eight or more channel layouts for surround sound speaker systems like 5.1 or 7.1. The list of available channel layouts for a chosen output device will be listed in the first pulldown menu for the "Output Format and Rate" setting.

The list of all possible channel layouts that RV supports is described in Appendix *J* .

10.6.4 4.6.4 Correcting for AV Sync Delay

The "Device Latency" preference is intended to be used to compensate for any positive or negative AV sync measured during playback. To correct for an AV sync lag, first measure the delay with an AV sync meter. Then input the number from the meter into the Device Latency preference.

Please note: The AV sync measurement can be influenced by the following audio preferences or playback settings: Hardware Audio Video Sync, PreRoll Audio on Device Open, and video/audio cache settings.

To generate a sync flash sequence for use in measuring the AV sync at a particular frame rate, the following RV command line can be used. This example generates a 500 frame sequence with an audio bleep/video flash at intervals of 1sec at 24 FPS:

```
rv syncflash,start=1,end=500,interval=1,fps=24.movieproc
```

10.7 4.7 Caching

RV has a three state cache: it's either off, caching the current in/out range, or being used as a look-ahead (also known as a ring) buffer.

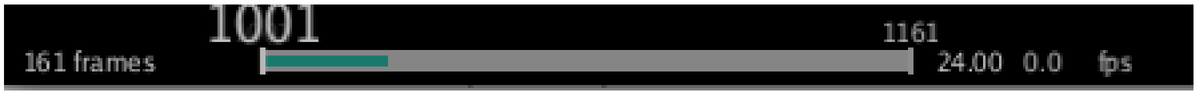


Figure 4.16: Timeline Showing Cache Progress

The region cache reads frames starting at the in point and attempts to fill the cache up to the out point. If there is not enough room in the cache, RV will stop caching. The region cache can be toggled on or off from the Tools menu or by using the shift-C hot key.

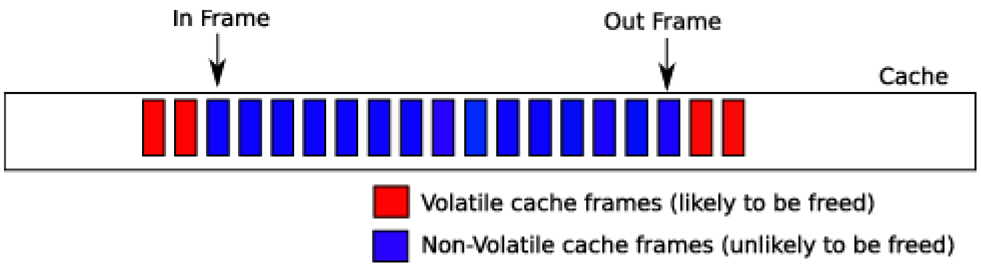

In Frame
Out Frame
Cache
Volatile cache frames (likely to be freed)
Non-Volatile cache frames (unlikely to be freed)

Figure 4.17: Region Cache Operation With Lots of Memory

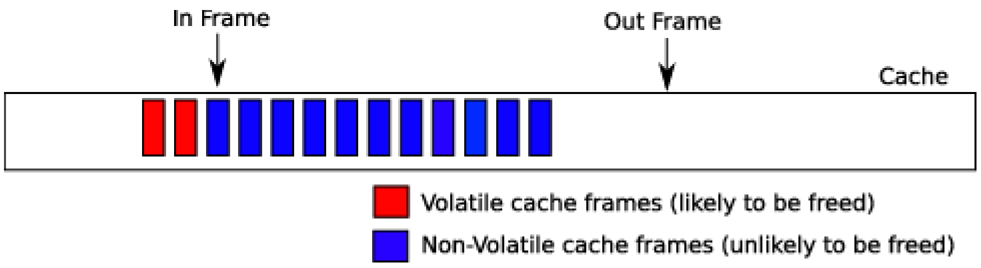

In Frame
Out Frame
Cache
Volatile cache frames (likely to be freed)
Non-Volatile cache frames (unlikely to be freed)

Figure 4.18: Region Cache Operation During Caching With Low Memory

Look-ahead caching can be activated from the Tools menu or by using the meta-l hot key. The look-ahead cache attempts to smooth out playback by pre-caching frames right before they are played. If RV can read the files from disk at close to the frame rate, this is the best caching mode. If playback catches up to the look-ahead cache, playback will be paused until the cache is filled or for a length of time specified in the Caching preferences. At that point playback will resume.

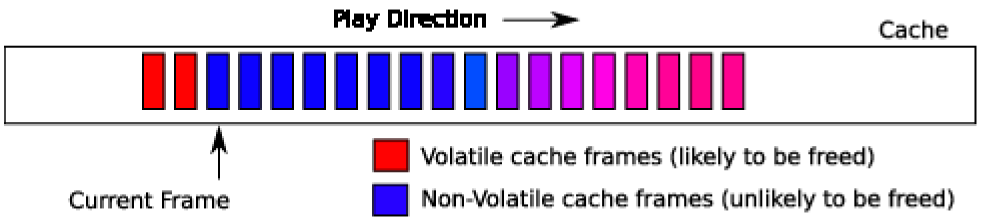

Play Direction
Cache
Current Frame
Volatile cache frames (likely to be freed)
Non-Volatile cache frames (unlikely to be freed)

Figure 4.19: Look-Ahead Cache Operation

RV caches frames asynchronously (in the background). If you change frames while RV is caching it will attempt to load the requested frame as soon as possible.

If the timeline widget is visible, cached regions will appear as a dark green stripe along the length of the widget. The stripe darkens to blue as the cache fills. The progress of the caching can be monitored using the timeline. On machines with multiple processors (or cores) the caching is done in one or more completely separate threads.

Note that there is usually no advantage to setting the lookahead cache size to something large (if playback does not overtake the caching, a small lookahead cache is sufficient, and if it does, you probably want to use region caching anyway).

10.8 4.8 Color, LUTs, and CDLs

RV provides users with fine grained color management and can support various color management scenarios. See *7.1* for detailed technical information about RV's color pipeline. Without adding any nodes the default graph in RV supports three LUTs and two CDLs per file, an overall display LUT, and has a number of useful color transforms built-in. You can load LUTs and CDLs using the File → Import menu (Display, Look, File, and Pre-Cache items), or you can drag and drop the files onto the RV window. Smart drop targets will allow you determine how the LUT or CDL will be applied. Note that there is no CDL slot for the display by default. See chapter *8* for more information about using LUTs and *9* for using CDLs in RV.

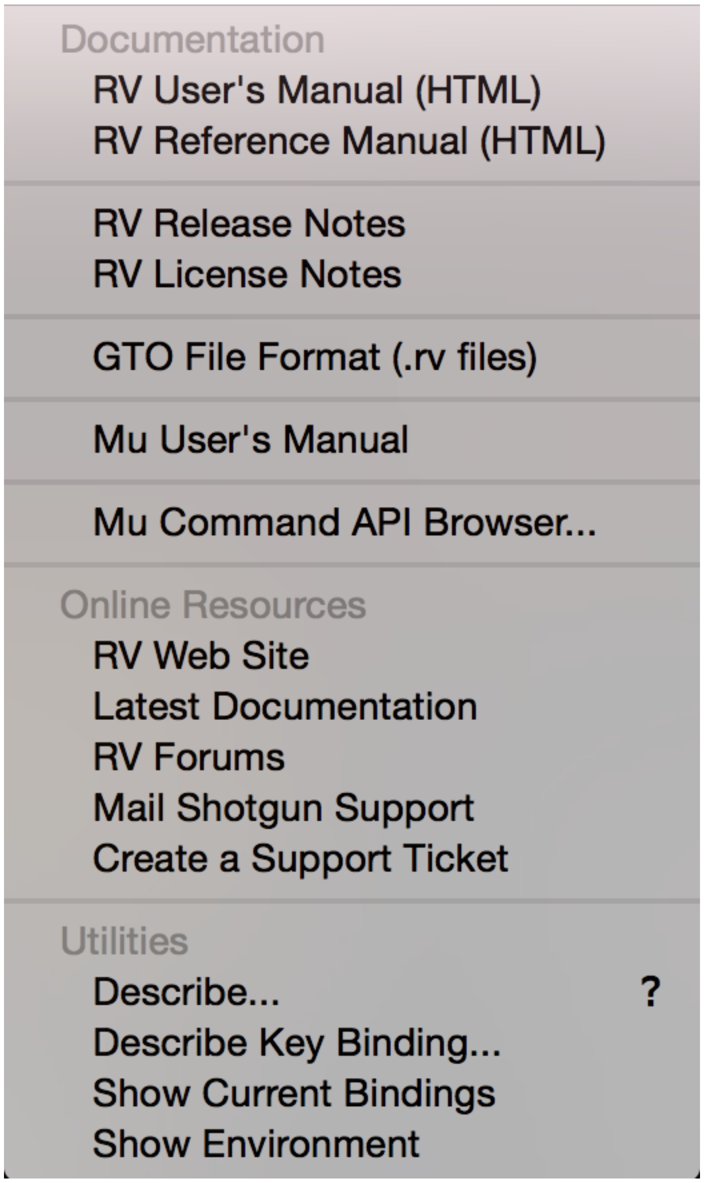
RV's color transforms are separated into two menus. The Color menu contains transforms that will be applied to an individual source (whichever source is current in the timeline) and the View menu contains transforms that will be applied to all of the sources. This provides the opportunity to bring diverse sources (say Cineon Log files, QuickTime sRGB movies, and linear-light Exr's) all into a common working color space (typically linear) and then to apply a common output transform to get the pixels to the display. RV's built in hardware conversions can handle Cineon Log, sRGB, Viper Log and other useful transforms.

10.9 4.9 Stereo

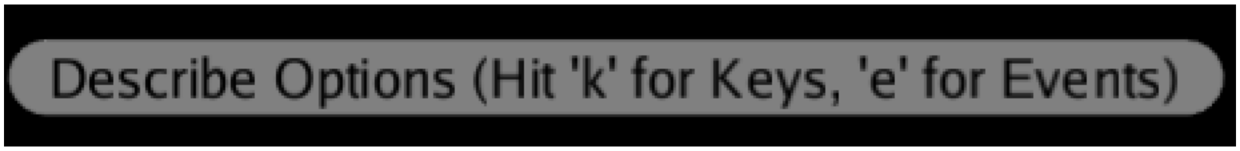
RV supports playback of stereoscopic source material. RV has two methods for handling stereo source material: first any source may have multiple layers, and RV will treat the first two video layers of a source as left and right eyes for the purpose of stereo display. Left and right layers do not need to be the same resolution or format because RV always conforms media to fit the display window; Second, RV supports stereo QuickTime movies (taking the first two video tracks as left and right eyes) and multi-view EXR files. RVIO can author stereo QuickTime movies and multi-view EXR files as well, so a complete stereo publishing and review pipeline can be built with these tools. See the section on *12* for more information about how stereo is handled.

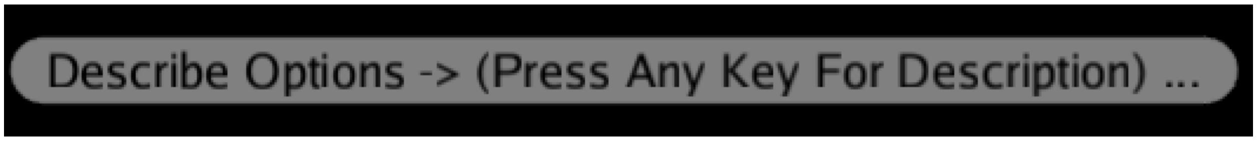
10.10 4.10 Key and Mouse Bindings

RV has many built-in shortcuts. You can learn about RV's hotkeys via RV's Help menu.

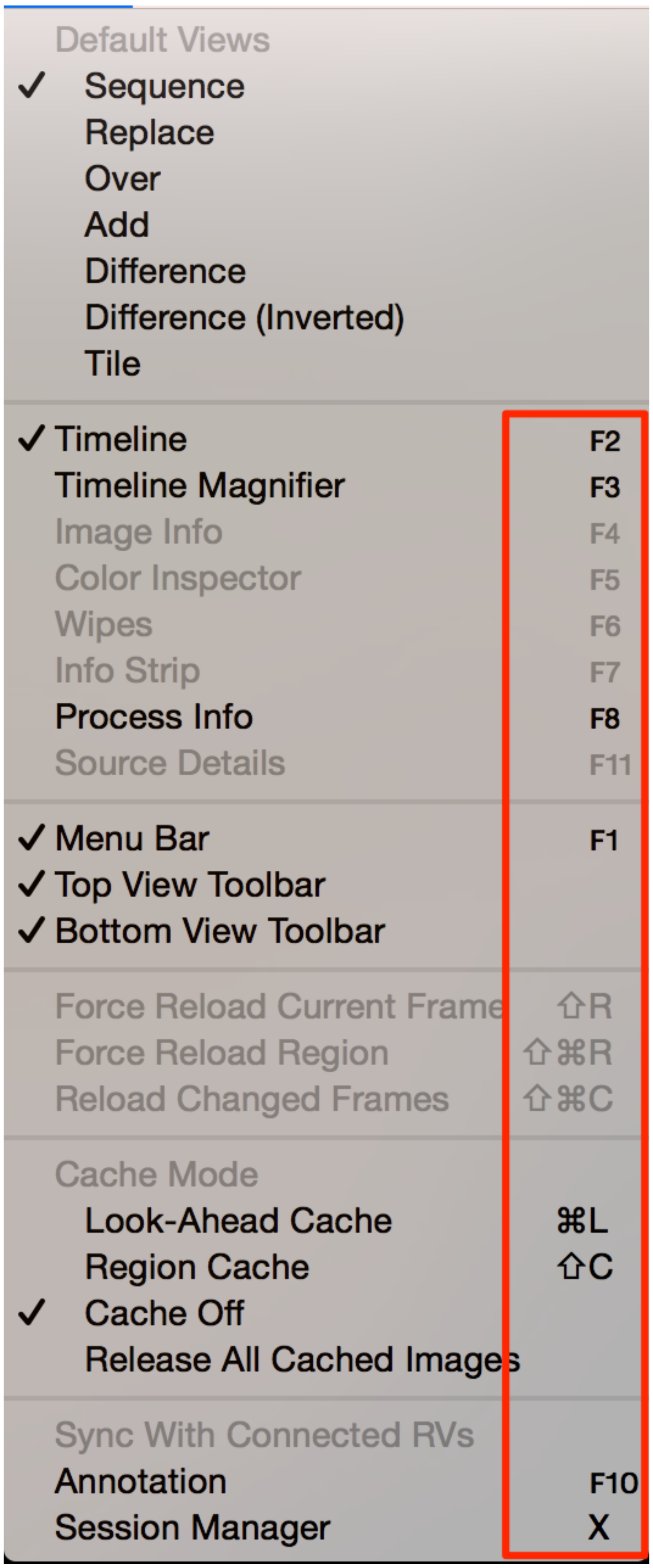


From the Utilities section of the Help menu, select "Describe…" or "Describe Key Binding…" to see an explanation within RV of what certain hotkeys do.





Menu items with hotkeys also display the hotkey on the right side of the menu item.


Default Views
✓ Sequence
Replace
Over
Add
Difference
Difference (Inverted)
Tile
✓ Timeline F2
Timeline Magnifier F3
Image Info F4
Color Inspector F5
Wipes F6
Info Strip F7
Process Info F8
Source Details F11
✓ Menu Bar F1
✓ Top View Toolbar
✓ Bottom View Toolbar
Force Reload Current Frame ⇧R
Force Reload Region ⇧⌘R
Reload Changed Frames ⇧⌘C
Cache Mode
Look-Ahead Cache ⌘L
Region Cache ⇧C
✓ Cache Off
Release All Cached Images
Sync With Connected RVs
Annotation F10
Session Manager X

If you'd like to see a list of all of RV's current key bindings, select "Show Current Bindings" from the Help menu. Below is also a list of RV's hotkeys (note that capital and lowercase letters are different hotkeys):

Hotkey	Action
F1 (Fn + F1 on Mac)	Toggle Menu Bar Visibility
F2 (Fn + F2 on Mac)	Toggle Heads-Up Timeline
F3 (Fn + F3 on Mac)	Toggle Timeline Magnifier
F4 (Fn + F4 on Mac)	Toggle Heads-Up Image Info
F5 (Fn + F5 on Mac)	Toggle Heads-Up Color Inspector
F6 (Fn + F6 on Mac)	Toggle Wipes
F7 (Fn + F7 on Mac)	Toggle Heads-Up Info Strip
F8 (Fn + F8 on Mac)	Toggle Heads-Up External Process Progress
~	Toggle Timeline
`	Toggle Fullscreen Mode
!	Set image scale to 1:1 on presentation device
*	Apply Random Luminance LUT
(	Cycle Image Stack Backwards
)	Cycle Image Stack Forwards
[	Set In Point
]	Set Out Point
|	Set In/Out Range From Surrounding Marks
\	Reset In/Out Points
,	Set Frame Increment to -1 (reverse)
.	Set Frame Increment to 1 (forward)
<	Go to Matching Frame of Previous Source
>	Go to Matching Frame of Next Source
?	Show Help Options
Space	Toggle Play
1	Scale 1:1
2	Scale 2:1
3	Scale 3:1
4	Scale 4:1
5	Scale 5:1
6	Scale 6:1
7	Scale 7:1
8	Scale 8:1
A	Toggle Real-Time Playback
a	Show Alpha Channel
B	Edit Display Brightness
b	Show Blue Channel
C	Toggle Region Caching
c	Normal Color Channel Display
D	Toggle Display LUT
e	Edit Current Source Relative Exposure
F	Enter FPS Value From Keyboard
f	Frame Image in View
G	Set Frame Number Using Keyboard
g	Show Green Channel
h	Edit Current Source Hue
i	Toggle Heads-Up Image Info
k	Edit Current Source Contrast
L	Toggle Cineon Log to Linear Conversion

continues on next page

Table 1 – continued from previous page

Hotkey	Action
l	Show Image Luminance
M	Cycle Matte Opacity
m	Toggle Mark At Frame
n	Toggle Nearest Neighbor/Linear Filter
P	Toggle Ping/Pong Playback
p	Toggle Premult Display
q	Close Session
R	Force Reload of Current Source
r	Show Red Channel
S	Edit Current Source Saturation
T	Toggle Current Luminance LUT
t	Toggle Heads-Up Timeline
v	Enter Display Gamma
W	Fit Window to Pixels
w	Resize Window to Fit
X	Flop Image
Y	Flip Image
y	Edit Current Source Gamma
Alt + f (Option + f on Mac)	Set image scale fit image width on presentation device
Alt + l (Option + l on Mac)	Rotate Image 90deg Counter-Clockwise
Alt + n (Option + n on Mac)	Turn On Nudge Keys
Alt + r (Option + r on Mac)	Rotate Image 90deg Clockwise
Alt + s (Option + s on Mac)	Turn On Stereo Keys
Alt + left arrow (Option + left arrow on Mac)	Go to Previous Marked Frame
Alt + right arrow (Option + right arrow on Mac)	Go to Next Marked Frame
Keypad-enter (Fn + enter/return on Mac)	Set Frame Number Using Keyboard
Home (Fn + left arrow on Mac)	Go to Beginning of In/Out Range
End (Fn + right arrow on Mac)	Go to End of In/Out Region
Page-up (Fn + up arrow on Mac)	Set In/Out to Next Marked Range
Page-down (Fn + down arrow on Mac)	Set In/Out to Previous Marked Range
Ctrl + e (Cmd + e on Mac)	Export Quicktime Movie
Ctrl + f (Cmd + f on Mac)	Frame Image Width
Ctrl + i (Cmd + i on Mac)	Add Source
Ctrl + l (Cmd + l on Mac)	Toggle Look-Ahead Caching
Ctrl + m (Cmd + m on Mac)	Cycle Mattes
Ctrl + o (Cmd + o on Mac)	Open File
Ctrl + p (Cmd + p on Mac)	Toggle Presentation Mode
Ctrl + q (Cmd + q on Mac)	Close Session
Ctrl + s (Cmd + s on Mac)	Save Session
Ctrl + v (Cmd + v on Mac)	Edit Global Audio Volume
Ctrl + w (Cmd + w on Mac)	Close Session
Ctrl + left arrow (Cmd + left arrow on Mac)	Set In/Out to Previous Marked Range
Ctrl + right arrow (Cmd + right arrow on Mac)	Set In/Out to Next Marked Range
Ctrl + up arrow (Cmd + up arrow on Mac)	Expand In/Out to Neighboring Marked Ranges
Ctrl + down arrow (Cmd + down arrow on Mac)	Contract In/Out from Neighboring Marked Ranges
Left arrow	Move Back One Frame
Right arrow	Step Forward 1 Frame
Up arrow	Toggle Forward/Backward Playback
Down arrow	Toggle Play
Tab	Toggle Heads-Up Timeline

continues on next page

Table 1 – continued from previous page

Hotkey	Action
Shift + home (Shift + Fn + left arrow on Mac)	Reset All Color
Shift + left arrow	Go to Previous View
Shift + right arrow	Go to Next View

10.10.1 Stereo Hotkeys

RV has a supplementary “stereo hotkeys” mode in which an additional set of stero-related hotkeys are active. You enter or leave the mode with Alt-s (Option-s on Mac). While this mode is active, the following additional hotkeys are available:

Hotkey	Action
a	Anaglyph Mode
d	Checked Mode
k	Scanline Mode
s	Side-by-Side Mode
p	Side-by-Side Stereo Mode
m	Mirrored Side-by-Side Stereo Mode
x	Stereo Mode Off (DEPRECATED – use alt-s (or option-s on the mac))
h	Hardware Stereo Mode
,	Left Eye Only Stereo Mode
.	Right Eye Only Stereo Mode
<	Left Eye Only Stereo Mode
>	Right Eye Only Stereo Mode
S	Swap Eyes
o	Edit Global Relative Stereo Offset
z	Horizontal Squeezed Stereo Mode
v	Vertical Squeezed Stereo Mode
r	Edit Global Right-Eye Stereo Offset
O	Turn OFF Stereo Display Mode (in Controller Window)
/	Reset Stereo Offsets
c	Edit Source/Clip Stereo Offset
R	Edit Source/Clip Right-Eye-Only Stereo Offset

10.10.2 Mouse Bindings

Key and mouse bindings as well as menu bar menus are loaded at run time. You can override and change virtually any key or mouse binding from a file called ~/.rvrc.mu if you need to. The bindings (and whole interface) that comes with RV are located at $RV_HOME/plugins/Mu/rvui.mu. Functions in this file can be called from ~/.rvrc.mu or overridden.

To override bindings, copy the file $RV_HOME/scripts/rv/rvrc.mu to ~/.rvrc.mu.

Alt	Ctrl	Shift	1	2	3	Wheel	Function
			↓				Toggle Play
				↓			Toggle Play Direction
							Scrub Frames
							Scrub Frames
	•						Scrub Frames 10x
•	•						Scrub Frames 100x
	•						Zoom
							Translate
•							Translate
•	•						Translate
•							Translate (Maya style)
•							Zoom (Maya style)
		•	↓				Inspect Pixel
		•					Average Pixels
					↓		Pop-up Menu

• held, drag left and right, ↓ push without drag, drag any direction.

Mouse button 1 is normally the left mouse button and button 3 is normally the right button on two button mice. Button 2 is either the middle mouse button or activated by pushing the scroll wheel on mice that have them.

If you can't annotate with the tablet and stylus: If you use various inputs to control RV, such as Wacom tablets, then sometimes there is an incompatibility with the events those inputs generate and Qt. Try turning ON *Treat Stylus Events as Mouse Events* from RV Preferences General tab.

10.11 4.11 Preferences File

RV stores configuration information in a preferences file in the user home directory. Each platform has a different location and possibly a different format for the file.

OS	File Location	File Format
macOS	~/Library/Preferences/com.tweaksoftware.RV	Property List
Linux	~/.config/TweakSoftware/RV.conf	Config File
Windows	%APPDATA%/TweakSoftware/RV.ini	INI File

Table 4.8: Preference File Locations

10.12 4.12 Audio Settings

The Output Module in the Audio tab under RV Preferences lists audio interfaces that RV can choose between to handle audio, and is specific to operating systems. Each operating system has its own platform specific option, but fairly recently we created a Qt based cross platform setting named Platform Audio. We encourage you to use this option for Output Module.

The Output Format and Rate configuration is an important setting, and you should choose a reasonable format and rate for the audio contained in the sources you most commonly review. Most of the time this is 32-bit float and either 44.1 or 48 kHz.

Using "Keep Audio Device Open When Not Playing" will help reduce slow down and pops when looping playback or when starting and stopping frequently. However, in some Linux distributions this can conflict with other applications using the audio system. Using "Hardware Audio and Video Synchronization" may help keep your media synced during playback, but could introduce small delays for the two systems to line up when starting playback.

10.12.1 Audio Scrubbing

Under the Audio menu is an option to enable Scrubbing. This can be turned on by default from the Audio tab of RV's preferences. For animators that are doing frame by frame stepping through clips, Scrubbing for lip-sync and other types of critical audio event timing can be useful.

10.12.2 Offset Playback Timing of Open RV audio

On the Audio tab of RV's preferences, you can offset the playback timing of all RV audio in Global Audio Offset. This setting exists in case your system has some kind of audio exclusive latency so that you cannot configure both audio and video latencies from the Video tab.

10.12.3 Synchronizing Audio and Video

When using RV to play an image sequence paired with an audio file you may find that RV appears to play your imagery faster (or slower) than the accompanying audio. This happens because the way RV works is based on assuming each source has a native frame rate and that audio files do not have a native frame rate, because they have no frames. If RV is unable to determine the frame rate of a source and no indication was given at load, then RV will use the Default FPS. This is set in the Preferences dialog on the General tab.

The Default FPS setting is used by both RV and RVIO, and it normally is only used for sources that have no discernible native frame rate, such as in an image sequence (without FPS metadata) or audio-only sources.

CHAPTER

ELEVEN

CHAPTER 5 - THE SESSION AND THE SESSION MANAGER

11.1 5.1 Open RV Session

Each viewer window represents an RV session. A session is composed of one or more source movies, frame markers, image transforms, color corrections, and interactive states (like caching and playback speed). The source movies are combined according to the session type. The default RV session type is ``sequence,'' which plays back the source movies one after another.

An RV session can be saved as an .rv file. The .rv file contains the entire state of its image processing tree—all of the variables that determine how it will work—as well as information about frame ranges, in/out points, etc. The .rv file stores references to movie files and images; it does not make copies of them. If you change source material on disk and load an .rv file that references those materials, the changes will be evident in RV.

The .rv file is a GTO file. Tools that operate on GTO files can be used on .rv files. C++ and Python source code is available for creating, reading, and manipulating GTO files, including the ASCII GTO files used by RV. *See RV File Format*.

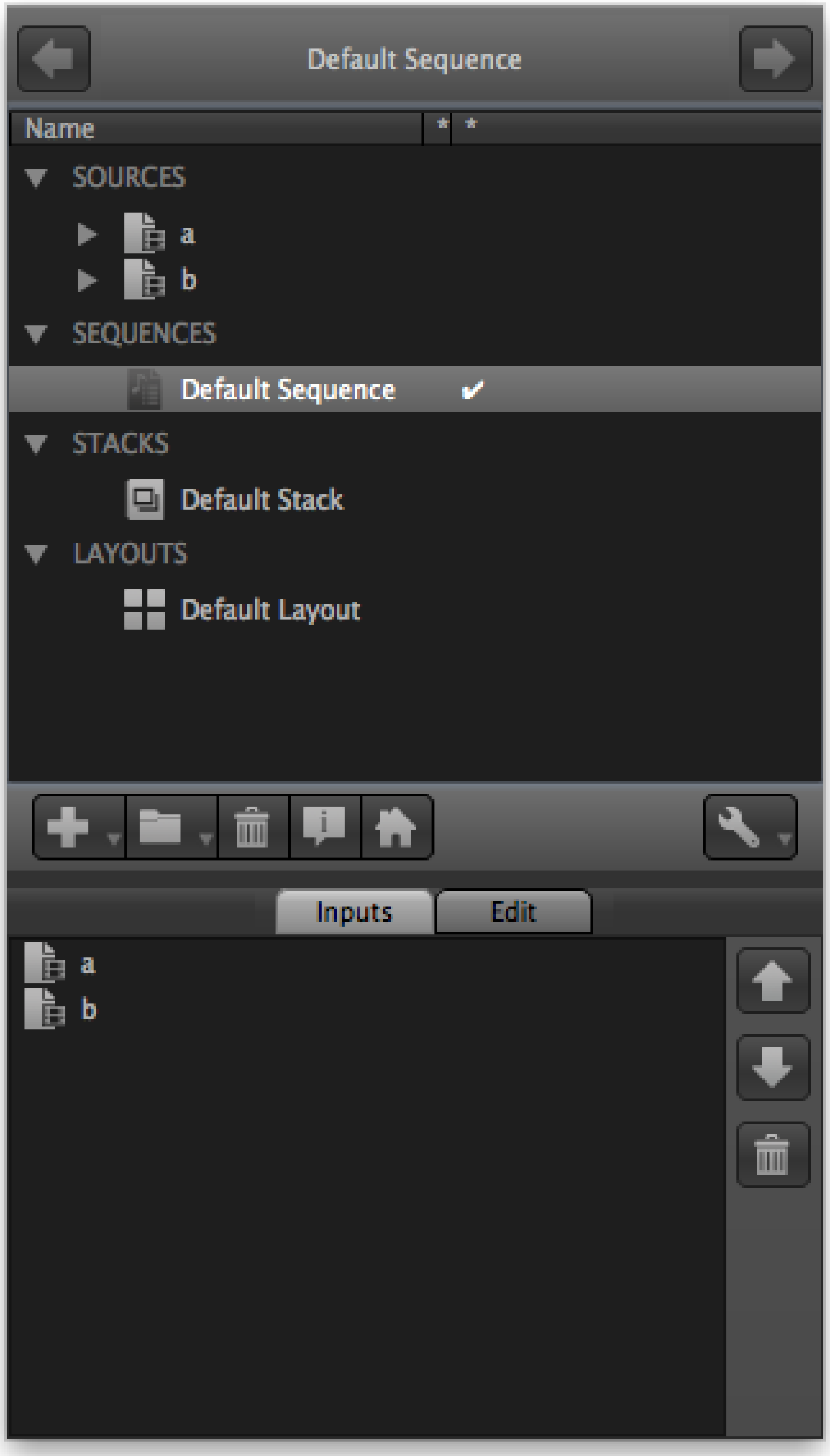
11.2 5.1.1 What's in a Session

A session is represented internally as a Directed Acyclic Graph (DAG) in which images and audio pass from the leaves to the root where they are rendered. Each node in the DAG can have a number of parameters or state variables which control its behavior. RV's user interface is essentially a controller which simply changes these parameters and state variables. An .rv file contains all of the state variables for the nodes in RV's image processing DAG. A description of each of the node types can be found in the Reference Manual.

You can use the gtoinfo program to view the contents of an .rv file from the command line. Other GTO tools like the python module can be used to edit .rv files without using RV itself.

The DAG nodes that are visible in the user interface are called Views. RV provides three default views, and the ability to make views of your own. In addition to any Sources you've loaded, the three views that all sessions have are the Default Sequence, which shows you all your sources in order, the Default Stack, which shows you all your sources stacked on top of one another, and the Default Layout, which has all the sources arranged in a grid (or a column, row, or any other custom layout of your own design). In addition to the default views, you can create any number of Sources, Sequences, Stacks, and Layouts of your own. Whenever a Source is added to the session, it is automatically added to the inputs of each of the default views, not to user-defined views.

11.3 5.2 Session Manager



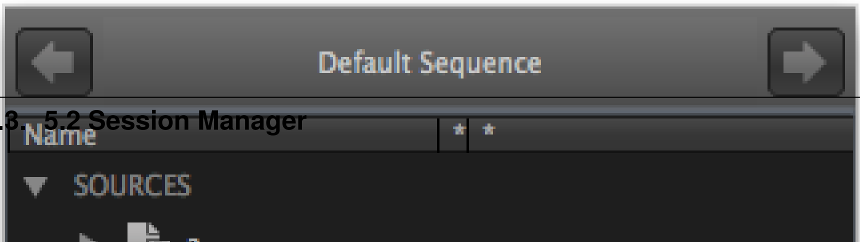
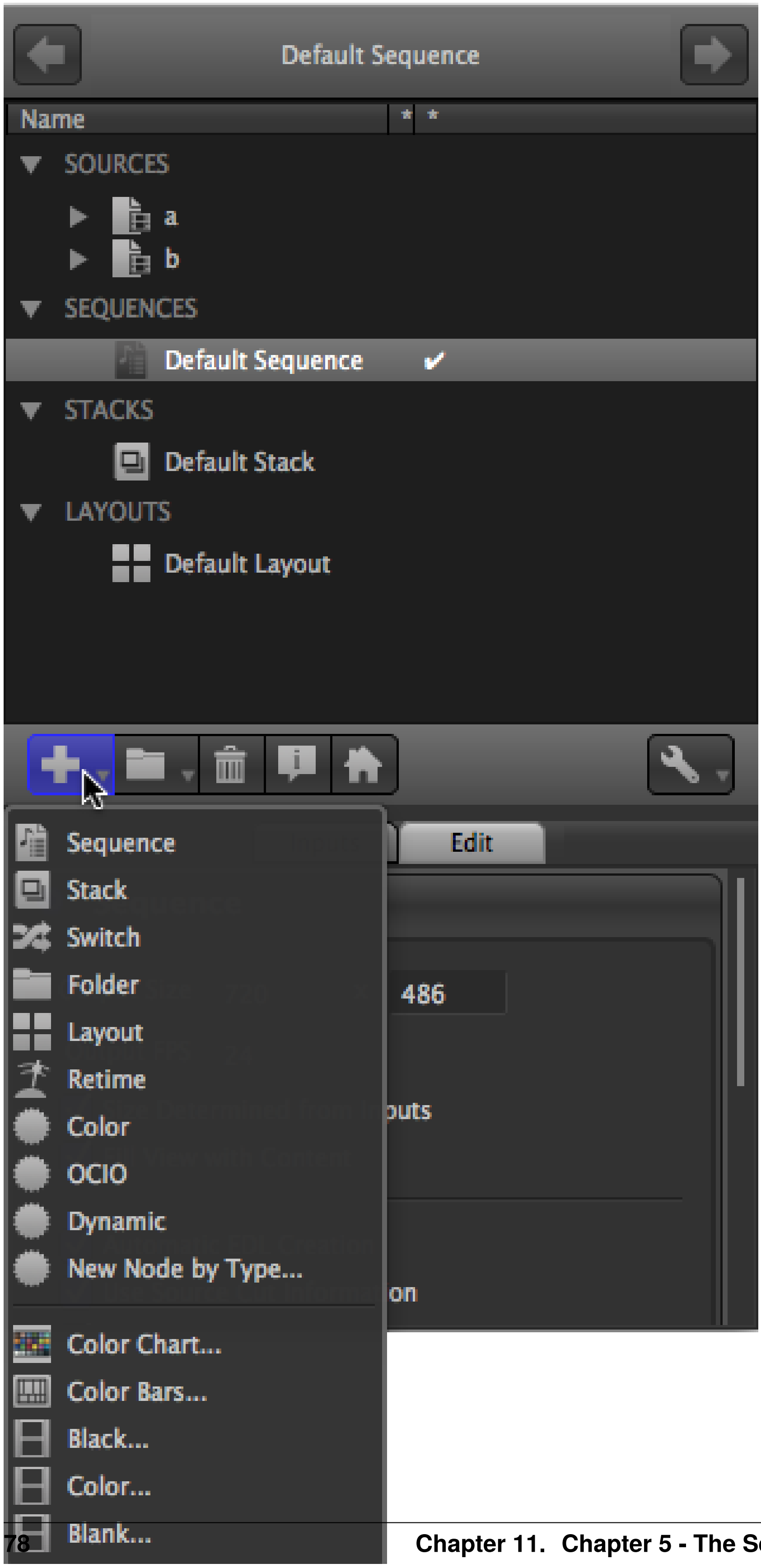


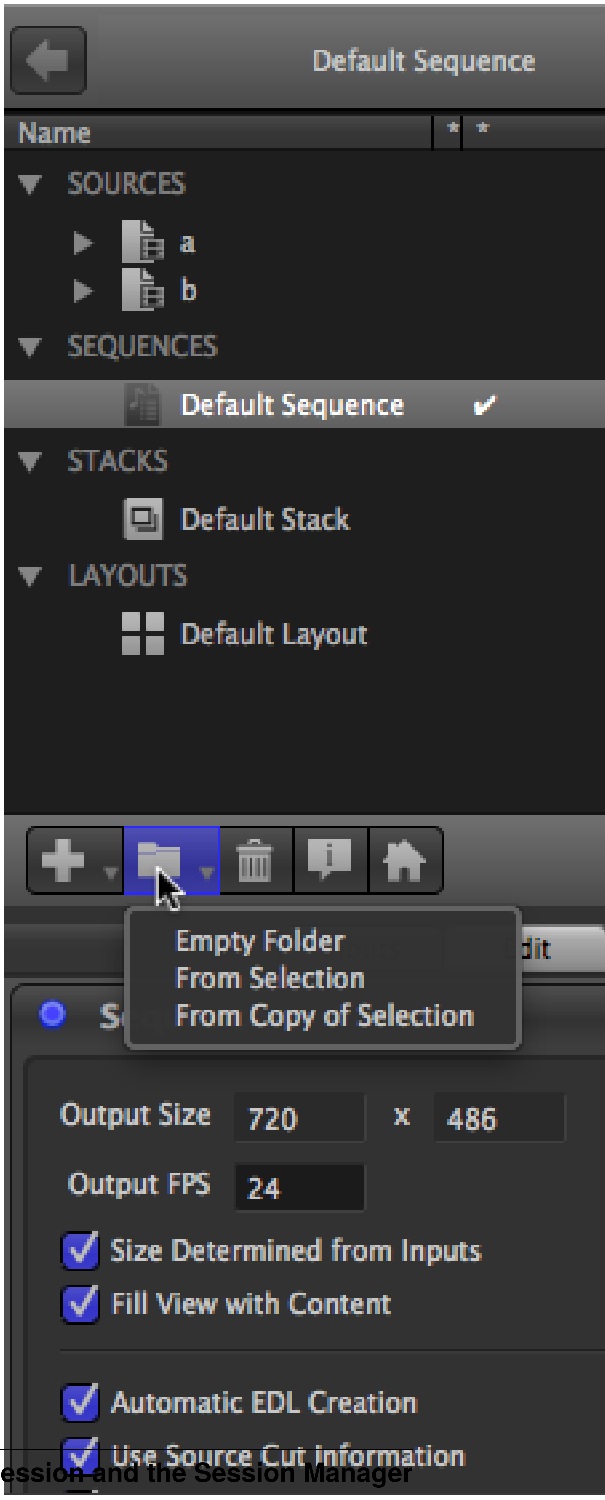
Figure 5.1: Session Manager on the Mac showing Inputs and Sequence Edit Panel. The DefaultSequence is being viewed.

The session manager is used to examine and edit the contents of an RV session. The session manager shows an outline of the session contents from which you can create, modify, and edit new sequences, stacks, layouts, and more. By default, the Session Manage comes up ``docked" at the left side of the RV window, but it can be un-docked (by clicking and dragging on the title) and positioned as a separate window, or docked at another edge of the RV window.

The session manager interface is in two parts: the top panel shows an outline of the session contents, and the bottom shows either the inputs of the currently viewed object or user interface to edit the current view. By double clicking on an icon in the top portion of the session manager you can switch to another view. By default RV will create a default sequence, stack, and layout which includes all of the sources in the session. When a new source is added, these will be automatically updated to include the new source.

11.4 5.3 Creating, Adding to, and Removing from a View





Output Size 720 x 486
Output FPS 24
☑ Size Determined from Inputs
☑ Fill View with Content
☑ Automatic EDL Creation
☑ Use Source Cut information

Figure 5.2:

The Add View and Folders Menus

A new view can be created via the add view menu. The menu is reachable by either the add view (+) button or by right clicking with the mouse in the session outline. Anything selected in the session outline becomes a member (input) of the newly created view. Alternately you can create a view and then add or subtract from it afterwards.

The top items in the view create new views from existing views. The bottom items create new sources which can be used in other views.

Folder views can be created either from the add menu or the folders menu. The folders menu lets you create a folder from existing views or with copies of existing views. When a view is copied in the session manager, the copy is really just a reference to a single object.

You can add to an existing view by first selecting it by double clicking on it, then dragging and dropping items from the session outline into the inputs section of the session manager.

Drag and drop of input items makes it possible to rearrange the ordering of a given view. For example, in a sequence the items are played in the order the appear in the inputs list. By rearranging the items using drag and drop or the up and down arrows in the inputs list you can reorder the sequence.

To remove an item from a view select the item(s) in the inputs list and hit the delete (trash can) button to the right of the inputs list. Similarly, the trashcan button in the upper panel well delete a view from the session. **Please Note** : neither of these remove/delete operations is undoable.

11.5 5.4 Navigating Between Views

For each RV session, there is always a single ``current view”, whose name is displayed at the top of the Session Manager. As in a web browser, RV remembers the history of views you have ``visited” and you can go backwards and forwards in that history.

To change to a different View you can:

- Double-click on any of the views listed in the top panel of the Session Manager
- Double-click on any input view of the current view (listed in the Inputs tab of the bottom panel of the Session Manager)
- Double-click on any visible image in the main RV window
- Click the left (backwards) and right (forwards) buttons at the top of the Session Manager
- For backwards compatibility, the items at the top of the Tools menu navigate to the usual default views

Once you have changed views, you can go backwards and forwards in the view history with the arrow buttons at the top of the Session Manager, or with the navigation hot keys ``Shift-left-arrow” (backwards) and ``Shift-right-arrow” (forwards). Note that you can easily navigate between views with out the Session Manager by double-clicking on the image to ``drill-down” and then using ``Shift-left-arrow” to go back.

11.6 5.5 Source Views

Source Views are the ``leaves” of the graph in that they are views with no inputs (since they get their pixels from some external source, usually files on disk somewhere). The Edit interface for source views is currently used only to adjust editorial information (in the future it may provide access to other per-source information like color corrections, LUTs, etc). In RV, each source has an Cut In/Out information which provide editorial information to views that use that source (like a Sequence view). These In/Out frame numbers can be set from the command line, or changed with the Edit panel of the Source View interface.

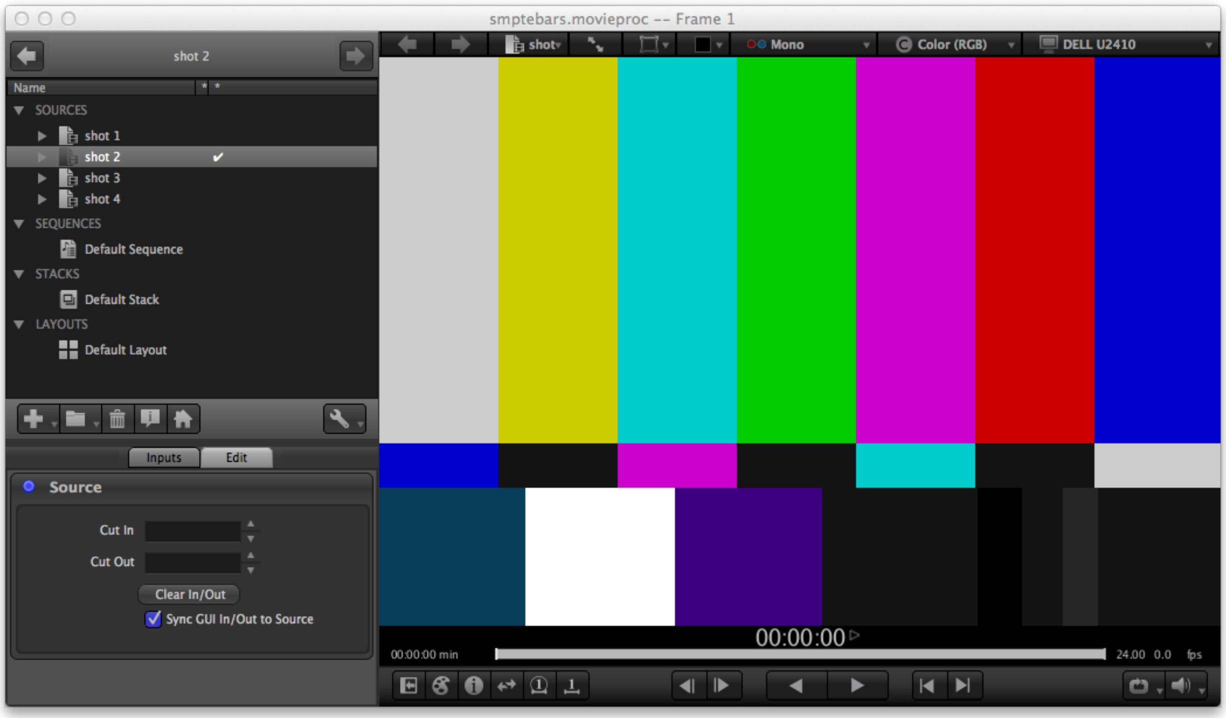


Figure 5.3:

The Source Edit Interface

By default, ``Sync GUI In/Out to Source” is checked, and you can manipulate the Cut In/Out numbers by setting the in/out frames in the timeline in all the usual ways (see Section *4.5.4*). You can also type frame numbers in the given fields, use the up/down nudge keys, or the mouse wheel (after clicking in the relevant field).

11.7 5.5.1 Source Media Containing Multiple Images (Subcomponents)

In the session manager, a source can be opened revealing the media it is composed of. If the media has multiple layers and/or views (e.g. a stereo OpenEXR file) the media can be further opened to reveal these.

When multiple layers or views (subcomponents) are present in media the session manager will present a radio button interface in one of its columns. Each subcomponent in the media has its own selectable toggle button. When a subcomponent is selected, the source will show only that subcomponent; stereo or any other multiple view effect will be turned off.

You can go back to the default by either double clicking on the media or deselecting the selected subcomponent (toggle it off).

In addition to restricting the media to one of its subcomponents, the session manager also allows you to build new views which include more than one subcomponent. For example if you select all the layers in a multiple layer OpenEXR file, you can create a layout view (right popup menu or the '+' tool button) that shows all of them simultaneously. When RV does this, it creates new temporary sources dedicated to the subcomponent views, layers, or channels that were selected. These subcomponent sources are placed in their own folder.

It's also possible to drag and drop subcomponents into existing view inputs.

11.8 5.6 Group Views

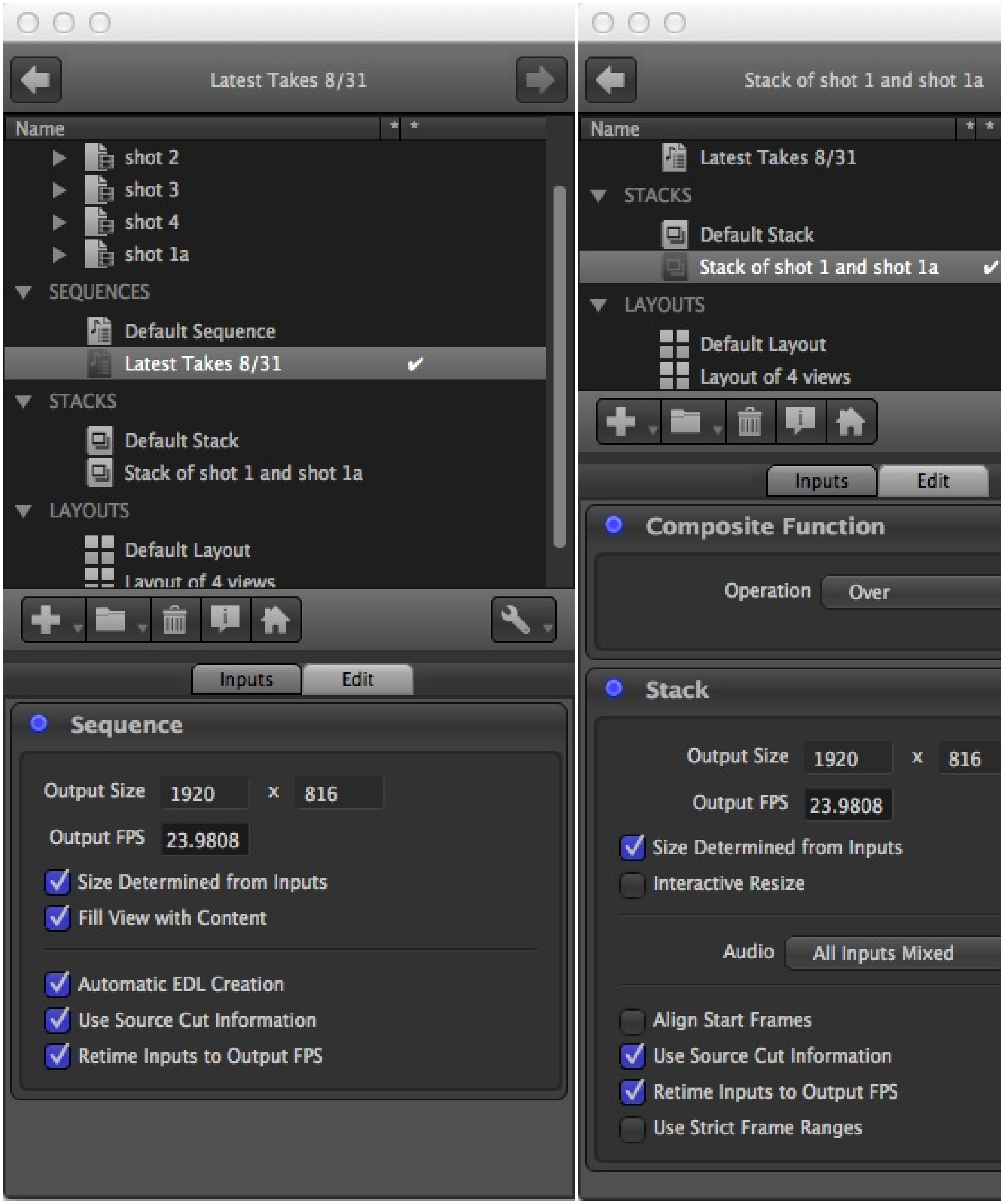
Folders, Sequence Views, Stack Views, Switch Views, and Layout Views are all ``Group Views" in that they take multiple inputs and combine them in some way for viewing. A Sequence plays it's inputs in order, a Stack layers it's aligned inputs on top of each other, and a Layout arranges it's inputs in a grid, row, column or arbitrary user-determined format.

Some interface is shared by all Group Views:

The Group interface gives you control over the resolution of it's output. During interactive use, RV's resolution invariance means that the aspect ratio is the only important part of the size, but during output with RVIO, this size would be the default output resolution. If Size Determined from Inputs' is checked, the group take it's size from the maximum in each dimension of all it's inputs. If the size is not being programmatically determined, you can specify any size output in the provided fields.

Similarly, the output frame rate can be specified in the Output FPS field. This is the frame rate that is used as the default for any RVIO output of this group, and is also passed to any view for which this group is an input. The output FPS is initialized from the default frame rate of the first input added to the group. If Retime Inputs to Output FPS is checked, inputs whose native frame rate differs from the group's output fps will be retimed so that they play correctly at the output fps. That is, a real-time pull up/down will be performed on the video, and the audio will be resampled to play at the output fps while preserving pitch.

As long as Use Source Cut Information is checked in the Group interface, the group will adopt the editorial cut in/out information provided by the sources (see Section *5.5*). This is particularly useful in the case of sequences, but also comes up with stacks and layouts, when, for example, you want to compare a matching region of movies with different overall frame ranges.


Latest Takes 8/31

Name

- shot 2
- shot 3
- shot 4
- shot 1a
- SEQUENCES
  - Default Sequence
  - Latest Takes 8/31 ✓
- STACKS
  - Default Stack
  - Stack of shot 1 and shot 1a
- LAYOUTS
  - Default Layout
  - Layout of 4 views

Inputs | Edit

Sequence

Output Size 1920 x 816

Output FPS 23.9808

- [✓] Size Determined from Inputs
- [✓] Fill View with Content
- [✓] Automatic EDL Creation
- [✓] Use Source Cut Information
- [✓] Retime Inputs to Output FPS

Stack of shot 1 and shot 1a

Name

- Latest Takes 8/31
- STACKS
  - Default Stack
  - Stack of shot 1 and shot 1a ✓
- LAYOUTS
  - Default Layout
  - Layout of 4 views

Inputs | Edit

Composite Function

Operation Over

Stack

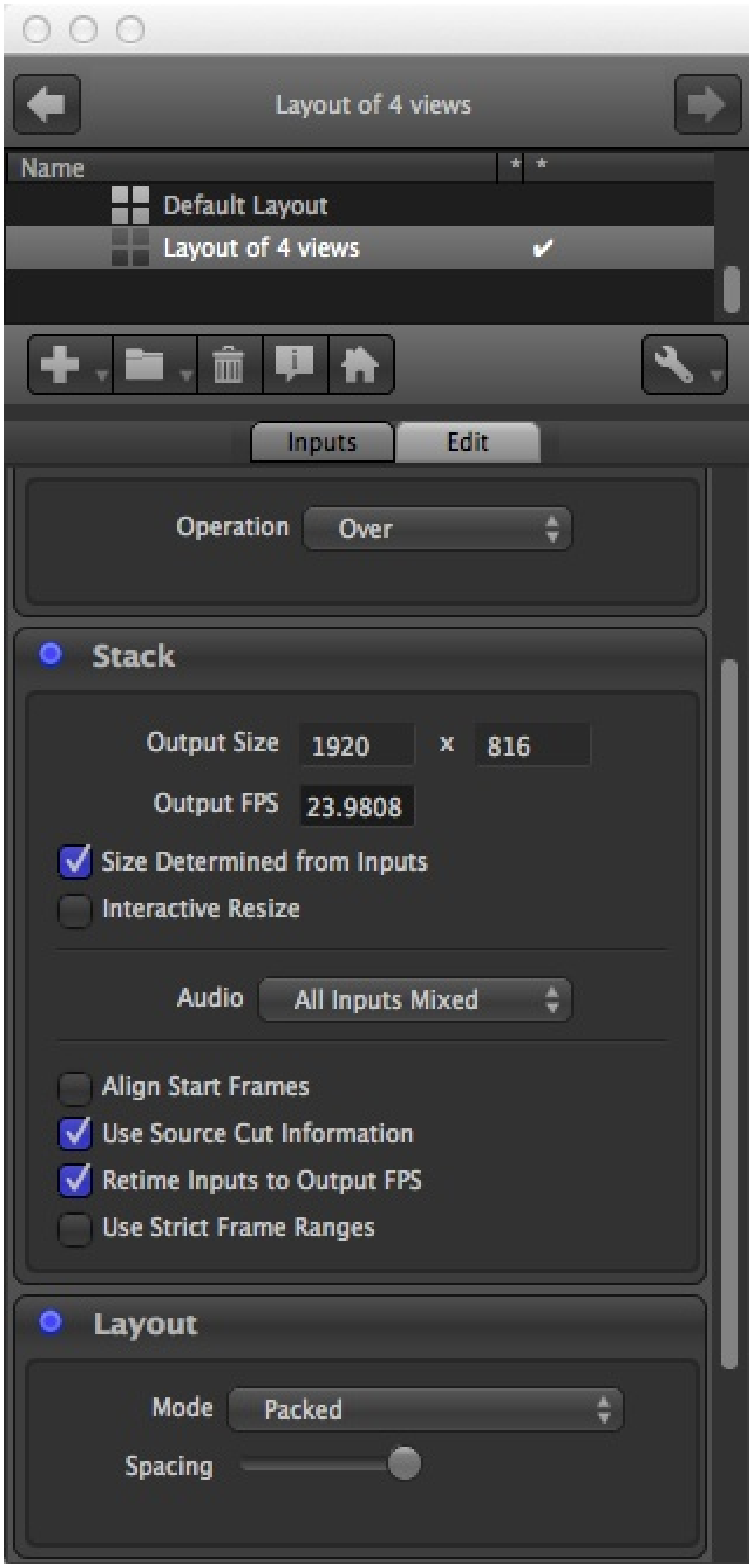
Output Size 1920 x 816

Output FPS 23.9808

- [✓] Size Determined from Inputs
- [ ] Interactive Resize

Audio All Inputs Mixed

- [ ] Align Start Frames
- [✓] Use Source Cut Information
- [✓] Retime Inputs to Output FPS
- [ ] Use Strict Frame Ranges


Layout of 4 views

Name	*	*
Default Layout		
Layout of 4 views		✔

Inputs | Edit

Operation: Over

Stack

Output Size: 1920 x 816

Output FPS: 23.9808

[✓] Size Determined from Inputs

[ ] Interactive Resize

Audio: All Inputs Mixed

[ ] Align Start Frames

[✓] Use Source Cut Information

[✓] Retime Inputs to Output FPS

[ ] Use Strict Frame Ranges

Layout

Mode: Packed

Spacing

Table 5.1:

Group View Interfaces: Sequence, Stack, and Layout

11.8.1 5.6.1 Sequence Views

A Sequence view plays back its inputs in the order specified in the Inputs tab of the the Squence interface. The order can be changed by dragging and dropping in the Inputs, or by selecting and using the arrow keys to the right of the list of Inputs. An input can be removed (dropped from the sequence) by selecting the input and then clicking the trashcan button.

In addition to the order of the clips being determined by the order of the inputs, the actual cut in/out points for each clip can also be specified. At the moment, the easiest way to do this is to specify cut information for each source that you want to appear in the sequence with the Source view interface described in Section *5.5* . As long as Use Source Cut Information is checked in the Sequence interface, the sequence will adopt the editorial cut in/out information provided by the sources.

Since the sequence only shows you one input at a time, if Fill View with Content is checked and the sequence is the current view, the output size of the sequence will be dynamically adjusted so that the ``framed" content always fills the RV window, even if different inputs of the sequence have different aspect ratios.

11.8.2 5.6.2 Stack Views

The Stack View presents it's inputs ``on top" of each other, for comparing or compositing. In this case the order of the inputs determines the stacking order (first input on top). In addition the usual ways of reordering the inputs, you can ``cycle" the stack forwards or backwards with items on the Stack menu, or with hotkeys: ')' and '('.

The compositing operation used to combine the inputs of the stack can be selected in the Edit interface. At the moment, you can choose from Over, Replace, Add, Difference, and Inverted Difference.

Because any or all of the inputs to the Stack may have audio, you can select which you want to hear. Either mix all the audio together (the default), play only the audio from the topmost input in the stack, or pick a particular input by name.

By default, stack inputs will be displayed so that matching ``source" frame numbers are aligned. For example if you stack foo.121-150#.exr on top of goo.56-200#.exr, you'll see frame foo.121.exr on top of goo.121.exr even though the two sequences have completely different frame ranges. If you don't want this behavior and you want the start frames of the inputs to be aligned regardless of their frame numbers, check Align Start Frames.

Also note that the Wipes mode is useful when comparing images in a stack. The use of wipes is explained in Section *4.4.3* .

11.8.3 5.6.3 Layout Views

A Layout is just what it sounds like; the inputs are arranged in a grid, column, row, or arbitrary user-defined pattern. In many ways, a Layout is similar to a stack (it even has a compositing operation for cases where you arrange on input ``over" another). All the interface actions described in Section *5.6.2* for Stack Views also apply to Layout Views.

To determine the arrangement of your layout, choose one of five modes. There are three procedural modes, which will rearrange themselves whenever the inputs are changed or reordered: Packed produces a tightly packed or tiled pattern, Row arranges all the inputs in a horizontal row, and Column arranges the inputs in a vertical column. If you want to position your inputs by hand, select the Manual mode. In this mode hovering over a given input image will show you a manipulator that can be used to reposition the image (by clicking and dragging near the center) or scale the image (by clicking and dragging the corners). After you have the inputs arranged to your liking, you may want to switch to the Static mode, which will no longer draw the manipulators, and will leave the images in your designated arrangement.

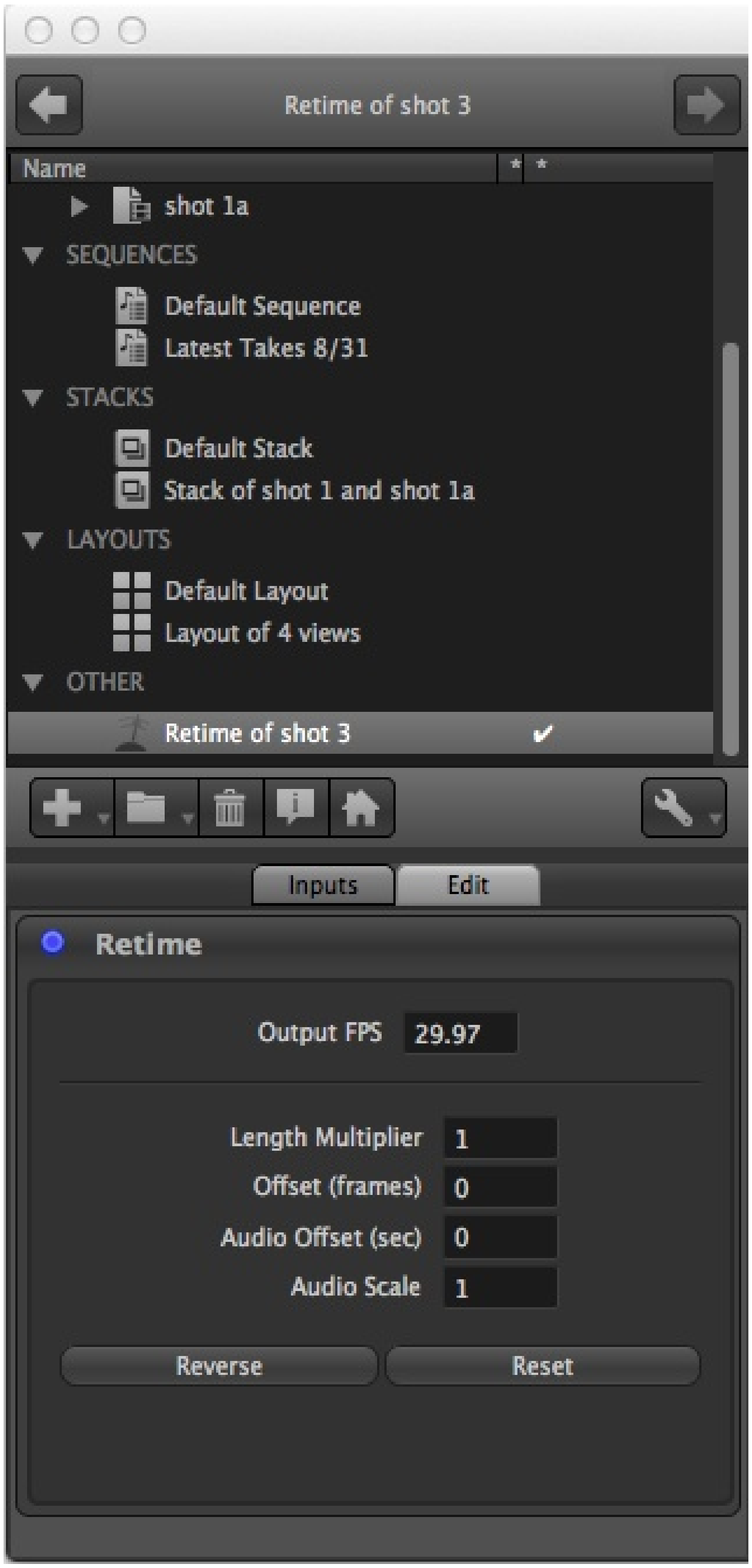
11.8.4 5.6.4 Switch Views

A Switch is a conceptually simpler than the other group views: it merely switches between its inputs. Only one input is active at a time and both the imagery and audio pass through the switch view. Otherwise, the switch shares the same output characteristics as the other group nodes (resolution, etc).

11.9 5.7 Retime View

The Retime View takes a single input and alters it's timing, making it faster or slower or offsetting the native frame numbers. For example, to double the length of an input (IE make every frame play twice, which will have the effect of slowing the action without changing the frame rate), set the Length Multiplier to 2. Or to have frame 1 of the input present itself on the output as frame 101, set the Offset to 100.

The Length Multiplier and Offset apply to both the video and audio of the input. If you want to apply an additional scale or offset to just the audio, you can use the Audio Offset and Audio Scale fields.



Output FPS 29.97

Length Multiplier 1
Offset (frames) 0
Audio Offset (sec) 0
Audio Scale 1

Figure 5.6: Retime View Edit Interface

11.10 5.8 Folders

Folders are special kind of group view used to manage the contents of the session manager. Unlike other views in RV, when you create a folder its inputs will appear as a hierarchy in the session manager. You can drag and drop and move and copy views in and out of folders to organize them. They can be used as an input just like any other view so they can be nested, placed in a sequence, stack, or layout and can be manipulated in the inputs interface in the same way other views are.

Folders have no display behavior themselves, but they can display their contents as either a switch or a layout.

You can change how a folder is displayed by selecting either Layout or Switch from its option menu.

When a view becomes a member of a folder, it will no longer appear in one of the other categories of the session manager. If a view is removed as a member of a folder, it will once again appear in one of the other categories.

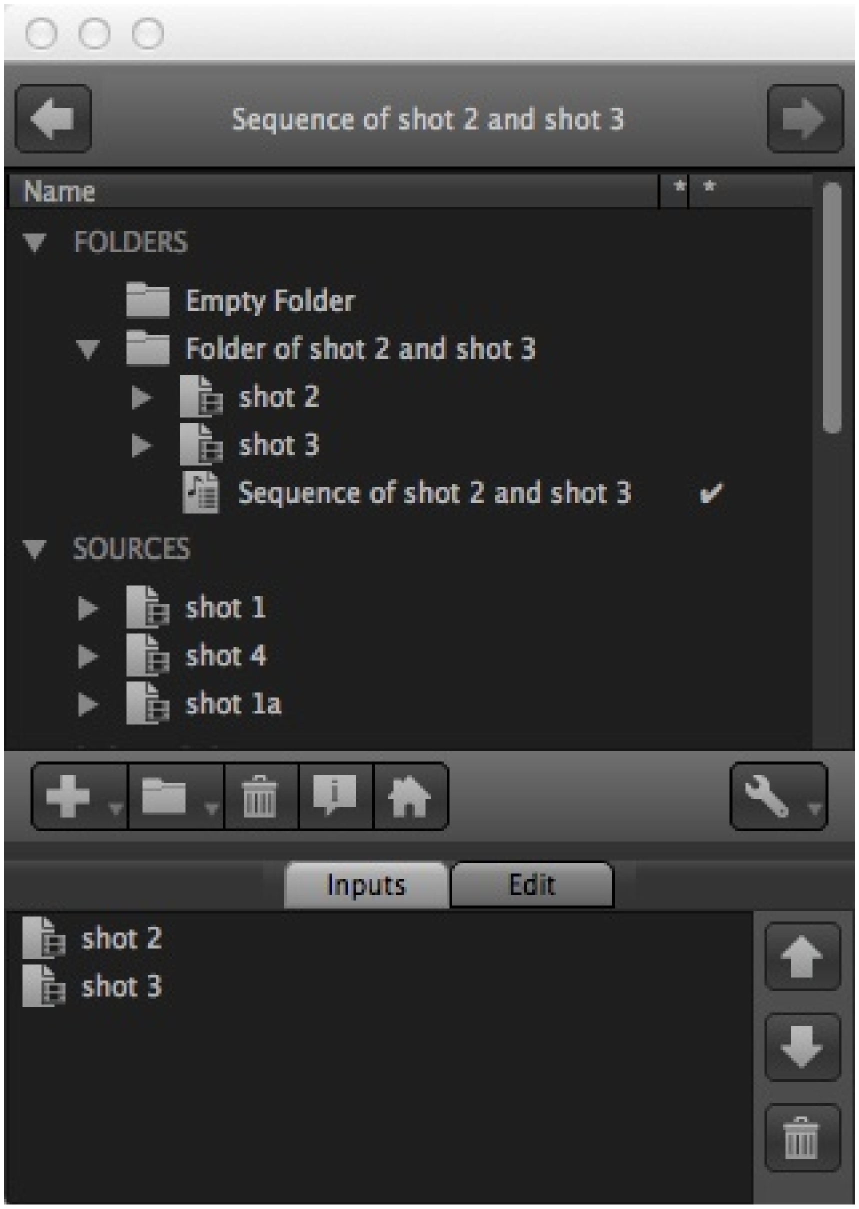

Sequence of shot 2 and shot 3
Name
FOLDERS
Empty Folder
Folder of shot 2 and shot 3
shot 2
shot 3
Sequence of shot 2 and shot 3
SOURCES
shot 1
shot 4
shot 1a
Inputs
Edit
shot 2
shot 3

Figure 5.7: Folders in the Session Manager

11.10.1 5.8.1 Folders and Drag and Drop

You can drag one or more views into a folder in the session manager to make it a member (input) of the folder. To make a copy of the dragged items hold down the drag copy modified while dragging. On the mac this is the option key, on Windows and Linux use the control key.

The session manager will not allow duplication of folder members (multiple copies of the same view in a folder) although this is not strictly illegal in RV.

Drag and drop can also be used to reorder the folder contents the same way the inputs are reordered. An insertion point will be shown indicating where the item will move to.

CHAPTER

TWELVE

CHAPTER 6 - PRESENTATION MODE AND VIDEO DEVICES

RV has a facility called presentation mode for use with machines with multiple attached display devices like projectors, second DVI/DisplayPort monitors or HDMI compatible stereo televisions.

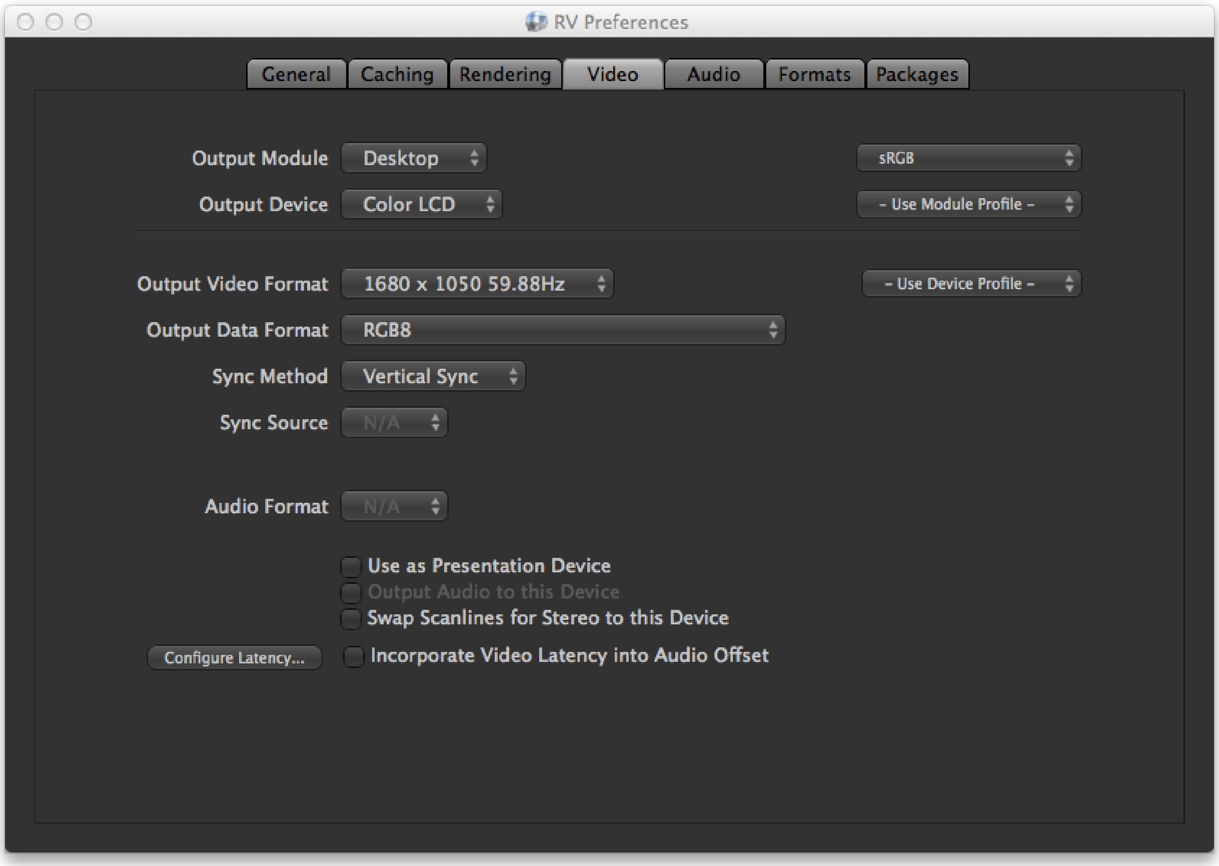
Presentation mode turns the main RV user interface into a control interface with output going to both it and a second video device. The secondary video device is always full screen. The primary use for presentation mode is multiple people viewing a session together.

Typically a video device is set up once in the preferences and used repeatedly. Presentation mode can be turned on from the View → Presentation Mode menu item. It's also possible to pass command line arguments to RV to configure and start presentation mode automatically when it launches.

12.1 6.1 Video Device Configuration

Video devices are configured from the preferences Video tab. The interface is in two parts: the module/device selection and the video configuration parameters. For each module/device combination there is a unique set of configuration parameters.

Different devices will have different configuration parameters and some devices may not use all of the available ones.



Output Module: Desktop — sRGB
Output Device: Color LCD — – Use Module Profile –
Output Video Format: 1680 x 1050 59.88Hz — – Use Device Profile –
Output Data Format: RGB8
Sync Method: Vertical Sync
Sync Source: N/A
Audio Format: N/A
☐ Use as Presentation Device
☐ Output Audio to this Device
☐ Swap Scanlines for Stereo to this Device
☐ Incorporate Video Latency into Audio Offset

Figure 6.1:

Video Preferences

Output Module

A video output module can contain multiple devices. All of the devices in a module will have the choices for configuration parameters. For example, on a system with multiple monitors there will be a Desktop output module with a device for each connected monitor. In the video preferences window, you can also specify a default display profile for the module.

Output Device

The output devices are either numbered or named depending the module. The Desktop module uses the actual monitor name and manufacturer to identify them. In the video preferences you can specify a default display profile for the specific device or indicate that the device should use the default module profile.

Output Video Format

Video formats typically include a resolution and frequency. If the video format is not changeable by the user this field will display the output resolution of the device. In the video preferences a specific profile for the selected video and data format can be assigned.

Output Data Format

The data format indicates how the the pixels are to be presented to the device. This can include the numerical precision as well as color space (e.g. RGB or YCbCr). For devices that support stereo this is usually where stereo options are found. To use HDMI 1.4a stereo modes like Side-by-Side or Top-and-Bottom, you select the appropriate Data Format here. **PLEASE NOTE** : if you use a stereo data format here, then the Stereo mode on RV's View menu should be set to "Off".

Sync Method

If the device as multiple options for synchronization these will be selectable under the sync method. For example, desktop devices typically have vertical sync on/off as an option.

Sync Source

If the synchronization can come from an external device it can be configured here. This option will usually change depending on the sync method.

Use as Presentation Device

If this is checked, the device will be used by presentation mode. Only one device can be selected as the presentation device.

Output Audio to this Device

If checked, audio will be routed to the device if it has audio capabilities.

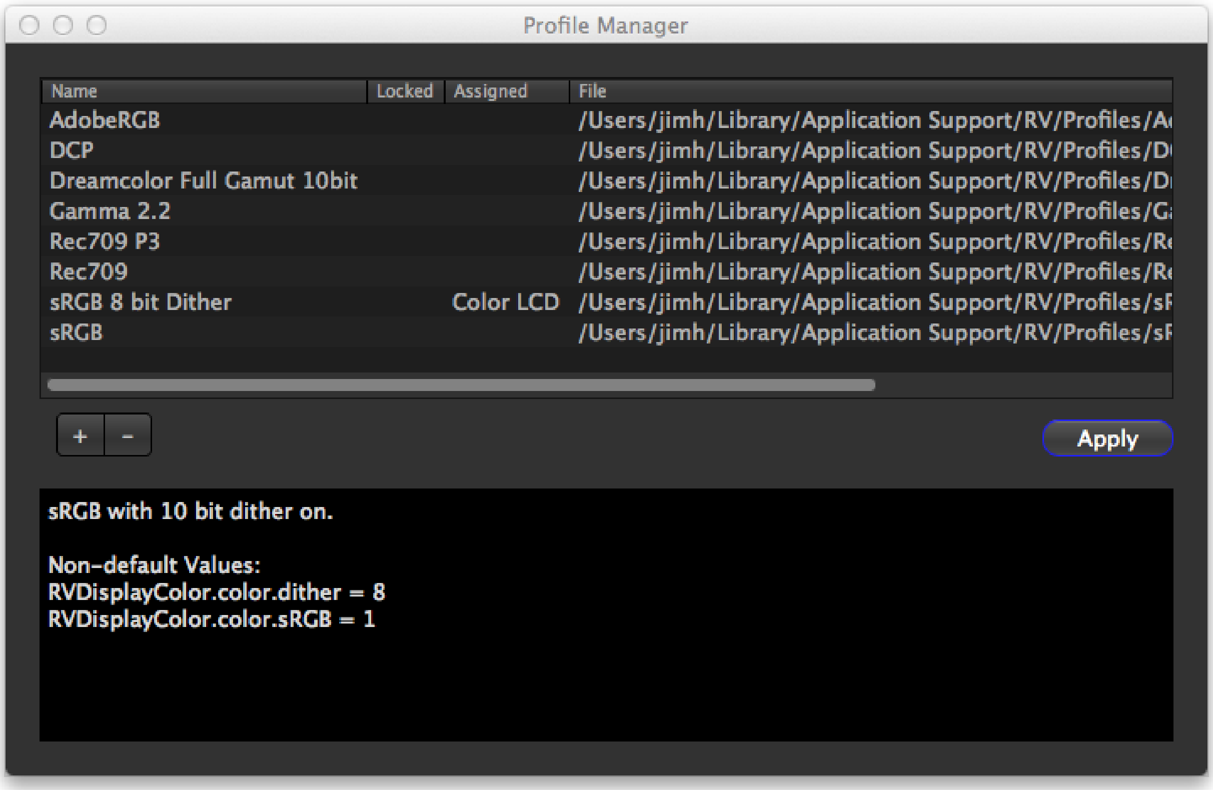
Incorporate Video Latency into Audio Offset

If checked, the total latency as indicated in the Configure Latency dialog box will be applied to the audio offset – but only if audio is being played by the controller instead of the output device (i.e. Output Audio to this Device is not checked). This is primarily useful if the video device has a large buffer causing a significant delay between the controller's audio output and the video output.

12.2 6.2 Display Profiles

Each video device configuration can have a unique display profile associated with it or can be made to use a default device or module profile. Profiles can be created and managed via View → Create/Edit Display Profiles.... Profiles are searched for in the RV_SUPPORT_PATH in the Profiles subdirectory (folder).

Display profiles are snapshots of the view settings including a display LUT if present, the transfer function (sRGB, Rec.709, Gamma 2.2, etc), the primaries, the background, view channel ordering, stereo view modes, and dithering. If a custom nodes have been defined and are used in the display color pipeline than those will also be stored in the display profile.



Name	Locked	Assigned	File
AdobeRGB			/Users/jimh/Library/Application Support/RV/Profiles/A
DCP			/Users/jimh/Library/Application Support/RV/Profiles/D
Dreamcolor Full Gamut 10bit			/Users/jimh/Library/Application Support/RV/Profiles/D
Gamma 2.2			/Users/jimh/Library/Application Support/RV/Profiles/G
Rec709 P3			/Users/jimh/Library/Application Support/RV/Profiles/R
Rec709			/Users/jimh/Library/Application Support/RV/Profiles/R
sRGB 8 bit Dither		Color LCD	/Users/jimh/Library/Application Support/RV/Profiles/s
sRGB			/Users/jimh/Library/Application Support/RV/Profiles/s

```
sRGB with 10 bit dither on.

Non-default Values:
RVDisplayColor.color.dither = 8
RVDisplayColor.color.sRGB = 1
```

Figure 6.2:

Display Profile Manager

The display profile manager can be started from the view menu. This is where profiles can be created and deleted.

When a profile is created, the values for the profile are taken from the current display device or you can select another device at creation time. Most of the view settings for the current device are present under the View menu

1

Custom or alternate nodes inserted into the RVDisplayPipelineGroup and their property values will also be stored in the profile if they are present. This makes it possible to use custom shaders, multiple display LUTs, or OpenColorIO nodes and have them be saved in the profile.

. Newly created profiles are always written to the user's local Profile area. To share a profile with other user's, the profile file can be moved to a common RV_SUPPORT_PATH location.

If a profile is already assigned to a device, the device name will appear next to the profile in the manager.

By selecting a profile and activating the Apply button, you can set the profile on the current view device. Applying a profile does not cause it to be remembered between sessions. In order to permanently assign a profile use the Video tab in the preferences.

12.3 6.3 Video Device Command Line Arguments

There are five arguments which control how presentation mode starts up from the command line:

-present	Causes the program to start up in presentation mode
-presentAudio [	Enables audio output to the presentation device if 1 or turns off if 0
-presentDevice ULE/DEVICE	Forces the use of DEVICE from MODULE. Note that the forward slash character must separate the device and module names.
-presentVideoFmat	Forces the use of format for the video format. The format is a string or substring of full description of the video format as the appear in the video module in the preferences. For example: "1080p" would match "1080p 24Hz". The first match is used.
-presentDataFomat	Forces the use of format for the data format. Like the video format above, the data format string is matched against the full description of the data formats as they appear in the video module in the preferences. For example: "Stereo" would match "Dual Stereo YCrCb 4:2:2".

Table 6.1:

Presentation Mode Command Line Arguments

The command line arguments will override any existing preferences.

12.4 6.4 Presentation Mode Settings

When the controller display mode is set to Separate Output and Control Rendering you can choose which elements of the user interface are visible on the presentation device. These can be turned on/off via the View → Presentation Settings menu. This includes not only things like the timeline and image info widget, but also whether or not the pointer location should be visible or not. In addition, you can show the actual video settings as an overlay on the display itself in order to verify the format is as expected. You can also control the display of feedback messages and remote sync pointers with items on this menu. The settings are retained in the preferences.

If you want your custom-build widget to be rendered on the Presentation Device, your widget just needs to set the Widget class data member _drawOnPresentation to true.

12.5 6.5 Platform Specific Considerations

12.5.1 6.5.1 Linux Desktop Video Module Issues

These issues apply only when using the desktop video module.

The current state of the X server XRANDR extension and client library prevents us from implementing automatic resolution switching from RV. If your distribution has the xrandr binary available and installed, you can manually use that to force the presentation monitor into the proper resolution.

When presentation mode starts up, RV will put the control window into a mode that allows tearing of the image in order to ensure that the presentation window will not tear. Be aware that the control window is no longer synced to a monitor.

nVidia Driver

RV will warn you if your presentation device is set to a monitor that the nVidia driver is not using for vertical sync. In that case you can continue, but tearing will probably occur if the attached monitors are not using identical timings. You can set the monitor which the driver syncs to using the __GL_SYNC_DISPLAY_DEVICE environment variable as per nVidia's driver documentation. See the nvidia-settings program to figure out the proper names (e.g. DPF-1 or CRT-1).

nVidia recommends setting the driver V-Sync on and turning RV's V-Sync off if possible. If not, RV will attempt to sync using the appropriate GL extension. You can turn on/off RV's V-Sync via the checkbox under Preferences → Rendering.

12.5.2 6.5.2 Mac Desktop Video Module Issues

On macOS it's not a known fact, but it appears that vertical sync is timed to the primary monitor. This is the monitor on which the menu bar appears. You can change this via the system display settings Arrangement tab.

Ideally, the presentation device will be on the primary monitor.

RV will configure the controller display to prevent interfering with the playback of the presentation monitor. The control device may exhibit tearing or other artifacts during playback. On some versions of macOS, once the controller has entered this mode, it cannot be switched back even after presentation mode has been exited.

12.5.3 6.5.3 Windows 7 Video Module Issues

On Windows, like macOS, the vertical sync is somewhat of an unknown. However, it appears that like macOS, the primary monitor (the one with the start menu) is the monitor the sync is derived from. So ideally, use the primary monitor as the presentation device, but your milage may vary.

12.6 6.6 HDMI Frame Packed Mode

Stereoscopic media can be displayed in Frame Packed (or "Frame Packing") mode on any HDMI 1.4a-compliant device (AKA 3DTV). But in order for RV to make use of this display resolution (which is roughly a double-height HD frame), the monitor or other display device must have the appropriate resolution defined in advance. The timings from the HDMI 1.4a specification are shown in the following table:

		Horizontal	Vertical
Active pixels	**1080p 24Hz**	1920	2205
	720p 60Hz	1280	1470
Front porch	**1080p 24Hz**	638	4
	720p 60Hz	110	5
Sync width	**1080p 24Hz**	48	5
	720p 60Hz	40	5
Back porch	**1080p 24Hz**	144	36
	720p 60Hz	220	20

Table 6.2:

HDMI 1.4a Frame Packed Video Timings

An additional important number is the required "pixel clock" or "pixel frequency", which should be 148.5 MHz (for either 1080p or 720p),.

12.6.1 Linux

Sample Xorg.conf lines for 1080p 24Hz (720p 60Hz commented out), using the timings from the table above:

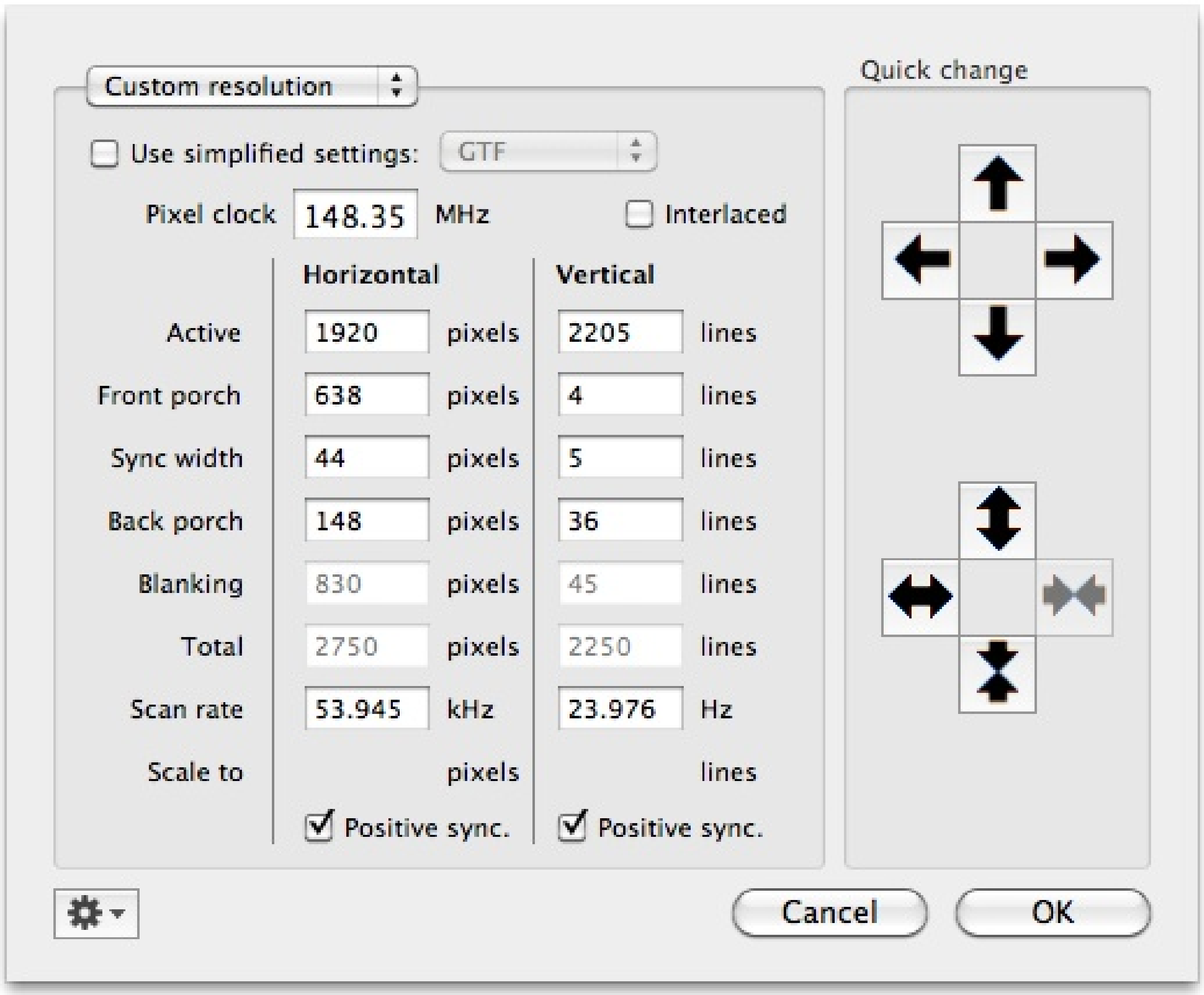
```
Section "Monitor"
...
Modeline  "1920x2205_24" 148.32 1920 2558 2602 2750 2205 2209 2214 2250 +hsync +vsync
#Modeline "1280x1470_60" 148.5  1280 1390 1430 1650 1470 1475 1480 1500 +hsync +vsync
...
EndSection
Section "Device"
...
Option "ModeValidation" "NoDFPNativeResolutionCheck"
Option "ExactModeTimingsDVI" "True"
...
EndSection
Section "Screen"
...
Option "metamodes" "DFP-0: 1920x1080_60 +0+0, DFP-2: 1920x2205_24 +1920+0"
...
EndSection
```

12.6.2 Mac

On OSX, we have used the shareware utility SwitchResX to add a Frame Packed resolution. Once the 1920x2205 resolution is visible in the Display preferences, it will show up in the Output Video Format selector on the Video Preference for the appropriate output device. If this device is selected as the presentation device, the monitor will go in and out of this mode when presentation mode is turned on and off.

It’s possible that there are various limitations to this approach. Our testing was on a MacBook Pro with Nvidia GFX (9600M GT), and we have reports of this working on a MacPro with Nvidia GFX (Quadro 4000) and a MacPro with ATI GFX (Radeon HD 5770), although the latter required some adjustments. Also, we have no reports of this working on OSX 10.7 yet. In one case, we haven’t been able to make this work with a MacBookPro running 10.7 with ATI GFX. So, bottom line is that it can be made to work in some cases, but we can’t guarantee it.

Here’s a screengrab of the SwitchResX custom resolution dialog, showing our 1920x2205 mode:



Use simplified settings: GTF

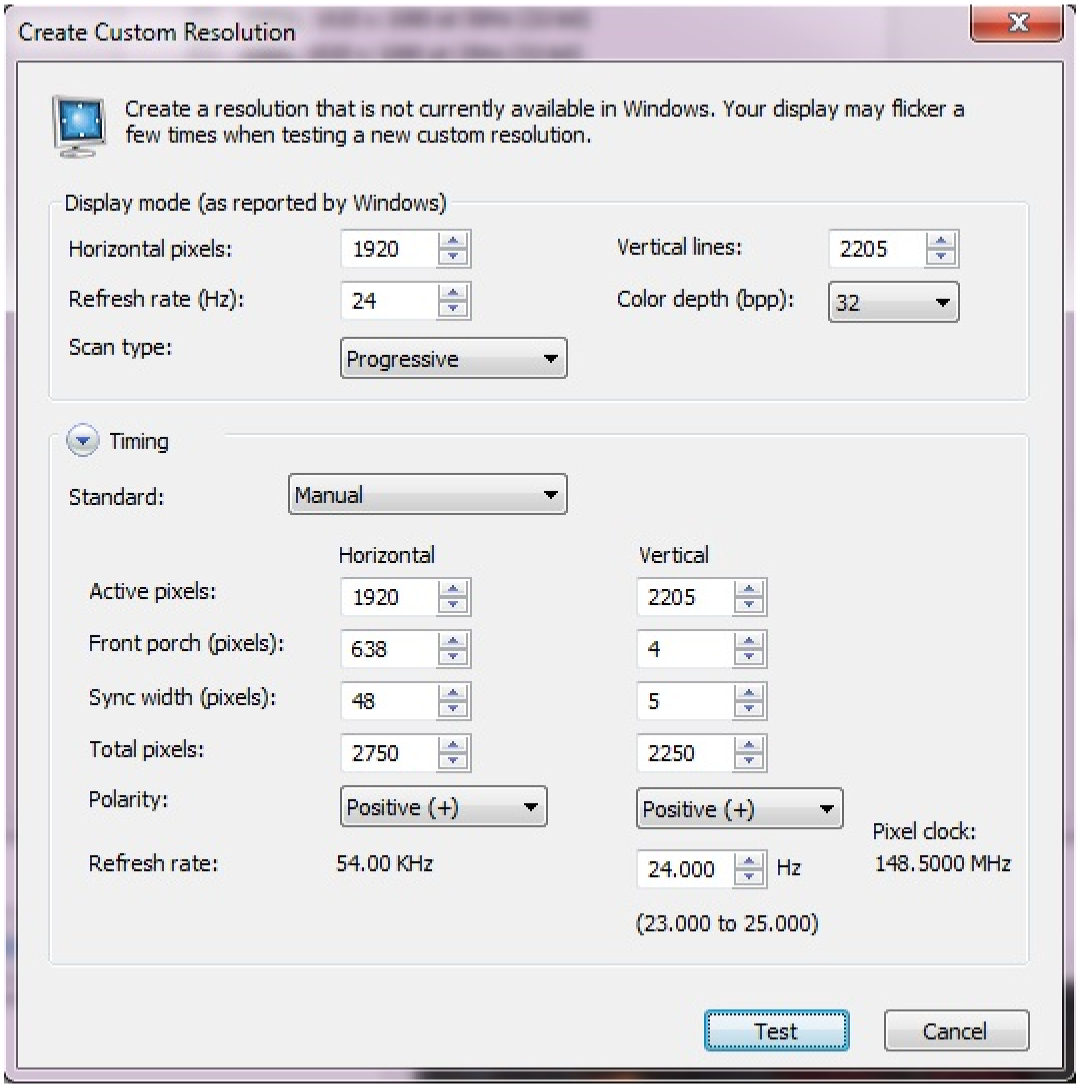
Pixel clock 148.35 MHz   Interlaced

	Horizontal		Vertical	
Active	1920	pixels	2205	lines
Front porch	638	pixels	4	lines
Sync width	44	pixels	5	lines
Back porch	148	pixels	36	lines
Blanking	830	pixels	45	lines
Total	2750	pixels	2250	lines
Scan rate	53.945	kHz	23.976	Hz
Scale to		pixels		lines
	✓ Positive sync.		✓ Positive sync.	

12.6.3 Windows

On Windows, you can use the NVIDIA control panel to create a custom resolution with the above timings. Go to "Change Resolution", then "Custimize", then "Create Custom Resolution", then "Timing", for the "Standard", choose "Manual" and input the timings described above. Please note that RV cannot put the monitor in this Custom Resolution (the windows API does not expose the custom resolutions), so you need to choose the custom mode yourself (in the NVIDIA control panel, "Change Resolution" section) before you start RV. When RV starts, it will add the current (custom) mode to the Video Formats list in the Video Preferences tab, and you can select it there, then select the "Frame Packed" Data Format.

Here's an example the timing setup that has worked for us:



Create Custom Resolution

Create a resolution that is not currently available in Windows. Your display may flicker a few times when testing a new custom resolution.

Display mode (as reported by Windows)

Horizontal pixels:	1920	Vertical lines:	2205
Refresh rate (Hz):	24	Color depth (bpp):	32
Scan type:	Progressive		

Timing

Standard: Manual

	Horizontal	Vertical
Active pixels:	1920	2205
Front porch (pixels):	638	4
Sync width (pixels):	48	5
Total pixels:	2750	2250
Polarity:	Positive (+)	Positive (+)
Refresh rate:	54.00 KHz	24.000 Hz (23.000 to 25.000)

Pixel clock: 148.5000 MHz

CHAPTER

THIRTEEN

CHAPTER 7 - HOW A PIXEL GETS FROM A FILE TO THE SCREEN

RV has a well defined image processing pipeline which is implemented as a combination of software and hardware (using the GPU when possible). Figure *7.1* shows the pixel pipeline.

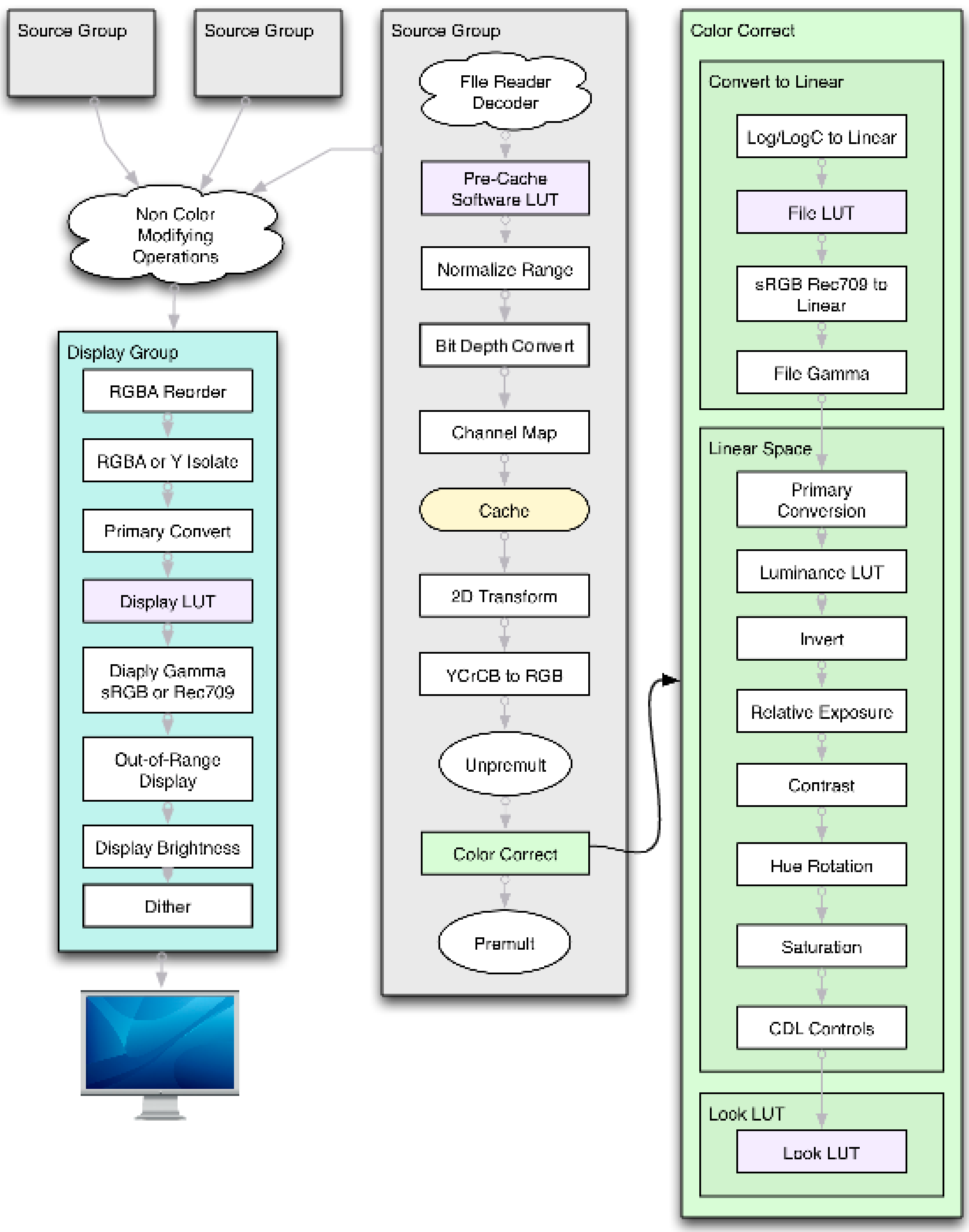


Figure 7.1: RV Pixel Pipeline

13.1 7.1 Image Layers

Each image source may be composed of one or more layers. Layers may come from multiple files, or a single file if the file format supports it or a combination of the two. For example a stereo source can be constructed from a left and right movie file; in that case each file is a layer. Alternately, layers may come from a single file as would be the case with a stereo QuickTime file or EXR images with left and right layers.

An image source may have any number of layers. By default, only the first layer is visible in RV unless an operation exposes the additional layers.

13.1.1 7.1.1 Stereo Layers

RV has a number of stereo viewing options which render image layers to a left and right eye image. The left and right eye images are both layers. RV doesn't require any specific method of storing stereo images: you can store them in a single movie file as multiple tracks, as multiple movie files, or as multiple image sequences. You can even have one eye be a completely different format than the other if necessary. Stereo viewing is discussed in Chapter *12* .

13.2 7.2 Image Attributes

RV tries to read as many image attributes as possible from the file. RV may also add attributes to the image to indicate things like pixel aspect ratio, alpha type, uncrop regions (data and display windows) and to indicate the color space the pixels are in. The image info window in the user interface shows all of the relevant image attributes.

Some of the attributes are treated as special cases and can have an effect on rendering. Internally, RV will recognize and use the **ColorSpace/Primary** attributes automatically. Other **ColorSpace** attributes are used by the default source setup package (See Reference Manual) to set file to linear properties correctly. For example, if the **ColorSpace/Transfer** attribute has the value "Kodak Log", the default source setup function will automatically turn on the Kodak log to linear function for that source.

Image attributes can be saved as a text file directly from the UI (File → Export → Image Attributes), viewed interactively with the Tools → Image Info widget, or using the rvls program from the command line.

Attribute	How It's Used
ColorSpace/Primaries	Documents the name of the primary space if it has one
ColorSpace/Transfer	Contains the name of the transfer function used to convert non-linear R G B values to linear R G
ColorSpace/Conversion	If the image is encoded as a variant of luminance + chroma this attribute documents the name of t
ColorSpace/ConversionMatrix	If the luminance + chroma conversion matrix is explicitly given, this attribute will contain it
ColorSpace/Gamma	If the image is gamma encoded the correction gamma is stored here
ColorSpace/Black Point	Explicit value for Kodak log to linear conversion if it exists or related functions which require a bl
ColorSpace/White Point	
ColorSpace/Rolloff	Explicit value for Kodak log to linear conversion
ColorSpace/Red Primary	Explicit primary values. These are used directly by the renderer unless told to ignore them.
ColorSpace/Green Primary	
ColorSpace/Blue Primary	
ColorSpace/White Primary	
ColorSpace/AdoptedNeutral	Indicates adopted color temperature (white point).
ColorSpace/RGBtoXYZMatrix	Explicit color space conversion matrix. This may be used instead of the primary attributes to dete
ColorSpace/LogCBlackSignal	LogC black signal. Black will be mapped to this value. The default is 0.
ColorSpace/LogCEncodingOffset	Derived from camera parameters.
ColorSpace/LogCEncodingGain	Derived from camera parameters.

Table 1 – continued from previous page

Attribute	How It's Used
ColorSpace/LogCGraySignal	The value mapped to 18% grey. The default is 0.18.
ColorSpace/LogCBlackOffset	Derived from camera parameters.
ColorSpace/LogCLinearSlope	Derived from camera parameters.
ColorSpace/LogCLinearOffset	Derived from camera parameters.
ColorSpace/LogCCutPoint	Indirectly determines the final linear/non-linear cut off point along with other parameters.
ColorSpace/ICC Profile Name	The name and data of an embedded ICC profile
ColorSpace/ICC Profile Data	
PixelAspectRatio	Pixel aspect ratio from file
DataWindowSize	Uncrop parameters if they appeared in a file
DataWindowOrigin	
DisplayWindowSize	
DisplayWindowOrigin	

Table 7.1:

Basic Special Image Attributes

13.3 7.3 Image Channels

RV potentially does a great deal of data conversion between reading a file and rendering an image on your display device. In some cases, you will want to have control over this process so it's important to understand what's occurring internally. For example, when RV reads a typical RGB TIFF file, you can assume the internal representation is a direct mapping from the data in the file. If, on the other hand, RV is reading an EXR file with A, B, G, T, and Z channels, and you are interested in the contents of the Z channel, you will need to tell RV specifically how to map the image data to an RGBA pixel.

To see what channels an image has in it and what channels RV has decided to use for display you can select Tools → Image Info in the menu bar. The first two items displayed tell you the internal image format. In some cases you will see an additional item called ChannelNamesInFile which may show not only R, G, B channels, but additional channels in the file that are not being shown.

RV stores images using between one and four channels. The channels are always the same data type and precision for a given image. If an image file on disk contains channels with differing precision or data type, the reader will choose the best four channels to map to RGBA (or fewer channels) and a data type and precision that best conserves the information present in the file. If there is no particular set of channels in the image that make sense to map to an internal RGBA image, RV will arbitrarily map up to the first four channels in order. By default, RV will interpret channel data as shown in Table *7.2*

# of Channels	Names	RGBA Mapping
1	Y	YYY1
2	Y, A	YYYA
3	R, G, B	RGB1
4	R, G, B, A	RGBA

Table 7.2: Mapping of File Channels to Display Channels

Default interpretation of channels and how they are mapped to display RGBA. ``1" means that the display channel is filled with the value 1.0. ``Y" is luminance (a scalar image).

When reading an image type that contains pixel data that is not directly mappable to RGB data (like YUV data), most of RV's image readers will automatically convert the data to RGB. This is the case for JPEG, and related image formats (QuickTime movies with JPEG compression for example). If the pixel data is not converted from YUV to RGB, RV will convert the pixels to RGB in hardware (if possible).

13.3.1 7.3.1 Precision

RV natively handles both integer and floating point images. When one of RV's image readers decides a precision and data type for an image, all of its channels are converted to that type internally.

Channel Data Type	Display Range	Relative Memory Consumption
8 bit int	[0.0 , 1.0]	1
16 bit int	[0.0 , 1.0]	2
16 bit float	[- inf , inf]	2
32 bit float	[- inf , inf]	4

Table 7.3: Characteristics of Channel Data Types

RV lets you modify how the images are stored internally. This ability is important because different internal formats can require different amounts of memory (see Table *7.3*). In some cases you will want to reduce or increase that memory requirement to fit more images into a cache or for faster or longer playback.

You can force RV to use a specific precision in the interface Color → Color Resolution menu. There are two options here: (1) whether or not to allow floating point and (2) the maximum bit depth to use.

In the RV image processing DAG, precision is controlled by the Format node. There are two properties which determine the behavior: **color.maxBitDepth** and **color.allowFloatingPoint** .

13.3.2 7.3.2 Channel Remapping

RV provides a few similar functions which allow you to remap image channels to display channels. The most general method is called Channel Remapping.

When Channel Remapping is active, RV reads all of the channel data in an image. This may result in images with too many channels internally (five or more), so RV will choose four channels to map to RGBA.

You specify exactly which channels RV will choose and what order they should be in. The easiest way to accomplish this is in the user interface. By executing Tools → Remap Source Image Channels…, you can type in the names of the channels you want mapped to RGBA separated by commas. Using the previous example of an EXR with A, B, G, R, Z channels and you'd like to see Z as alpha, you could type in R,G,B,Z when prompted. If you'd like to see the value of Z as a greyscale image, you could type in Z or Z,A if you want to see the alpha along with it.

It is also possible to add channels to incoming images using Channel Remapping. If you specify channel names that do not exist in the image file, new channels will be created. This is especially useful if you need to add an alpha channel to a three channel RGB image to increase playback performance

1

New channels currently inherit the channel data from the first channel in the image. If the data needs to be 1.0 or 0.0 in the new channel, use Channel Reorder to insert constant data.

Channel Remapping is controlled by the ChannelMap node. The names of the channels and their order is stored in the **format.channels** property. Channel Remapping occurs when the image data in the file is converted into one of the internal image formats.

Note that there is overlapping functionality between Channel Remapping and Channel Reordering (See *7.8.1*) and Channel Isolation (See *7.8.2*) which are described below. However, Channel Remapping occurs just after pixels are read from a file. Channel Reordering and Isolation occur just before the pixel is displayed and typically happen in hardware. Channel Remapping always occurs in software.

13.4 7.4 Crop and Uncrop

Cropping an image discards pixels outside of the crop region. The image size is reduced in the process. This can be beneficial when loading a large number of cached images where only a small portion of the frame is interesting or useful (e.g. a rendered element). For some formats, RV may be able to reduce I/O bandwidth by reading and decoding pixels only within the crop region.

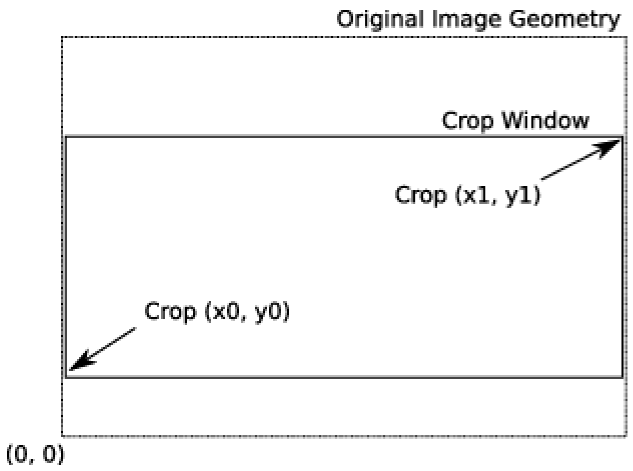


Figure 7.2:

Cropping Parameters. Note that (x1, y1) are coordinates.

Note that the four crop parameters describe the bottom and top corners not the origin, width, and height. So a “crop” of the entire image would be (0 , 0) to (*w* - 1 , *h* - 1) where *w* is the image width and *h* is the image height.

Uncropping (in terms of RV) creates a virtual image which is typically larger than the input image

2

Quicktime calls this same functionality “clean aperture.” The OpenEXR documentation refers to something similar as the data and display windows.

. The input image is usually placed completely inside of the larger virtual image.

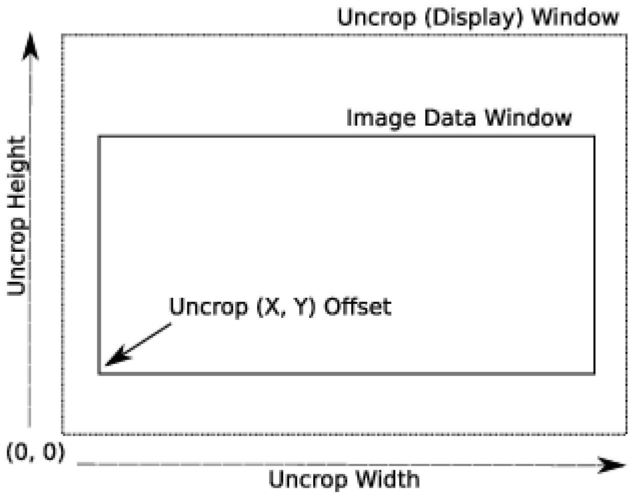


Figure 7.3:

Uncrop Parameters

It is also possible to uncrop an image to a smaller size (in which case some pixels are beyond the virtual image border) or partially in and out of the virtual uncrop image. This handles both the cases of a cropped render and an a render of pixels beyond the final frame for compositing slop.

The OpenEXR format includes a display and data window. These are almost directly translated to uncrop parameters except that in RV the display window always has an origin of (0 , 0) . When RV encounters differing display and data window attributes in an EXR file it will automatically convert these to uncrop values. This means that a sequence of EXR frames may have unique uncrop values for each frame.

Currently EXR is the only format that supports per-frame uncrop in RV.

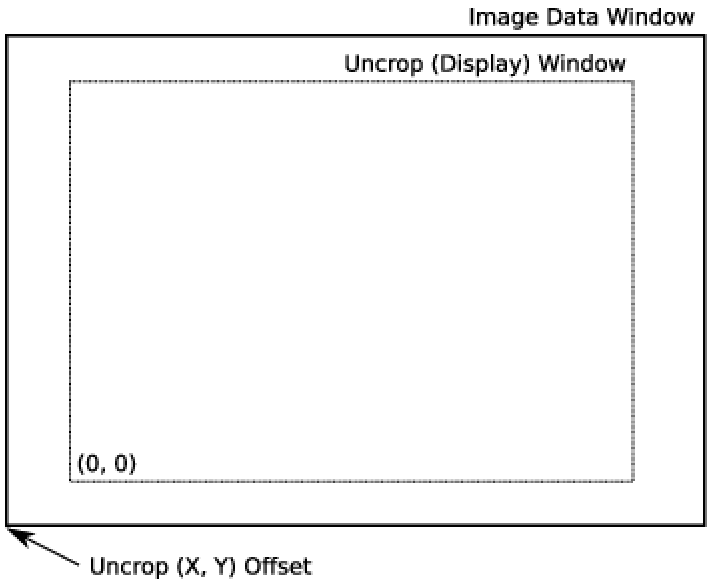


Figure 7.4:

Uncrop of Oversized Render

RV considers the uncropped image geometry as the principle image geometry. Values reported relating to the width and height in the user interface will usually refer to the uncropped geometry. Wipes, mattes, and other user interface will be drawing relative to the uncropped geometry.

13.5 7.5 Conversion to Linear Color Space

If an image format stores pixel values in a color space which is non-linear, the values should be converted to linear before any color correction or display correction is applied. In the ICC and EXR documentation, linear space is also called *Scene Referred Space*. The most important characteristic of scene referred space is that doubling a value results in twice the luminance.

Although any image format can potentially hold pixel data in a non-linear color space, there are few formats which are designed to do so. Kodak Cineon files, for example, store values in a logarithmic color space.

3

The values in Cineon files are more complex than stated here. Color channel values are stored as code values which correspond to *printing density* – not luminance – as is the case with most other image file formats. Furthermore, the conversion to linear printing density is parameterized and these parameters can vary depending on the needs of the user. See the Kodak documentation on the Cineon format for a more detail description.

JPEG images may be stored in ``video” space

4

This could mean NTSC color space or something else!

which typically has a 2.2 gamma applied to the color values for better viewing on computer monitors. EXR files on the other hand are typically stored in linear space so no conversion need be applied.

If values are not in linear space, color correction and display correction transforms can still be applied, but the results will not be correct and in some cases will be misleading. So it’s important to realize what color space an image is in and to tell RV to linearize it.

13.5.1 7.5.1 Non-Rec. 709 Primaries

If an image has attributes which provide primaries, RV will use this information to transform the color to the standard REC 709 primaries RGB space automatically. When the white points do not differ, this is done by concatenating two matrices: a transform to CIE XYZ space followed by a transform to RGB 709 space. When the white points differ chromatic adaptation is used.

13.5.2 7.5.2 Y RY BY Conversion

OpenEXR files can be in stored as Y RY BY channels. The EXR reader will pass these files to RV as planar images (three separate images instead of one image with three channels). RV will then recombine the images in hardware into RGBA.

This is advantageous when one or more of the original image planes are subsampled. Subsampled image planes have a reduced resolution. Typically the chroma channels (in this case RY and BY) are subsampled.

13.5.3 7.5.3 YUV (YCbCr) Conversion

Some formats may produce YUV or YUVA images to be displayed. RV can decode these in hardware for better performance. RV uses the following matrix to transform YUV to RGB:

1	0	1.402
1	- 0.344136	- 0.714136
1	1.772	0
	Y	
	U	
	V	

where

0	*Y*	1
- 0.5	*U*	0.5
- 0.5	*V*	0.5

In the case of Photo-JPEG, the YUV data coming from a movie file is assumed to use the full gamut for the given number of bits. For example, an eight bit per channel image would have luminance values of [0, 255]. For Motion-JPEG a reduced color gamut is assumed.

On hardware that cannot support hardware conversion, RV will convert the image in software. You can tell which method RV is using by looking at the image info. If the display channels are YUVA the image is being decoded in hardware

5

Conversion of YUVA to RGBA in hardware is an optimization that can result in faster playback on some platforms.

. On Linux, this option must be specified when RV is started using the -yuv flag.

6

Note that the color inspector will convert these values to normalized RGBA.

13.5.4 7.5.4 Log to Linear Color Space Conversion

When RV reads a 10 bit Cineon or DPX file, it converts 10 bit integer pixel values into 16 bit integer values which are typically in a logarithmic color space. In order to correctly view one of these images, it is necessary to transform the 16 bit values into linear space. RV has two transforms for this conversion. You can access the function via the Color menu from the menu bar. For standard Cineon/DPX RV uses the default Kodak conversion parameters.

For Shake users, RV produces the same result as the default parameters for the LogLin node. Alternately, if the image was created with a Grass Valley Viper FilmStream camera, you should use the Viper log conversion.

The Kodak Log conversion can produce values in range [0, ~13.5]. To see values outside of [0,1] use the exposure feature to stop down image. A value of -3.72 for relative exposure will show all possible output values of the Kodak log to linear conversion.

13.5.5 7.5.6 File Gamma Correction

If the image is stored in ``video" or ``gamma" space, you can convert to linear by applying a gamma correction. RV has three gamma transforms: one for linearization, one for display correction, and one in the middle for color correction. The gamma color correction can be activated via the Color → File Gamma menu items. When an image is stored in gamma space, it typically is transformed by the formula *c* 1 where is the gamma value and *c* is the channel value. When transforming back to linear space it's necessary to apply the inverse (*c*). RV's file gamma applies *c* while the color correction gamma and the display gamma apply *c* 1 (in all cases refers to the value visible in the interface).

13.5.6 7.5.7 sRGB to Linear Color Space Conversion

If an image or movie file was encoded in sRGB space with the intention of making it easy to view with non-color aware software, RV can convert it to linear on the fly using this transform. In addition, some file LUTs are created to transform imagery from file space to sRGB space directly. If you apply one of these LUTs to an incoming image as a file LUT in RV you can then using the sRGB to linear function to linearize the output pixels.

The sRGB to linear function can be activated via Color → sRGB and is stored per-source like the file gamma above or log to linear.

RV uses the following equation to go from sRGB space to linear:

clinear =

$csRGB12.92$	$csRGB\ \ p$
$(csRGB + a1 + a)2.4$	$csRGB > p$

where

$p = 0.04045$

$a = 0.055$

13.5.7 7.5.8 Rec. 709 Transfer to Linear Color Space Conversion

Similar to sRGB, The Rec. 709 non-linear transfer function is a gamma-like transform. High definition television imagery is often encoded using this color space. In order to view it properly on a computer monitor, it should be converted to linear and then to sRGB for display. The Rec. 709 to linear function can be activated via Color → Rec709 and is stored per-source like the file gamma, sRGB, or log to linear.

RV uses the following equation to go from Rec. 709 space to linear:

clinear =

*c709*4.5	*c709* *p*
(*c709* + 0.0991.099)10.45	*c709* > *p*

where

p = 0.081

13.5.8 7.5.9 Pre-Cache and File LUTs

RV has four points in its pixel pipeline where LUTs may be used. The first of these is the pre-cache LUT. The pre-cache LUT is applied in software, and as the name implies, the results go into the cache. The primary use of the pre-cache LUT is to convert the image colorspace in conjunction with a bit depth reformatting to maximize cache use.

The file LUT and all subsequent LUTs are applied in hardware and are intended to be used as a conversion to the color working space (usually some linear space). However, the file LUT can also be used for as a transform from file to display color space if desired.

See Chapeter *8* for more information about how LUTs work in RV.

13.5.9 7.5.10 File CDLs

In much the same way you can assign a File LUT to help linearize a file source into the working space in RV you can also use a file CDL. This CDL is applied before linearization occurs. There is also a look slot for CDL information described later.

See Chapter *9* for more information about how LUTs work in RV.

13.6 7.6 Color Correction

None of the color corrections affects the Alpha channel. For a good discussion on linear color corrections, check out Paul Haeberli's Graphica Obscura website Graphica Obscura .

Color corrections are applied independently to each image source in an RV session. For example, if you have two movies playing in sequence in a session and you change the contrast, it will only affect the movie that is visible when the contrast is changed. (If you want to apply the same color correction to all sources, perform the correction in tiled mode: Tools → Stack, Tools → Tile.)

13.6.1 7.6.1 Luminance LUTs

After conversion to linear, pixels may be passed though a luminance look up table. This can be useful when examining depth images or shadow maps. RV has a few predefined luminance LUTs: HSV, Random, and a number of contour LUTs. Each of these maps a luminance value to a color.

13.6.2 7.6.2 Relative Exposure

RV's computes relative exposure like this:

$c \times 2 exposure$

where c is the incoming color. So a relative exposure of -1.0 will cause the color to be divided by 2.0.

Relative exposure can alternately be thought of as increasing or decreasing the stop on a camera. So a relative exposure of -1.0 is equivalent to viewing the image as if the camera was stopped down by 1 when the picture was taken.

To map an exact range of values (where 0 is always the low value) set the exposure to log21*max* where *max* is the upper bound.

13.6.3 7.6.3 Hue Rotation

The unit of hue rotation is radians. RV's hue rotation is luminance preserving. A hue rotation of 2 will result in no hue change.

The algorithm is as follows:

- Apply a rotation that maps the grey vector to the blue axis.
- Compute the vector L that is perpendicular to the plane of constant luminance.
- Apply a skew transform to map the vector L onto the blue axis.
- Apply a rotation about the blue axis N radians where N is the amount of hue change.
- Apply a rotation that maps the blue axis back to the grey vector.

RV computes luminance using this formula

8

This is also the formula used for luminance display.

:

Rw	*Gw*	*Bw*	*0*
Rw	*Gw*	*Bw*	*0*
Rw	*Gw*	*Bw*	*0*
0	*0*	*0*	*1*
	R		
	G		
	B		
	1		

where

9

Weight values for R, G, and B are applicable in linear color space. Values used for determining luminance for NTSC video are not applicable in linear color space.

Rw = 0.3086 Gw = 0.6094 Bw = 0.0820

13.6.4 7.6.4 Relative Saturation

RV applies the formula:

Rw(1 - s) + s	*Gw(1 - s)*	*Bw(1 - s)*	*0*
Rw(1 - s)	*Gw(1 - s) + s*	*Bw(1 - s)*	*0*
Rw(1 - s)	*Gw(1 - s)*	*Bw(1 - s) + s*	*0*
0	*0*	*0*	*1*
	R		
	G		
	B		
	1		

where

Rw = 0.3086 *Gw* = 0.6094 *Bw w* = 0.0820

and *s* is the saturation value.

13.6.5 7.6.5 Contrast

RV applies the formula:

1 + k	*0*	*0*	*- k2*
0	*1 + k*	*0*	*- k2*
0	*0*	*1 + k*	*- k2*
0	*0*	*0*	*1*
	R		
	G		
	B		
	1		

where *k* is the contrast value.

13.6.6 7.6.6 Inversion

RV applies the formula:

-1	*0*	*0*	*0*
0	*-1*	*0*	*0*
0	*0*	*-1*	*0*
1	*1*	*1*	*1*
	R		
	G		
	B		
	1		

13.6.7 7.6.7 ASC Color Decision List (CDL) Controls

ASC-CDL controls are as follows:

SOP	=	*Clamp (Cin * slope + offset)power*

where *slope* , *offset* , and *power* are per-channel parameters

The CDL saturation is then applied to the result like so:

Clamp(

Rw(1 - s) + s	*Gw(1 - s)*	*Bw(1 - s)*	*0*
Rw(1 - s)	*Gw(1 - s) + s*	*Bw(1 - s)*	*0*
Rw(1 - s)	*Gw(1 - s)*	*Bw(1 - s) + s*	*0*
0	*0*	*0*	*1*
	SOPR		
	SOPG		
	SOPB		
	1		
)			

where

Rw = 0.2126 Gw = 0.7152 Bw w = 0.0722

and s is the saturation when s 0

13.7 7.7 Display Simulation and Correction

After color corrections have been applied in linear space and before pixels are sent to the display device, they undergo display transformations. These transforms are intended to simulate the appearance of pixels on alternate display devices (like film) and to correct for any color transform that will be applied by the primary display device. In addition, RV provides a few tools to help visualize image pixel values in various ways.

Unlike color corrections, display color corrections apply to all source material in an RV session.

13.7.1 7.7.1 Display and Look LUTs

There are times when it's necessary to have two separate display LUTs — one which might be per-shot and one which is global. For example, this can happen when digital intermediate color work is being done on plates while the uncorrected plates are being worked on; a temporary LUT may be needed to simulate the ``look" of the final result.

There is a unique look LUT per source. There is a single display LUT for an RV session.

RV applies the look LUT just before the display LUT.

See Chapter *8* for a more detailed explanation of usage and how to load a LUT into RV.

13.7.2 7.7.2 Display Gamma Correction

This gamma correction is intended to compensate for monitor gamma. It is not related to File Gamma Correction, which is discussed in Section *7.5.6* .

For a given monitor, there is usually one good value (e.g., 2.2) which when applied corrects the monitor's response to be nearly linear. Note that you should not use the Display Gamma Correction if your monitor has been calibrated with a gamma correction built in.

If you are using the X Window System (Linux/Unix) or Microsoft Windows, the default is not to add a gamma correction for the monitor.

On X Windows, this can be checked using the xgamma command. For example, in a shell if you type:

```
shell> xgamma
```

and you see:

```
-> Red 1.000, Green 1.000, Blue 1.000
```

then your display has no gamma correction being applied. In this case you will want RV to correct for the non-linear response by setting the Display Gamma to correct for the monitor (e.g., 2.2).

On macOS, things are more complex. Typically, a ColorSync profile is created for your monitor and this includes a gamma correction. However, the monitor may be corrected to achieve a non-linear response. The best bet on macOS is to calibrate the display in such a way that a linear response is achieved and don't use a Display Gamma other than 1.0 in RV.

The Display Gamma can be set from the View menu.

13.7.3 7.7.3 sRGB Display Correction

Most recently made monitors are built to have an sRGB response curve. This is similar to a Gamma 2.2 response curve, but with a more linear function in the blacks. RV supports this function directly for both input and output without the need to use a LUT. The sRGB display is a better default for most monitors than the display gamma correction above.

RV's default rules for color set up use the sRGB display. RV also assumes by default that QuickTime movies and JPEG files are in sRGB color space unless they specifically indicate otherwise. This behavior can be overridden (see the Reference Manual).

RV uses the following formula to convert from linear to sRGB:

csRGB =

12.92*clinear*	*clinear q*
(1 + *a*)*c*1/2.4*linear* - *a*	*clinear* > *q*

where

q = 0.0031308

a = 0.055

The sRGB Display Correction can be set from the View menu.

13.7.4 7.7.4 Rec. 709 Non-Linear Transfer Display Correction

If the display device is an HD television or reference monitor it may be naturally calibrated to the Rec. 709 color space. Similar to sRGB, Rec. 709 is a gamma-like curve. RV uses the following formula to convert from linear to Rec. 709:

*c*709 =

4.5*clinear*	*clinear q*
1.099*clinear*0.45 - 0.099	*clinear* > *q*

where

q = 0.018

13.7.5 7.7.5 Display Brightness

There is a final multiplier on the color which can be made after the display gamma. This is analogous to relative exposure discussed above. The Display Brightness will never affect the hue of displayed pixels, but can be used to increase or decrease the final brightness of the pixel.

This is most useful when a display LUT is being used to simulate an alternate display device (like projected film). In some cases, the display LUT may scale luminance of the image down in order to represent the entire dynamic range of the display device. In order to examine dark parts of the frame, you can adjust the display brightness without worrying about any chromatic changes to the image. Note that bright colors may become clipped in the process.

The Display Brightness can be set from the View menu.

13.8 7.8 Final Display Filters

These operations occur after the display correction has been applied and before the pixel is displayed. Unlike color corrections, there is only a single instance of each of these for an RV session. So if you isolate the red channel for example, it will be isolated for all source material.

13.8.1 7.8.1 Channel Reorder

If you have RGBA or fewer channels read into RV and you need to rearrange them for some reason, you can do so without using the general remapping technique described above. In that case you can use channel reordering which makes it possible to reorder RGBA channels and set one or more of the channels to 0.0 or 1.0. Channel Reordering can be accessed in the menu bar under View → Channel Order. On supported machines, this function is implemented in hardware (so it's very fast).

This function can be useful if the image comes from a format with unnamed RGB channels which are not in order. Another useful feature of Channel reordering is the ability to flood one or more channels with the constant value of 1.0 or 0.0. For example to see the red channel as red without green and blue you could set the order to R000 (R followed by three zeros). To see the red channel as a grey scale image you could use the order RRR0. (Yes, that is exactly what channel isolation does! See below.)

Channel reordering is controlled by the Display node. The **display.channelOrder** attribute determines the order. Channel reordering occurs when the internal image is rendered to the display.

13.8.2 7.8.2 Isolating Channels

Finally, you can isolate the view to any single channel using the interface from the menu View → Channel Display. Since this is the most common operation when viewing channels, it is mapped to the keys ``r”, ``g”, ``b”and ``a” by default (shows isolated red, green, blue, and alpha) and can be reset with the ``c” key (show all color channels). You can also view luminance as a pseudo-channel with the ``l” key.

Channel isolation is controlled by the Display node. The **display.channelFlood** property controls which channel (or luminance) is displayed. Channel isolation occurs when the internal image is rendered to the display.

13.8.3 7.8.3 Out-of-range Display

The Out-of-Range color transform operates per channel. If channel data is 0.0 or less, the channel value is clamped to 0.0. If the channel data is greater than 1.0, the channel data is clamped to 1.0. If the channel data is in the range [0.0, 1.0] the data is clamped to 0.5.

The idea behind the transform is to display colors that are potentially problematic. If the pixels are grey, then they are ``safe” in the [0.0, 1.0] range. If they are brightly colored or dark, they are out-of-range.

0.0 is usually considered an out-of-range color for computer graphics applications when the image is intended for final output.

Some compositing programs have trouble dealing with negative values.

Note that HDR images will definitely display non-grey (bright) pixels when the out-of-range transform is applied. However, they probably should not display dark pixels.

You can turn on Out-of-range Display from the View menu.

CHAPTER

FOURTEEN

CHAPTER 8 - USING LUTS IN OPEN RV

Look up tables (LUTs) are useful for approximating complicated color transforms, especially those which have no known precise mathematical representation. RV provides four points in its color pipeline where LUTs can be applied: just after reading the file and before caching (pre-cache LUT *8.3*) directly after the cache (file LUT), just before display transforms (look LUT), and as one of the display transforms (display LUT). The first three are per-source while there is only a single display LUT for each RV session. See *7.1*

Each of the LUTs can be either a channel LUT or a 3D LUT (the difference is explained below *8.1* . In the case of a 3D LUT there can also be an additional channel pre-LUT which can be used to shape the data. Both types of LUT are preceded by an input matrix which can scale high dynamic range data into the range of the LUT input (which is the range [0,1]). The values the LUT produces can be outside of the [0,1] range. This makes it possible for any of the LUTs to transform colors outside of the typical [0,1] range on both input and output.

Internally, RV will store the LUT as either half precision floating point or 16 bit integral. Not all hardware is capable of processing LUTs stored as floating point (esp. the 3D LUTs) so if you notice banding or noisy output when using floating point LUT storage, you may have better luck with the 16 bit integral representation. If RV can determine whether the floating point LUTs are usable itself it will default to whatever is appropriate.

When applied in hardware, the LUTs are interpolated when a value is not exactly represented in the LUT. This is usually more of an issue with 3D LUTs than channel LUTs since they have fewer samples per dimension. When interpolating between sample values, RV uses linear interpolation for channel LUTs and tri-linear interpolation for 3D LUTs.

There are a number of ways to create a LUT. For film look simulation, it's often necessary to have special hardware to measure and compare film recorder output. Alternately, you use a lightbox; and assuming you have a well calibrated neutral monitor, ``eyeball" the LUT by comparing the film to the monitor.

RV has two different algorithms for applying the LUTs on the GPU: using floating point or fixed-point integer textures. Not all cards are equally capable with 3D LUTs and floating point. If RV detects that the card probably can't do a good job with the floating point hardware it will switch to a fixed-point representation using 16 bit integer LUTs. Sometimes even though the driver reports that the LUTs can be floating point, you will see banding in the final images. If that occurs, try forcing the use fixed-point LUTs by turning off the Floating Point 3D LUTs item on the Rendering tab of the preferences. The fixed point LUT algorithm will perform just as well as floating point in 99% of normal use cases.

To assign a LUT file to the source or Display:

- The Import menu under the File menu is used to assign LUT files to either the source's Look or File pipelines. Additionally, LUT files can be assigned to the Pre-Cache and Display.

14.1 8.1 Channel (1D) versus 3D LUTs

A channel LUT (also called a 1D LUT) has three independent look-up tables: one each for the R G and B channels. The alpha channel is not affected by the channel LUT. Channel LUTs may be very high resolution with up to 4096 samples. Each entry in the channel LUT maps an input channel value to an output channel value. The input values are in the [0,1] range, but the output values are unbounded.

Channel LUTs differ from 3D LUTs in one critical way: they can only modify channel values independently of one another. In other words, e.g., the output value of the red channel can only be a result of the incoming red value. In a 3D LUT, this is not the case: the output value of the red channel can be dependent on any or all of the input red, green, and blue values. This is sometimes called channel cross-talk.

The other important difference between channel and 3D LUTs is the number of samples. Channel LUTs are one dimensional and therefor consume much less memory than 3D LUTs. Because of this, channel LUTs can have more samples per-channel than 3D LUTs.

The implication of all this is that channel LUTs are useful for representing functions like gamma or log to linear which don't involve cross-talk between channels whereas 3D LUTs are good for representing more general color transforms and simulating physical output like film.

3D LUTs can be very memory intensive. A 64 × 64 × 64 LUT requires 64 3 × 4 bytes of data (3Mb). You can quickly run out of memory for your image on the graphics card by making the 3D LUT too big (e.g. 128 × 128 × 128 , this will slow RV down). RV does not require the 3D LUT to have the same resolution in each dimension. You may find that a particular LUT is smooth or nearly linear in one or more dimensions. In that case you can use a lower resolution in those dimensions.

Some graphics cards have resolution issues with 3D textures which can cause loss of precision when RV's 3D LUT feature is enabled. On older NVidia cards and ATI cards in general, 3D textures may be limited to non-floating point color representations. Precision loss when using a LUT can be exacerbated by applying display gamma on these cards. To minimize precision loss on those types of cards, bake monitor gamma and/log-lin conversion directly into the display LUT. With newer GPUs this is not as much of an issue.

14.2 8.2 Input Matrix and Pre-LUT

For HDR applications, the incoming data needs to be rescaled and possibly shaped. RV has two separate components which do this: the LUT input matrix and a channel pre-LUT. The input matrix is a general 4 × 4 matrix. For HDR pixels, the matrix is used to scale the incoming pixel to range [0 , 1] . The pre-LUT, the channel LUT, and the 3D LUT all take inputs in that range. Figure *8.1* shows a diagram of the channel and 3D LUT components and their input and output ranges.

The pre-LUT is identical to the channel LUT in implementation. It maps single channel values to new values. Unlike the general channel LUT, the pre-LUT must always map values in the [0 , 1] range into the same range. The purpose of the pre-LUT is to condition the data before it's transformed by the 3D LUT.

For example, it may make sense for 3D LUT input values to be in a non-linear space – like log space. If the incoming pixels are linear they need to be transformed to log before the 3D LUT is applied. By using a relatively high resolution pre-LUT the data can be transformed into that space without precision loss.

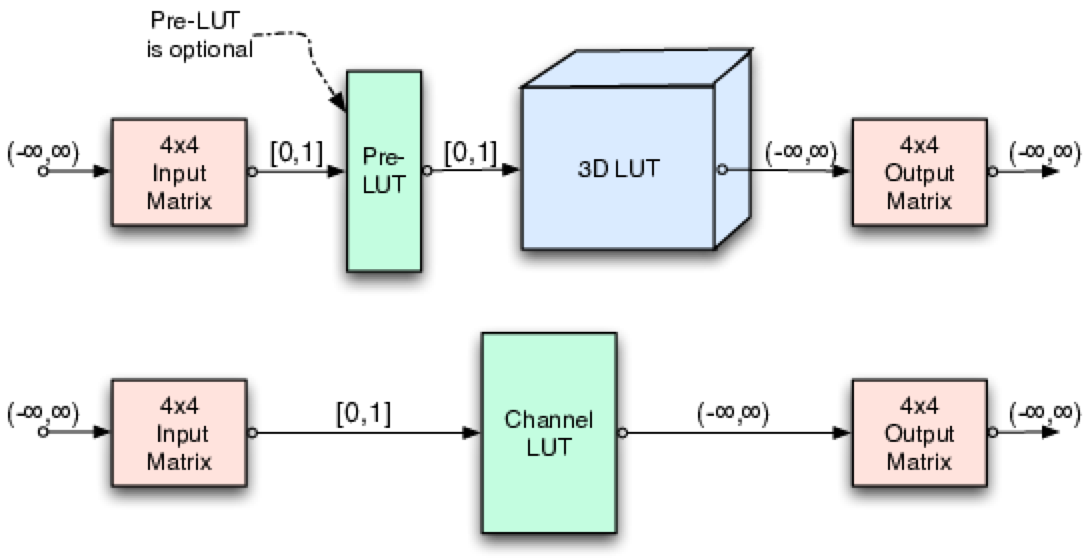


Figure 8.1: 3D and Channel LUT Components

14.3 8.3 The Pre-Cache LUT

The first LUT that the pixels can be transformed by is the pre-cache LUT. This LUT has the same parameters and features as the other LUTs, but it is applied before the cache. The pre-cache LUT is currently applied by the CPU (not on the GPU) whereas the file, look, and display LUTs are all used by the graphics hardware directly. For this reason the pre-cache LUT is slightly slower than the others.

The pre-cache LUT is useful when a special caching format is desired. For example by using the pre-cache LUT and the color bit depth formatting, you can have RV convert linear OpenEXR data into 8 bit integer format in log space. By using RV's log to linear conversion on the cached 8 bit data you can effectively store high dynamic range data (albeit limited range) and get double the number of frames into the cache. Many encoding schemes are possible by coupling a custom pre-cache LUT, change of bit depth, and the hardware file LUT to decode on the card.

14.4 8.4 LUT File Formats

Extension	Type	1D	3D	PreLUT	Float	Input	Output
csp	Rising Sun Research	•	•	•	•	[- ∞ , ∞]	[- ∞ , ∞]
rv3dlut	RV 3D		•		•	[0 , 1]	[- ∞ , ∞]
rvchlut	RV Channel	•			•	[0 , 1]	[- ∞ , ∞]
3dl	Lustre		•			[0 , 1]	[0 , 1]
cube	IRIDAS		•		•	[0 , 1]	[- ∞ , ∞]
any	Shake	•			•	[0 , 1]	[- ∞ , ∞]

Table 8.1: LUT Formats (as Supported in RV)

RV supports several of the common LUT file formats (See Table *8.1*). Unfortunately, not all LUT formats are equally capable and some of them are not terribly well defined. In most cases, you need to know the intended use of a particular LUT file. For example, it doesn't make sense to apply a LUT file which expects the incoming pixels to be in Kodak Log space to pixels from an EXR file (which is typically in a linear space). Often there is no way to tell the intended usage of a LUT file other than its file name or possibly comments in the file itself. Most formats do not have a public mechanism to indicate the usage to an application.

To complicate matters, many LUT files are intended to map directly from the pixels in a particular file format directly to your monitor. When using these types of LUTs in RV you should be aware than making any changes to the color using RV's color corrections or display corrections will not produce expected results (because you are operation on pixels in the color space appropriate for the display, rather than in linear space).

One of the more common types of LUT files you are likely to come across is one which maps Kodak Log space to sRGB display space. The file name of that kind of LUT might be log2sRGB or something similar. A variation on that same type of LUT might include an additional component that simulates the look of the pixels when projected from a particular type of film stock. Strictly speaking, you do not need to use log to sRGB LUTs with RV because it implements these functions itself (and they are exact, not approximated). So ideally, if you require film output simulation you have a LUT which only does that one transform. Of course this is often not the case; the world of LUT formats is a complicated one.

14.4.1 8.4.1 RSR .csp LUT Format

Currently, the best LUT format for use with RV is the .csp format. This format handles high dynamic range input and output as well as non-linear and linear pre-LUTs. It maps most closely to RV's internal LUT functions.

There is one type of .csp file which RV does not handle: a channel LUT with a non-linear pre-LUT. This is probably a very rare beast since an equivalent 1D LUT can be created with a linear pre-LUT. An error will occur if you attempt to use a channel LUT with a non-linear pre-LUT.

When RV reads a pre-LUT from this file format and it can determine that the pre-LUT is linear, it will convert the pre-LUT into a matrix and apply it as the LUT input matrix. In that case the non-linear channel pre-LUT is not needed. If the pre-LUT is non-linear (in any channel) RV will construct a channel LUT which is used just before the 3D LUT. Input values in the .csp pre-LUT are normalized and the scaling is then moved to the input matrix. Using a matrix when possible frees up resources for other LUTs and images in the GPU. Any pre-LUT in a .csp file with only two values is by definition a linear pre-LUT.

```
CSPLUTV100
3D

2 ^\label{preLUTStart}^
0 13.5
0 1
2
0 13.5
0 1
2
0 13.5
0 1 ^\label{preLUTEnd}^

...
```

In the above listing, lines `preLUTStart` to `preLUTEnd` are linear pre-LUT values. In this case the pre-LUT values are mapping values int the range [0,13.5] down to [0,1] for processing by a 3D LUT (which is not shown). For a summary of the RSR .csp format see appendix *G*.

For the most part, it's not necessary to know the distinction between a linear and non-linear pre-LUT in the file. However, the behavior of the pre-LUT outside the bounds of its largest and smallest input values will be different for linear pre-LUTs. Since the pre-LUT is represented as a matrix, it will not clamp values outside the specified range. Non-linear pre-LUTs will clamp values.

CHAPTER

FIFTEEN

CHAPTER 9 - USING CDLS IN OPEN RV

As discussed previously there are two default places to set ASC CDL properties in the RV node graph. One is as a file CDL on the RVLinearize node in the RVLinearizePipepline and the other is as a look CDL on the RVColor node in the RVLookPipeline. In the case of the RVLinearize node the CDL is applied before linearization occurs whereas in the case of the RVColor node the CDL s applied after linearization and linear color changes.

To assign a CDL file to the source:

- The Import menu under the File menu is used to assign CDL files to either the source's Look or File pipelines.

15.1 9.1 CDL File Formats

The types of files RV supports right now are Color Decision List (.cdl), Color Correction (.cc) and Color Correction Collection (.ccc) files. Color Correction Collection files can include multiple Color Corrections tagged by ids. We do not support reading the properties by id. Therefore the first Color Collection found in the file will be read and used.

CHAPTER

SIXTEEN

CHAPTER 10 - PACKAGES

There are multiple ways to customize and extend RV, and almost all these ways can be encapsulated in an RV Package. A package is a zip file with source or binary code that extends RV's feature set. Packages are constructed so that they can be automatically installed and removed. This makes it much easier for you to maintain RV than if you modify RV source directly. (Note: although packages are zip files, they are labeled with a .rvpkg extension to prevent mail programs, etc, from automatically unzipping them.)

Packages can encapsulate a wide variety of features: everything from setting the titile of the window automatically to implementing a paint annotation package. Some of RV's internal code is also in package form prior to shipping: e.g. the entire remote sync feature is initially constructed as a package of Mu source code (but is permanently installed when shipped).

The package manager can be found in the preferences dialog. There are two sections to the user interface: the list of packages available and the description of the selected package. Some packages may be "hidden" by default; to see those packages toggle the show hidden packages button. To add and remove packages use the add and remove buttons located at the bottom left.

The reference manual contains detailed information about how to create a package.

16.1 10.1 Package Support Path

When a package is added, a list of permissible directories in the support path is presented. At that time you can choose which support directory to add the package to. When RV starts, these directories are automatically added to various paths (like image I/O plugins).

By default, RV will include the application directory's plug-ins directory (which is probably not writable by the user) and one of the following which is usually writable by the user:

~/Library/Application Support/RV	macOS
~/.rv	Linux
$APPDATA/RV	Windows

You can override the default support path locations by setting the environment variable RV_SUPPORT_PATH. The variable contents should be the usual colon (Linux and Mac) or semicolon (Windows) separated list of directories. The user support directory is not included by default if you set the RV_SUPPORT_PATH variable, so be sure to explicitly include it if you want RV to include that path. Also note that the support path elements will have subdirectories called Mu, Packages, etc. In particular, when using the support path to install Packages, you want to include the directory above Packages in the path, not the Packages subdir itself.

The file system of each directory in the support path contains these directories:

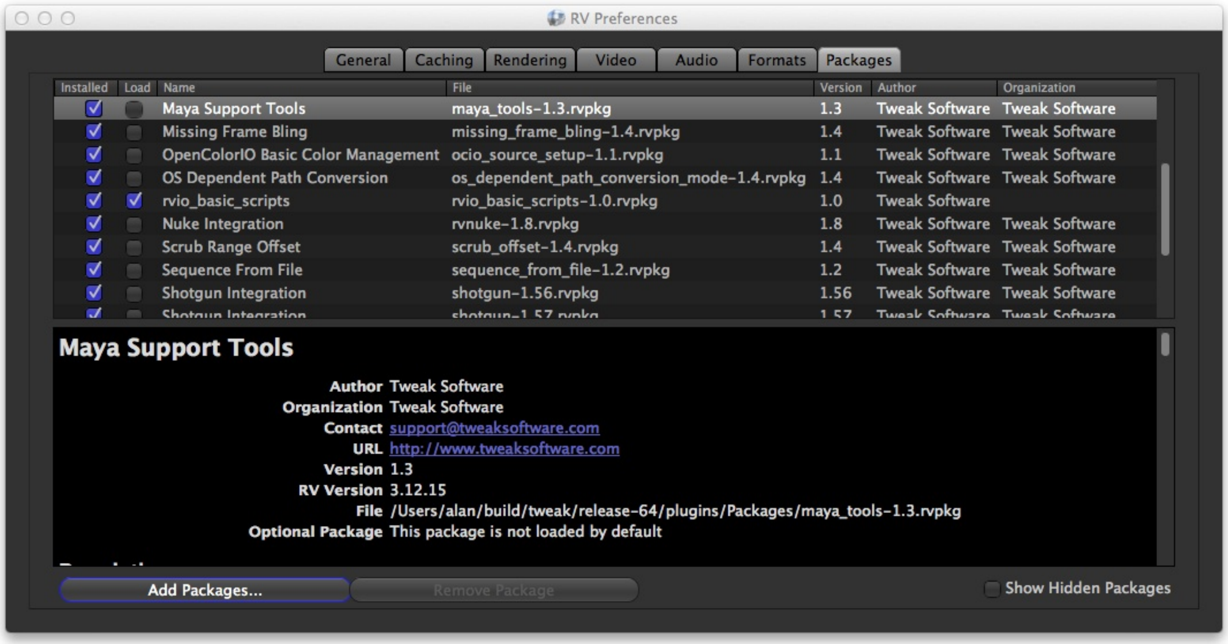
Packages/	Package zip files
ConfigFiles/	Area used by packages to store non-preference configuration information
ImageFormats/	Image format Plug-Ins
MovieFormats/	Movie format Plug-Ins
Mu/	Mu files implementing packages
Output/	Output Plug-Ins (Audio/Video)
MediaLibrary/	Media Library Plug-Ins
Python/	Python files implementing packages
SupportFiles/	Additional files used by packages (icons, etc)
lib/	Shared libraries required by packages

RV will create this structure if it's not already there the first time you add or install a package.

16.2 10.2 Installation

To add a package:

1. Open the Package Manager.
2. In the Packages tab, click **Add Packages…**
3. Navigate to the package's .rvpkg file.



Installed	Load	Name	File	Version	Author	Organization
✓		Maya Support Tools	maya_tools-1.3.rvpkg	1.3	Tweak Software	Tweak Software
✓		Missing Frame Bling	missing_frame_bling-1.4.rvpkg	1.4	Tweak Software	Tweak Software
✓		OpenColorIO Basic Color Management	ocio_source_setup-1.1.rvpkg	1.1	Tweak Software	Tweak Software
✓		OS Dependent Path Conversion	os_dependent_path_conversion_mode-1.4.rvpkg	1.4	Tweak Software	Tweak Software
✓	✓	rvio_basic_scripts	rvio_basic_scripts-1.0.rvpkg	1.0	Tweak Software	
✓		Nuke Integration	rvnuke-1.8.rvpkg	1.8	Tweak Software	Tweak Software
✓		Scrub Range Offset	scrub_offset-1.4.rvpkg	1.4	Tweak Software	Tweak Software
✓		Sequence From File	sequence_from_file-1.2.rvpkg	1.2	Tweak Software	Tweak Software
✓		Shotgun Integration	shotgun-1.56.rvpkg	1.56	Tweak Software	Tweak Software
✓		Shotgun Integration	shotgun-1.57.rvpkg	1.57	Tweak Software	Tweak Software

Maya Support Tools

Author	Tweak Software
Organization	Tweak Software
Contact	support@tweaksoftware.com
URL	http://www.tweaksoftware.com
Version	1.3
RV Version	3.12.15
File	/Users/alan/build/tweak/release-64/plugins/Packages/maya_tools-1.3.rvpkg
Optional Package	This package is not loaded by default

Figure 10.1: Package Manager

Tip: When you are troubleshooting packages you have installed, enable the **Show Hidden Packages** checkbox.

Once a package has been added, to install or uninstall simply click on the check box next to the name. The package is installed in the same support directory in which it was added.

A package can be added, removed, installed, and uninstalled for all users or by a single user. Usually administrator privileges are necessary to operate on packages system wide. When a package is added (the rvpkg file) it is copied into a known location in the support path.

It's best to avoid editing files in these locations because RV tries to manage them itself. When a package is installed the contents will be installed in directories of the support directory.

When first installed, packages are loaded by default. To prevent a package from loading uncheck the load check box. This is useful for installed packages which are not uninstallable because of permissions. While the install status of a package is universal to everyone that can see it, the load status is per-user.

Note : A restart of RV is before a change in a package Installed or Loaded state takes effect.

16.3 10.3 Package Dependencies

Packages may be dependent on other packages. If you select a package to be installed but it requires that other packages be installed as well, RV will ask you if it can install them immediately. A similar situation can occur when setting the load flag for a package. When uninstalling/unloading the opposite can happen: a package may be required by another that is "using" it. In that case RV will ask to uninstall/unload the dependent packages as well.

Some packages may require a minimum version of RV. If a package requires a newer version, RV shouldn't allow you to install it.

In some cases, manual editing of the support directory may lead to a partially installed or uninstalled package. The package manager has a limited ability to recover from that situation and will ask for guidance.

CHAPTER

SEVENTEEN

CHAPTER 13 - NETWORKING

RV implements a simple chat-like network protocol over TCP/IP. Network connections can be between two RVs or between a custom program and RV. The protocol is documented in the reference manual. This chapter discusses how to make network connections, how network permissions work, and how to synchronize two RVs using networking.

Note that just turning on networking and establishing a connection doesn't actually do anything in RV. But this step is necessary before any of the features that use the networking can start.

Some of the things you can do with networking include synchronizing to RV sessions across a network, sending pixels from another program to RV (e.g., a RenderMan display driver), or controlling RV from another program (remote control). Sync is part of RV, but remote control and sending pixels must be implemented in another program. RV comes with source code to a program called rvshell which shows you how to write software that connects with RV and controls playback remotely as well as sending images to RV over the connection. Some other applications are using special devices to control RV (like apple remote control for example) or controlling RV as an embedded playback component in a custom application on Linux.

The reference manual has documentation about the networking protocol and how to create programs that talk to RV. The inner workings of the rvshell program are also discussed there.

While it's possible to create connections across the internet, it's a little tricky, see *13.1.4* .

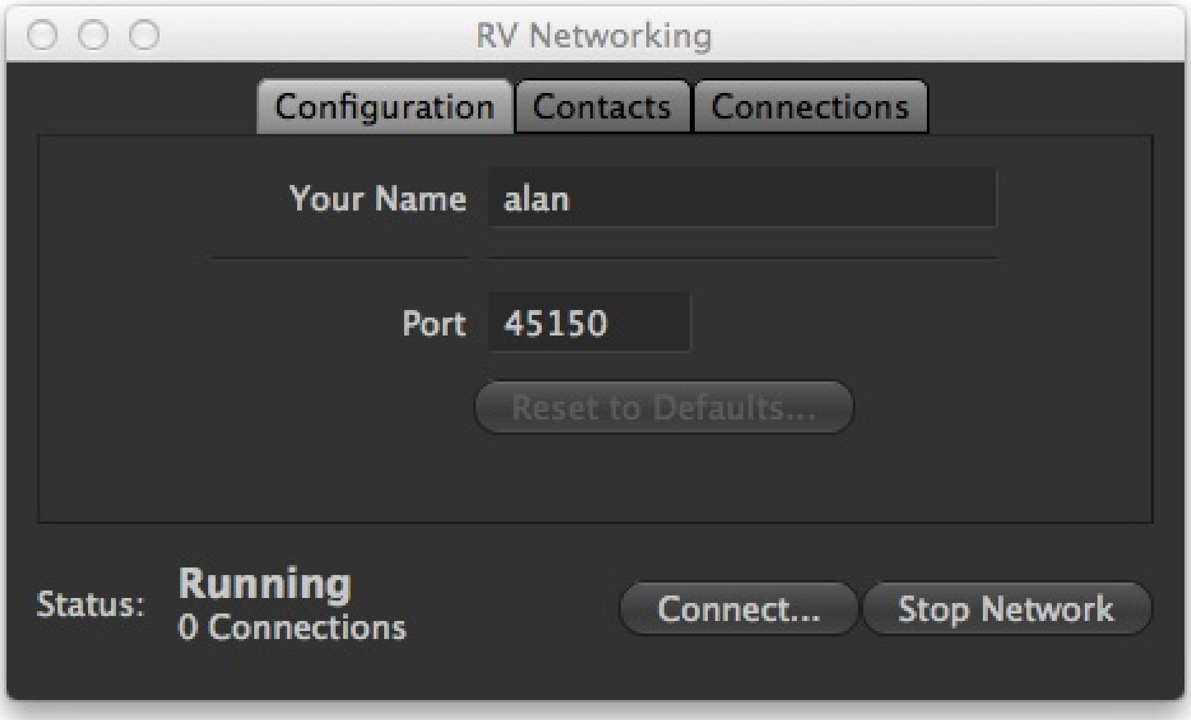
Warning: Care should be taken with RV networking. Once a connection is established to a client application (RV or any other client) all scripting commands (including "system(…)") are allowed and executed with the user id and permissions of the local RV user. Local connections (from a client application running on the same machine as RV), are not authenticated and the user is not required to confirm connection. Remote connections must be explicitly allowed by the user, but the identity of the remote user can not be confirmed. Also note that traffic over RV network connections is generally not encrypted.

Use of RV Remote Sync is at your own risk. Although we may recommend preferred configurations, we cannot guarantee that any configuration will be secure or that your use of RV Remote Sync will not introduce vulnerabilities into your systems. If you do not wish to accept this risk, please do not use RV Remote Sync.

17.1 13.1 The Network Dialog

Before you can use any of RV's networking features, you have to tell RV to begin listening for connections. There are two ways to do this: from the command line using the -network flag or interactively using the the Network dialog via the RV → Networking… menu item.

The network dialog box has three pages: Configuration for setting up your identity and the port on which RV will communicate, the Contacts page for managing permissions, and the Connections page which shows a list of the currently active connections. At the bottom of the dialog are buttons for starting and stopping the networking and for initiating a new connection.



Your Name alan

Port 45150

Status: Running
0 Connections

Figure 13.1: RV Network Dialog

17.1.1 13.1.1 Configuration

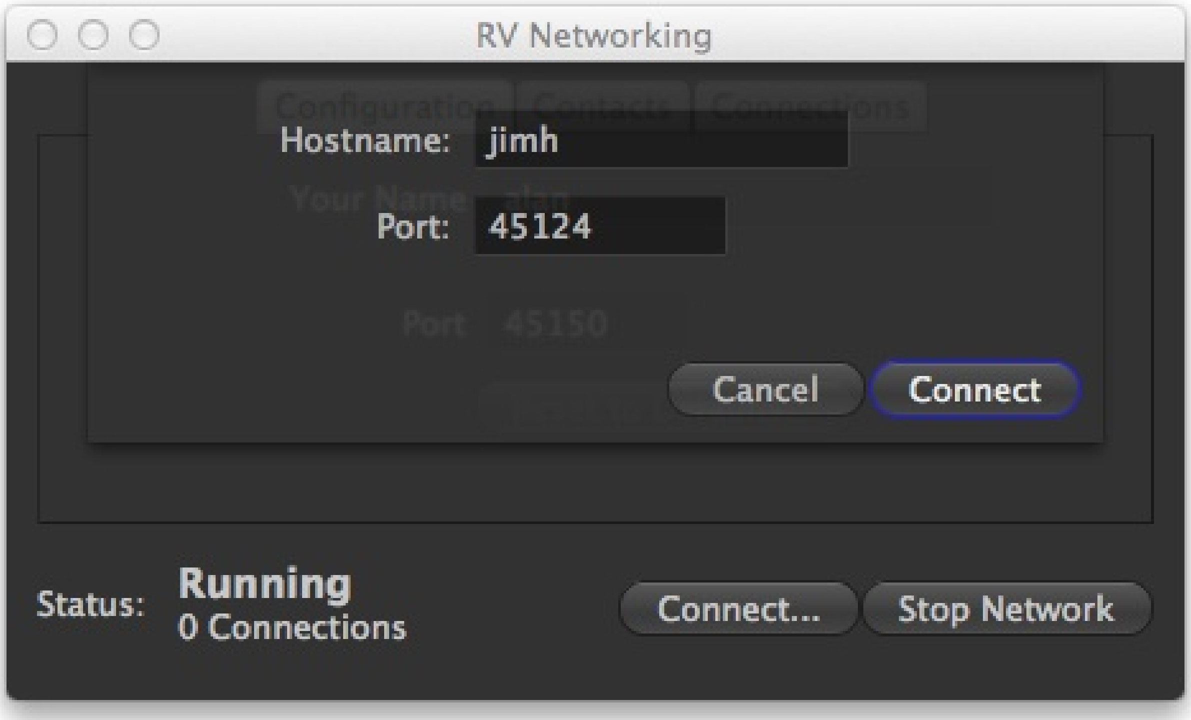
The configuration requires two pieces of information: your identity (which appears at the other end of the connection) and a port number. Unless your systems administrator requires RV to use a different port you should leave the default value for the port number. Networking must be stopped in order to change the configuration.

17.1.2 13.1.2 Starting a Connection

To start networking and have RV accept connection just hit the Start Network button. Once running, the status indicator in the bottom left-hand corner of the dialog will show you the network status and the number of active connections (which starts out as 0).

There are two ways to initiate a new connection: by pressing the Connect… button at the bottom of the dialog or by selecting a known address on the Contacts page.

Using the Connect… button will show another dialog asking for the host name of the machine to connect to. Using this method, RV will not care which user it finds at the other end of the connection. If RV does not yet have record of the user it finds it will create one.



Hostname: jimh
Port: 45124
Status: Running
0 Connections

Figure 13.2: Network Dialog Starting a Connection

17.1.3 13.1.3 Contacts and Permissions

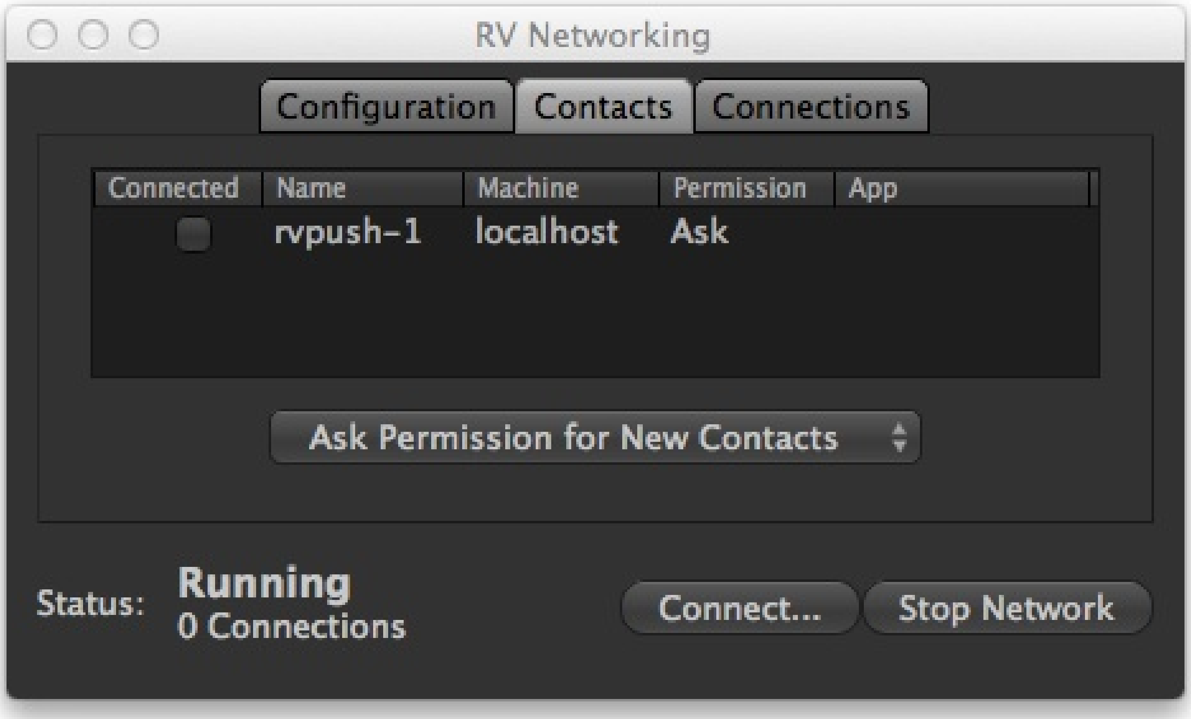
If networking is on, and a connection is initiated by another party, RV will ask whether or not you want the connection to occur. At that point you have three choices: accept the connection but ask for permission next time, always accept connection from the user/program that's asking, or deny the connection.

Each new contact that RV receives (whether or not you accept) is recorded in the Contacts page. On this page you delete existing contacts, change permissions for contacts, and initiate connections to contacts. You can also specify the behavior RV when new contacts try to connect to your RV. This is most useful when RV is used by multiple people (for example in a common space like a view station). Often instances of RV running in common spaces should be more conservative about allowing connections without asking.

To change an existing contacts permissions, double click on the current value (where it says, Allow, Ask, or Reject). You should see a pop-up menu which lets you change the value.

To initiate a connection to an existing contact, double click on the contact name or machine name.

There is also a pop-up menu which lets you delete an existing contact or initiate a connection.



Connected	Name	Machine	Permission	App
☐	rvpush-1	localhost	Ask	

Status: Running
0 Connections

Figure 13.3: Network Contacts Page

17.1.4 13.1.4 Networking Across the Internet

It's possible to connect to a remote RV across the internet in a peer-to-peer fashion, but special care needs to be taken. We recommend one of two methods: using ssh tunneling or using a VPN.

SSH tunnelling is well known in the IT community. Since there is no standard firewall configuration, you will need to understand both the way in which ssh tunnelling is set up and the topology of the firewalls on both ends of the connection. No third party connection is needed to sync across ssh tunnels and the connection is relatively secure.

Using a VPN is no different than a local area network once it has been established: you need to know the IP address or name of the host with which you want to connect. If you do not have a VPN in common with the remote participant, you can create one using Hamachi . The service is free for non-commercial use. See their website for more details.

17.2 13.2 Synchronizing Multiple RVs

Once a connection has been established between two RVs, you can synchronize them.

To start sync select the menu item Tools → Sync With Connected RVs. You should see ``Sync ON" in the feedback area on both RVs and a Sync menu will appear in the menu bar.

Usually it's a good idea to have all participants looking at the similar media, but it's not enforced. In particular, note that RV's auto-conforming features mean that one party can be looking at a high-res OpenEXR sequence and another at a quicktime movie of the same sequence, and the sync can still be quite useful.

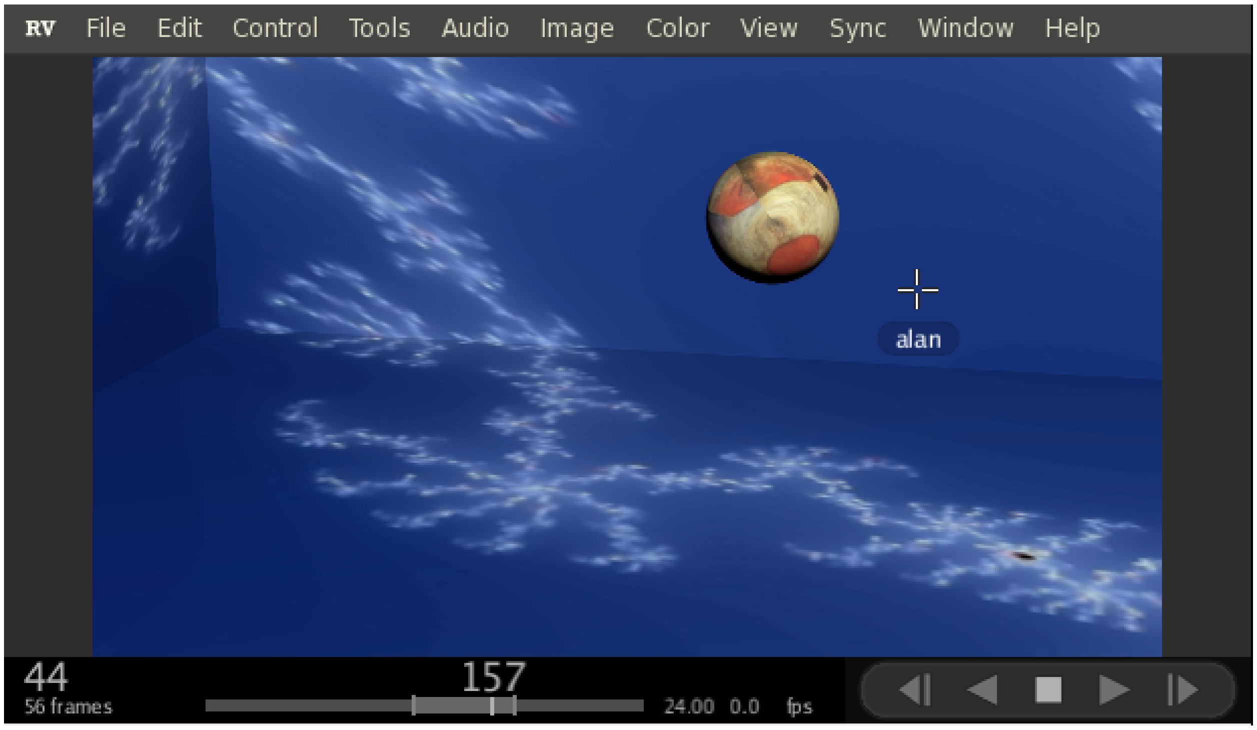


Figure 13.4: Sync Mode Showing Remote User's Cursor

17.2.1 13.2.1 Using Sync

You can control which aspects of RV are transmitted to and received from remote RV's from the Sync menu. The menu is divided into to two main sections: things you can send and things you have agreed to receive. By default, sync mode will accept anything it gets, but will only send certain types of operations. So if one of the participants decides to send more than the default state the remote RVs will automatically use it.

Sync mode will always send frame changes and playback options like the current fps, and the playback mode.

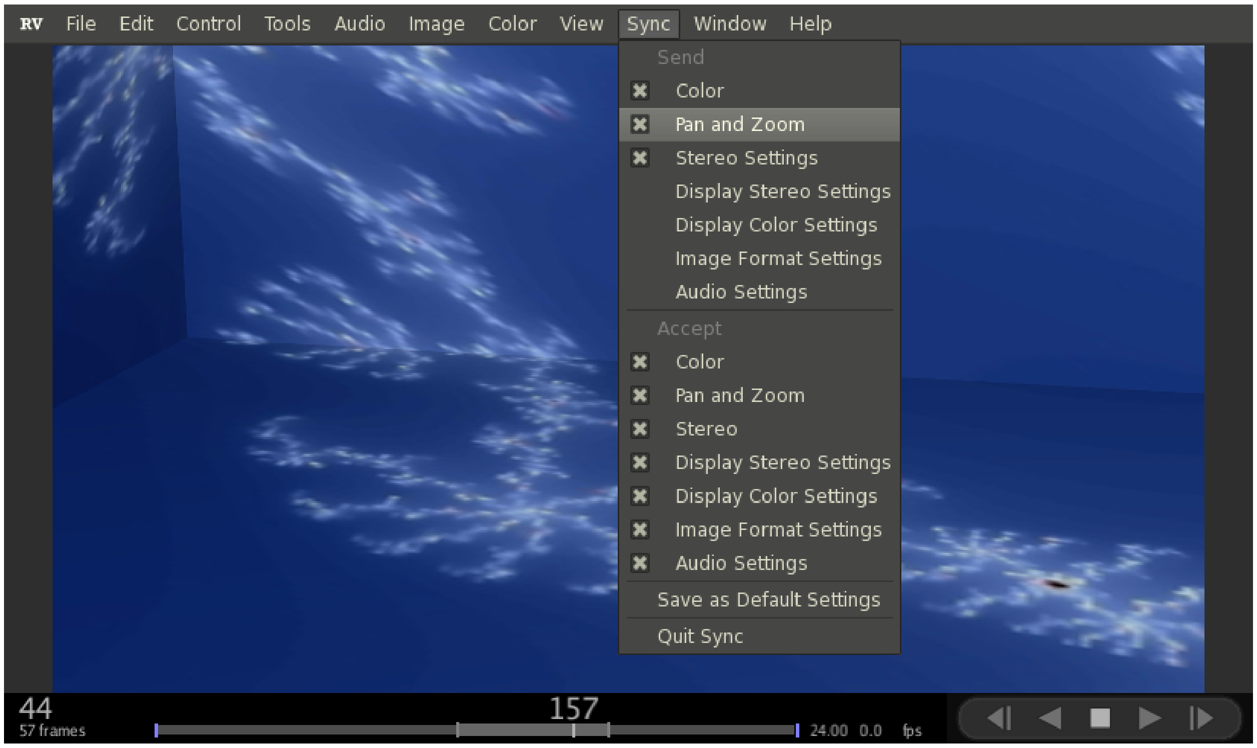


Figure 13.5: Sync Mode Menu

Event Group	What Gets Sync'ed
Color	Any per-source color changes. E.g., exposure, gamma, hue, saturation and contrast. The File LUT is currently not synced between RVs.
Pan and Zoom	Any viewing pan and zoom changes. The scales are relative to the window geometry (which is not synced).
Stereo Settings	Settings that are per-source (e.g., those that are found under the Image → Stereo menu). Stereo settings from the under View → Stereo menu are not included.
Display Stereo Settings	Stereo settings that affect the entire session (e.g. those found under the View → Stereo menu). This includes the stereo display mode. Be careful when using this when any participant is not capable of hardware stereo viewing.
Display Color Settings	Overall brightness, red, green, blue, channel isolation, display gamma, display sRGB and Rec709 modes. Most of the items found under the View menu. The look and display LUTs are currently not synced.
Image Format Settings	Image resolution, color resolution, rotation, alpha type, pixel aspect ratio changes.
Audio Settings	Soundtrack and per-source audio volume, balance, cross-over, etc. Things found under the Audio menu.
Default	Frame changes, FPS, realtime and play all frames playback mode settings, in and out points. These are always sync'ed.

Table 13.1: Sync Mode Sending and Receiving Events

17.3 13.3 “Streaming” Movie Media from Online Sources

Media “paths” provided to RV may be URLs that link to online media. These may be provided via the command line, in RVSession Files, or through scripting. This functionality should be considered “Beta” and there are several caveats you should be aware of:

- The “media from online” workflow is a worst-case for RV’s caching system, which generally assumes that the media can be accessed at arbitrary locations. In general, it’s probably best to use Region Caching, with a large cache size, and allow the media to cache before playback begins. For media that is low-bandwidth, or for particularly low-latency connections, it may work well to use a large Lookahead Cache.
- The URL needs to end in “.mov” or “.mp4” or some other extension that the MIO_FFMPEG plugin will recognize.
- Audio may be a problem, since it is generally decoded and cached by a separate thread in RV. It may be best to turn on the “Cache All Audio” preference.
- The movie should be created with the “faststart” property (for RVIO this means adding the “-outparams of:movflags=faststart” command-line options).

CHAPTER

EIGHTEEN

CHAPTER 11 - OPENCOLORIO

OpenColorIO (OCIO) is a software library which provides cross application color consistency.

Note: Open RV supports OCIO v2, but is limited to the legacy API of OCIO v1.

You can use OCIO in RV's display, look, viewing, linearize and color pipelines. In each case OCIO can be used to convert from an incoming and outgoing color space with a user defined OCIO context. OCIO requires some work to set up and should be considered an advanced feature. Large facilities may find OCIO particularly useful in RV when used in conjunction with Nuke, Mari, or other products which support it.

You can learn about OpenColorIO at the OpenColorIO website .

In order to use OCIO with RV a source setup package needs to be created along with OCIO configuration files, LUTs, and an appropriate user environment. RV comes with a package called ocio_source_setup which implements a default policy for using OCIO. We recommend the package be copied and modified to suit each facility that uses OCIO. When the OCIO source setup package is enabled, parts or all of RV's existing color pipelines may be replaced with OCIO equivalents.

Note: The sample OCIO source setup package **supplements** the default source_setup and does not replace it; there is no need to turn off the default source_setup. We highly recommend copying and customizing the sample OCIO package for real world use.

There are four OCIO node types available in RV: OCIOLook, OCIOFile, OCIODispay, and the generic OCIO node. The default ocio_source_setup will use these nodes to replace RV's existing color, look, and display pipelines.

RV uses the OCIO GPU path instead of the more common CPU path for color computation. There are slight differences between the two which you will need to be aware of. Specifically, the color allocation parameters in the config file can have a big effect on the quality of the OCIO LUTs generated by the library.

The reference manual contains more detailed information about how to configure RV's OCIO nodes.

18.1 11.1 The ocio_source_setup Package

OpenColorIO defines color spaces which are used to transform images for viewing or for storage in various file formats. It does not have any policy about when to apply these transforms.

The provided ocio_source_setup package uses the default Sony Picture Imageworks method of detecting color space names in the incoming file names. The package queries OCIO and if the file name matches, the package will use OCIO nodes in various places in RV's pipelines instead of the usual RV color processing. If the incoming file name does not match an OCIO color space, the package will allow the existing RV color management to be used.

One gotcha with this package is that once an OCIO managed file name has been detected, the package will use OCIO for the display correction as well. So mixed OCIO and RV managed imagery will always be viewed using OCIO's display color corrections.

The package provides a basic menu fashioned after the OCIO display example program which allows you to file, look, and display transforms as well as forcing unrecognized media to use OCIO.

In order to use the ocio_source_setup package you need to do the following:

- Find the "OpenColorIO Basic Color Management" package in the Packages tab of the preferences. Make sure the "load" button next to the package name is activated and restart RV.
- Set the OCIO environment variable to the path to your config file.
- Start RV and load an image with the color space in the name.
- Note the "OCIO" top-level menu that appears when you use RV this way. You can use this menu to chose a Linearizing transform, or Display transform, from those provided by your config.

The OpenColorIO mailing list and website is a good place to get help about config files, documentation, and general operation.

18.1.1 11.2 Testing the Sample OCIO Source Setup

1. Find the "OpenColorIO Basic Color Management" Package in the Packages tab of the Preferences. Click "Load" button next to the package and exit.
2. Set the "OCIO" environment variable to the path to your favorite OCIO config file:

   ```
   setenv OCIO /OCIO/spi-vfx/config.ocio
   ```

3. Start RV and load an image with the color space in the name or otherwise (however is appropriate for your config).

   ```
   rv /media/images/ocio_special_names/marcie_clean_lg10.cin
   ```

4. Note the "OCIO" top-level menu that appears when you use RV this way. You can use this menu to chose a Linearizing transform, or Display transform, from those provided by your config.

18.1.2 11.3 Operation of the OCIO Node

The OCIO node in the OpenColorIO Basic Color Management package is configured to emulate three of the OCIO nuke node types: color, look, and display.

The OCIO nodes can be used in any of the RV pipelines as either a supplement to RV's existing color pipeline or in place of it. The pipelines are placed in RV's node graph in the places where all of the significant color processing occurs. By default these pipelines contain RV's color nodes (RVLinearize, RVColor, RVLookLUT, RVDisplayColor).

The default OCIO package will swap in OCIO equivalent nodes for the existing RV nodes. For example when using the SPI OCIO config with an "lg10" cineon file the OCIO package will remove RV's RVLinearize node in the linearization pipeline and replace it with a OCIOFile node set to take the lg10 input by default and output scene referred linear. The default OCIO package will also swap out the RVDisplayColor node (which does display color correction) for an OCIODisplay node. The OCIODisplay node is set to take scene referred linear and output to the default viewing transform (by default).

There are four OCIO node types: OCIONode, OCIOFile, OCIODisplay, and OCIOLook. All of them have the same properties; they only differ in their default configuration. Each of them can be used in any context if desired. The different node type names are primarily to make it easy to identify the nodes from the user interface.

The generic OCIONode can used as a top level (user) node. As with the RVColor node the OCIONode can therefor be used as a secondary color correction.

Note: You can use both RV and OCIO nodes in any of the pipelines. However, the example package does not do this: the intention is to show how OCIO can be used (at least for some media) *in place of* RV's color pipeline.

Table 1. OCIO Node Properties

string ocio.function	One of “color”, “look”, or “display”
float ocio.lut	Used internally to store the OCIO generated 3D LUT
int ocio.active	Activates/deactivates the OCIO node
int ocio.lut3DSize	The desired size of the OCIO generated 3D LUT (default=32)
string ocio.inColorSpace	The OCIO name of the input color space
string ocio_color.outColorSpace	The OCIO name of the output color space when ocio.function == “color”
string ocio_look.look	The OCIO command string for the look when ocio.function == “look”
int ocio_look.direction	0=forward, 1=inverse
string ocio_display.display	OCIO display name when ocio.function == “display”
string ocio_display.view	OCIO view name when ocio.function == “display”
component ocio_context	String properties in this component become OCIO config name/value pairs

18.1.3 11.4 The ocio_context Component

You can add properties to the OCIO node in RV to create an OCIO context. Any string property in the component called `ocio_context` will become a name/value pair for the OCIO context.

CHAPTER

NINETEEN

CHAPTER 12 - STEREO VIEWING

RV can display the first two image layers as stereo. There are several options:

- Anaglyph
- Side-by-Side
- Mirrored Side-by-Side
- DLP Checker
- Scanline Interleaved
- Left or Right eye only
- Hardware Left and Right Buffers

The stereo rendering is designed to work with the other features of RV. In most cases, color corrections, image geometry manipulations, and display corrections will work in conjunctions with stereo viewing.

When caching is on, all layers are cached (left and right eyes). You will need to reduce the image resolution to cache as many frames as non-stereo.

The left and right eye images are normalized (i.e. conformed to fit the RV window) so they may have different resolutions and/or bit depths. However, it is not advisable to have differing aspect ratios.

You can create stereo sources and set individual stereo parameters from the command line. See Section *3.2.2* to see how to do this.

19.1 12.1 Anaglyph or Luminance Anaglyph

The anaglyph modes display the left eye in the red channel and the right in eye in the green and blue channels (as cyan). If you were to wear colored red-cyan glasses and the eyes are correctly arranged, you should be able to see stereo with pseudo-color. Color anaglyph images work best for outdoor scenes (with lots of green) and in similar cases. They work very poorly with blue or green screen photography. For grey scale/non-color rendition of anaglyph, select the "Luminance Anaglyph" mode (Figure *12.4*)..

For color anaglyphs, if the color contrast is too great, the stereo effect will be difficult to see. If you turn on luminance viewing (hit the "l" key in the user interface), you can improve the effect (Figure *12.2*). A similar solution is to desaturate the image slightly (hit shift "S" and scrub); this will reduce the color contrast but keep some hints of color (Figure *12.3*).

Compression artifacts can seriously degrade stereo viewing especially in anaglyph mode. QuickTime movies, for example, with low quality compression may look fine when viewed one eye at a time, but in anaglyph mode JPEG or similar artifacts will be greatly amplified by slight color differences. The best way to view compressed material is with luminance display turned on (no color).

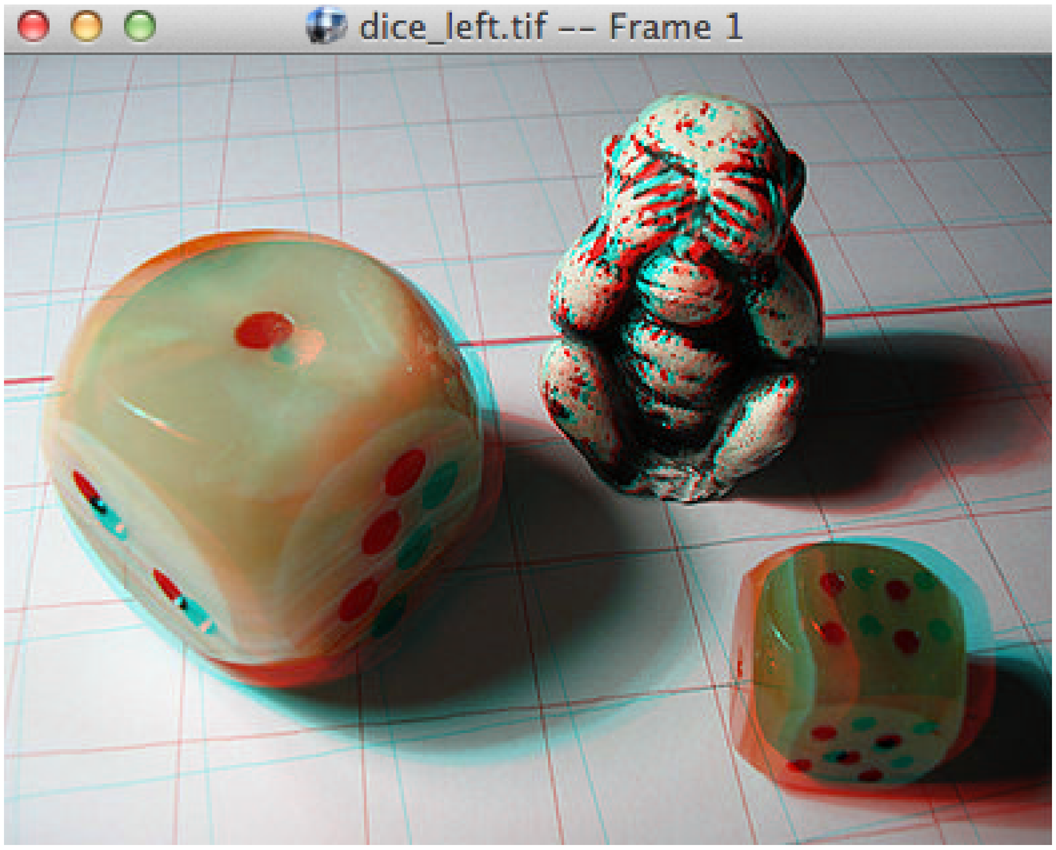


Figure 12.1: Anaglyph Stereo Display

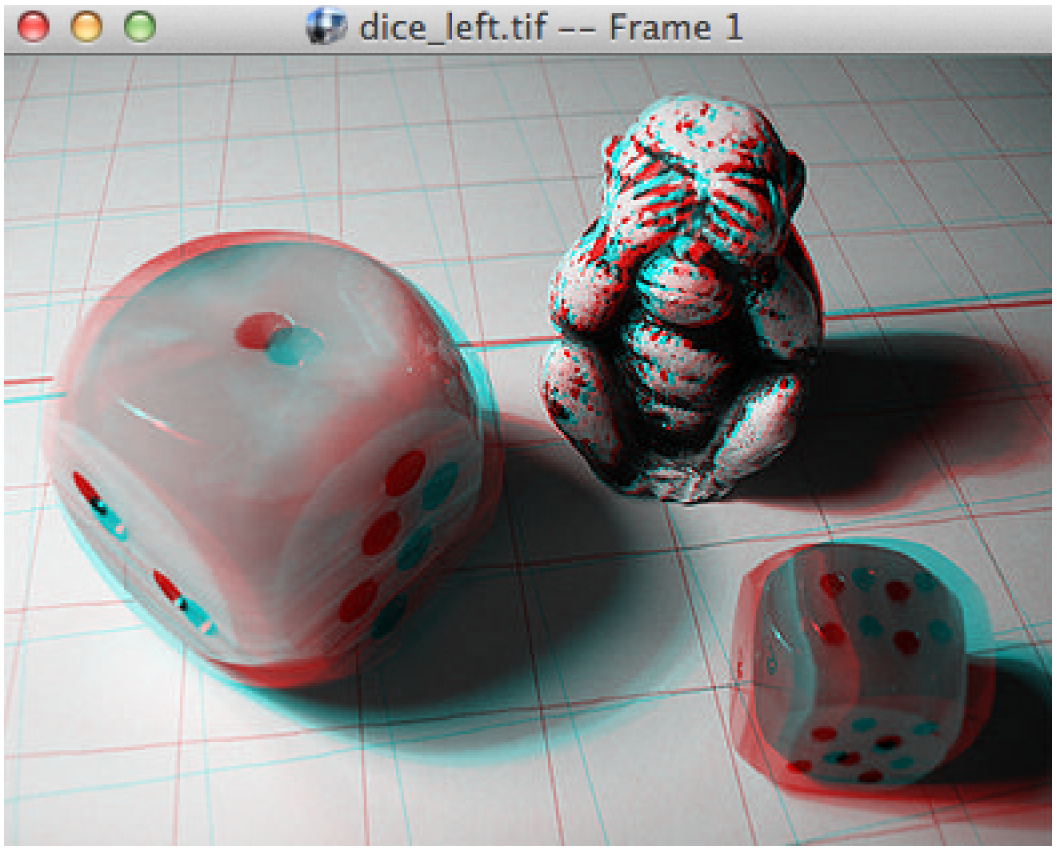


Figure 12.2: Anaglyph Stereo With Luminance Display

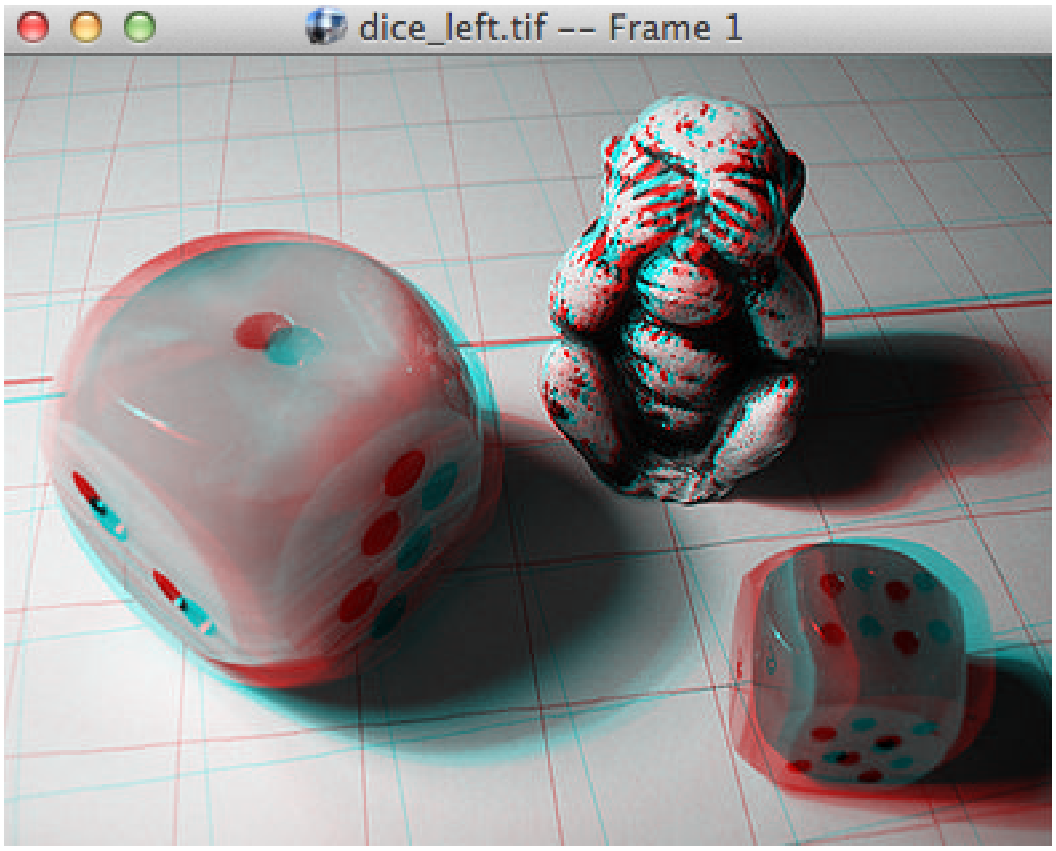


Figure 12.3: Anaglyph Display With Desaturation

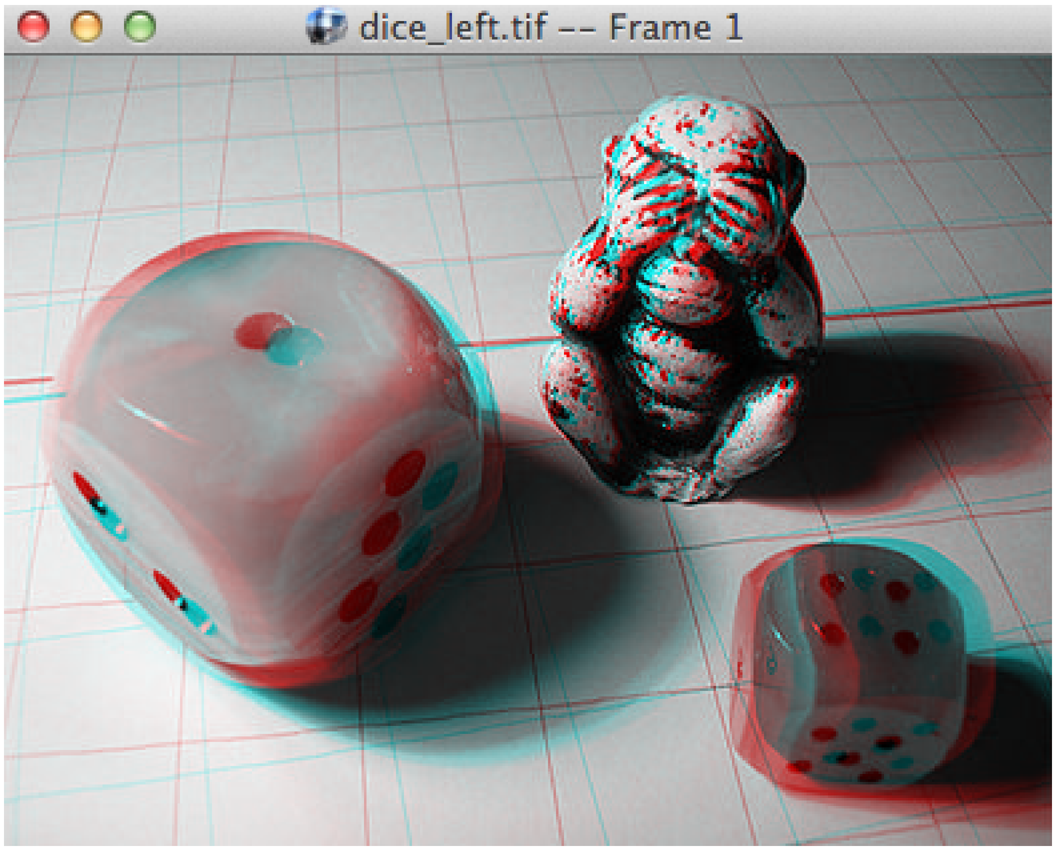


Figure 12.4: Luminance Anaglyph Display

19.2 12.2 Side-by-Side and Mirror

Side-by-Side mode displays the left and right eyes next to each other horizontally in full color. Some people are comfortable crossing their eyes to see stereo using this mode.

Mirror mode is similar, but the right eye is flopped. If you need the left eye flopped, turn on mirror mode and select Image->Flop or hit shift-"X" this will have the effect of mirroring the left eye instead. Note that the same effect can be produced by flopping the right eye only in mirror mode.

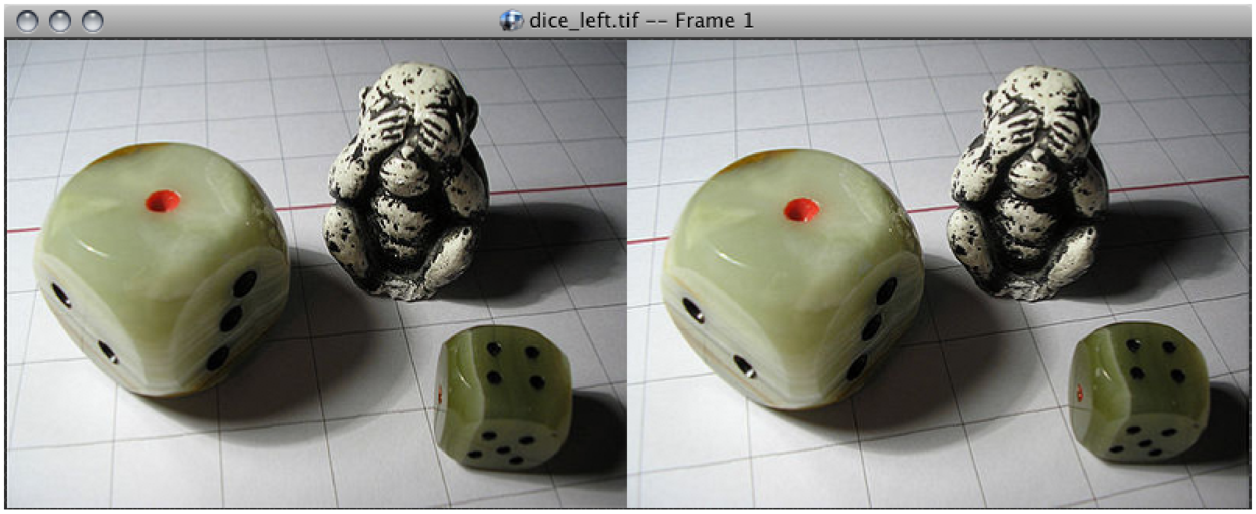


Figure 12.5:

Side-by-Side Stereo Display

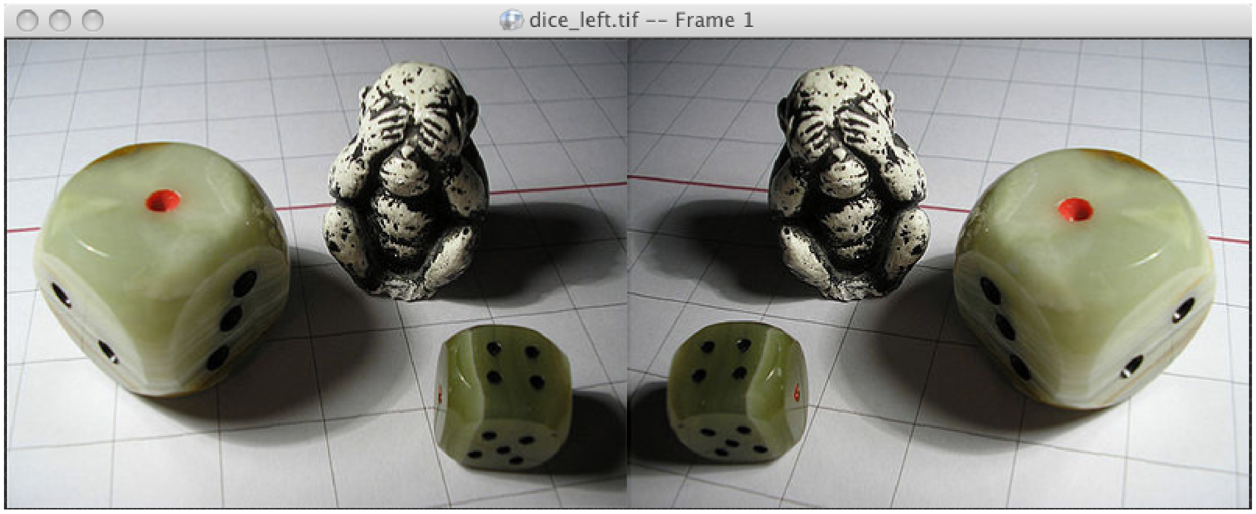


Figure 12.6:

Mirror Display Mode

19.3 12.3 DLP Checker and Scanline

These modes are designed to work with DLP projectors or LCD displays that directly support stereo display. In particular RV supports the SpectronIQ HD LCD display and DLP projectors using the Texas Instrument’s checkerboard 3D DLP input.

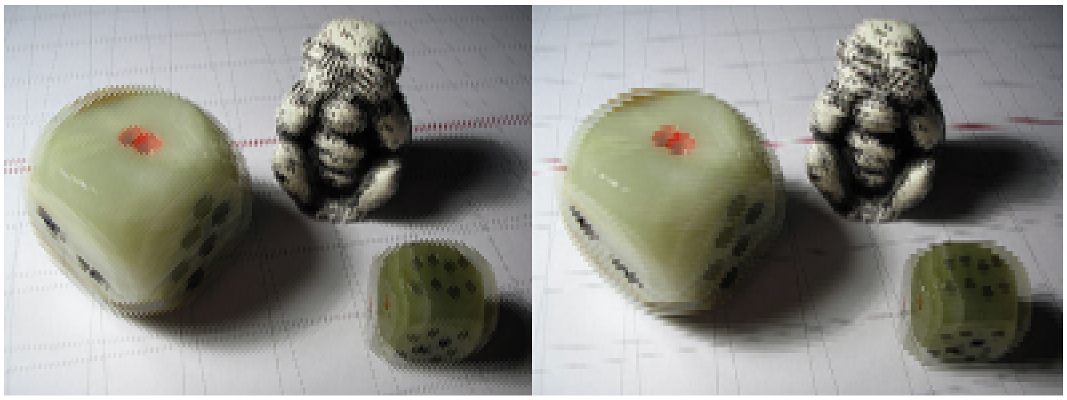


Figure 12.7:

DLP (left) and LCD Scanline (right) Stereo Display

19.4 12.4 HDMI 1.4a

HDMI 1.4a stereo modes like Side-by-Side and Top-and-Bottom are supported via RV’s Presentation Mode, described in Chapter *6* . To select one of these modes for your Presentation Device, you choose the appropriate Output Data Format in the Video preferences.

19.5 12.5 Hardware Stereo Support

RV can render into left and right buffers if your graphics card supports hardware stereo. You can tell if this is the case by seeing if the menu item View → Stereo → Hardware can be selected. If so, RV should be able to create left and right buffers. There are a number of different ways to view stereo with a standard graphics card. See Chapter *B* for information about how to setup each platform and what options are available at the hardware level.

Typically, hardware support requires shutter glasses (monitor or projection) or polarized glasses (projection only) in order to be useful.

19.6 12.6 Additional Stereo Operations

These options can be applied per-source as well as part of the global viewing stereo options. The per-source options can be found under the Image → Stereo menu and the global view options are under View → Stereo.

19.6.1 12.6.1 Swap eyes

Swap eyes changes the order of the left and right eyes (left becomes right and vice versa). If you are looking at stereo pairs and you just cannot see the stereo, you may want to try swapping the eyes. It is difficult to view stereo when the eyes are inverted.

19.6.2 12.6.2 Relative Eye Offset

Relative eye offset controls how separated the left and right images are horizontally. In the case of the anaglyph and hardware left and right buffer modes the offset value determines the fusion depth. Objects at the fusion depth appear coincident with the screen depth. In other words, they appear to be right on the screen: not behind it or in front of it. This is most evident in the anaglyph mode where the red and cyan components of a shape will not be separated if it's at the fusion depth. The units of the offset numbers are a percentage of the image width.

You have a choice to either offset the eye images away from each other (both at the same time) or to offset the right eye only.

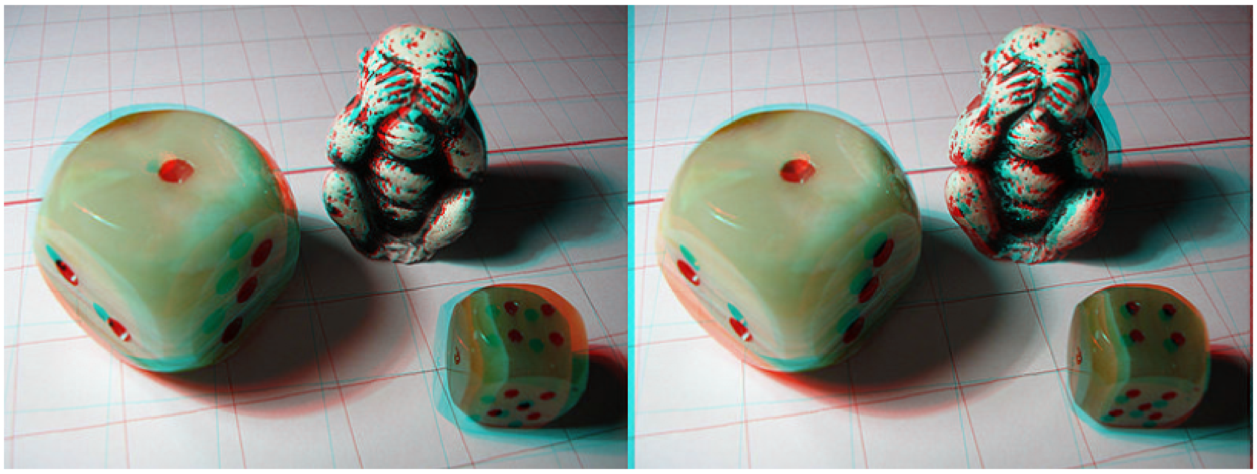


Figure 12.8:

Changing stereo relative eye offset. The left is the original image viewed in anaglyph mode. The right has an additional offset applied.

19.6.3 12.6.3 Flip/Flop the Right Eye Only

If you are projecting stereo and require one eye be flipped (vertical) or flopped (horizontal), you can select Image → Stereo → Flip Right Eye or Image → Stereo → Flop Right Eye. This can further be combined with Image → Flip and Image → Flop and rotation to move the images into the correct position.

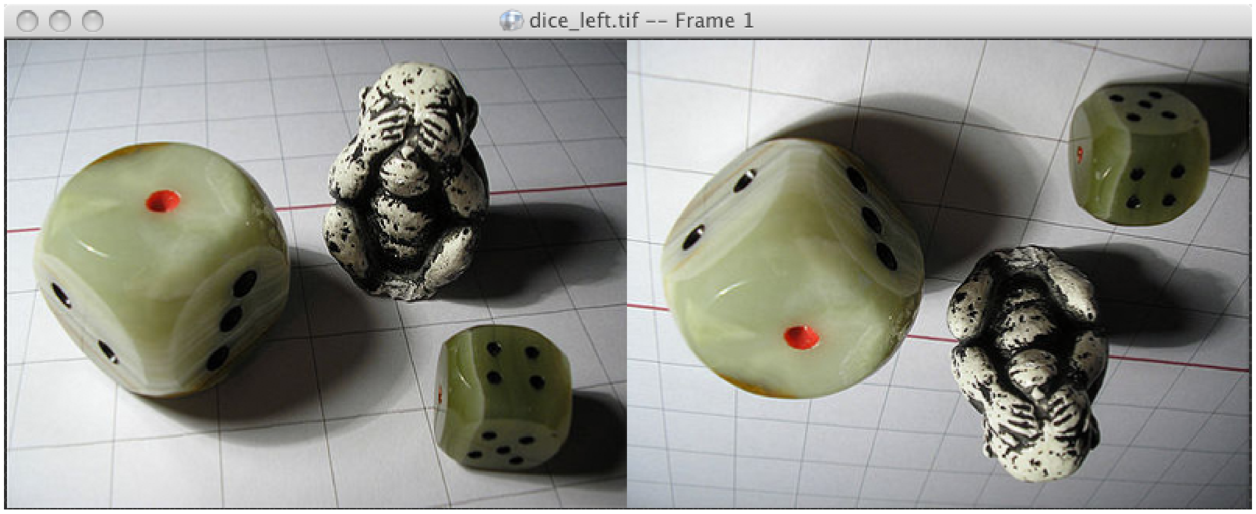


Figure 12.9:

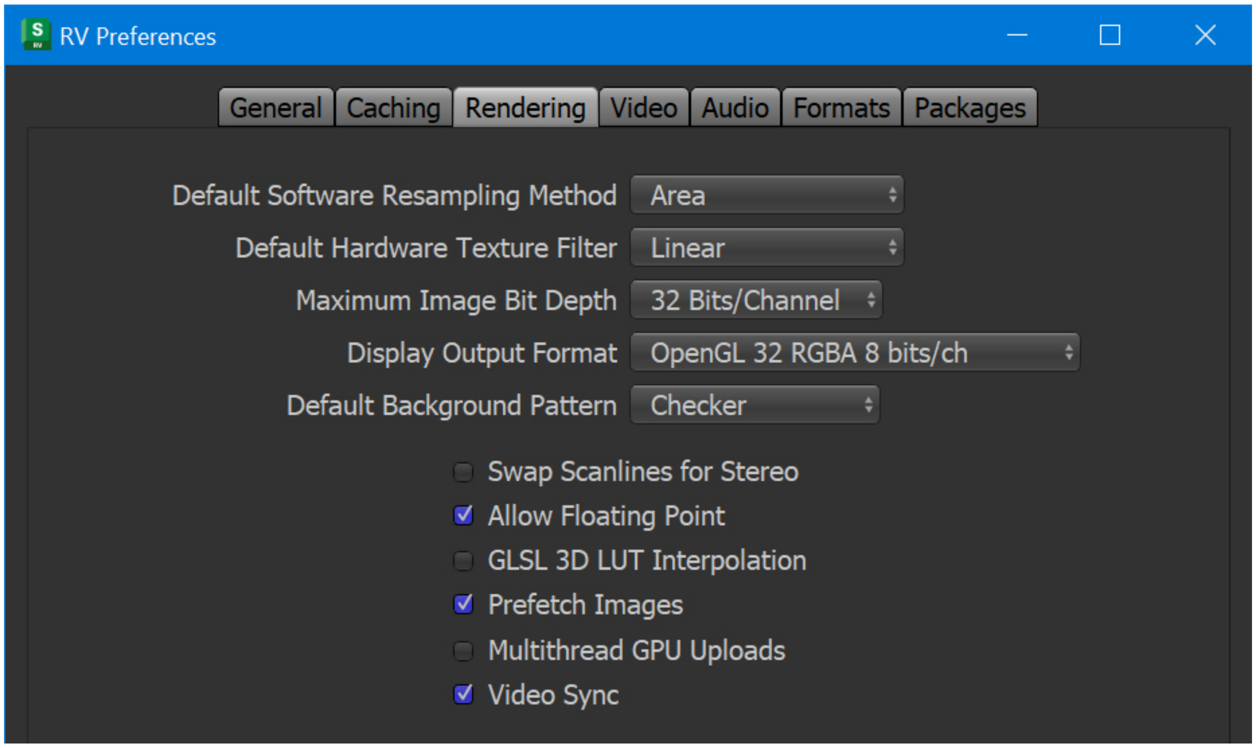
Flipping One Eye

CHAPTER 14 - MAXIMIZING PERFORMANCE

RV has various features intended to make it possible to play back high resolution media from fast I/O devices. If the playback performance is not sufficient and does not meet your needs, you can tune these features to eke out even more frames per second.

There are more than a few variables that determine I/O and decoding speed. When tuning RV you want to start with a simple set of parameters and then adjust one at a time. If you try to adjust all of them at the same time you can't isolate the contribution of each setting and make it much harder to figure out a sweet spot.

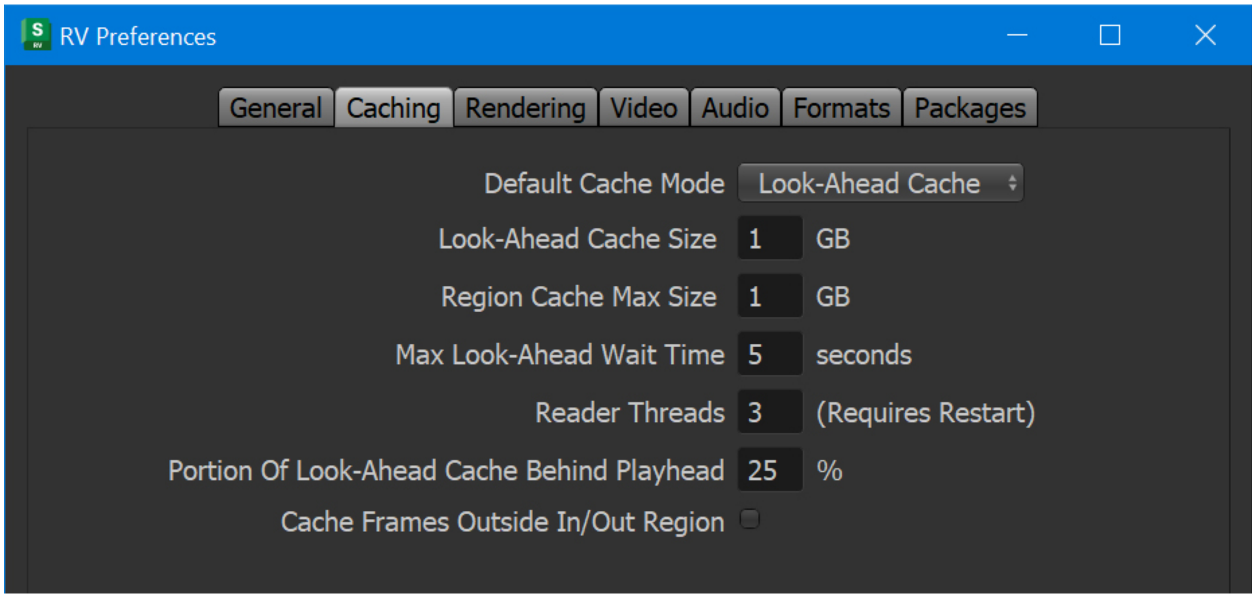
Most of these settings are available from either the Caching preference or the Rendering preference panes.



Default Software Resampling Method: Area
Default Hardware Texture Filter: Linear
Maximum Image Bit Depth: 32 Bits/Channel
Display Output Format: OpenGL 32 RGBA 8 bits/ch
Default Background Pattern: Checker

- [ ] Swap Scanlines for Stereo
- [✓] Allow Floating Point
- [ ] GLSL 3D LUT Interpolation
- [✓] Prefetch Images
- [ ] Multithread GPU Uploads
- [✓] Video Sync

Figure 14.1: Render Preference Pane



Default Cache Mode Look-Ahead Cache

Look-Ahead Cache Size 1 GB

Region Cache Max Size 1 GB

Max Look-Ahead Wait Time 5 seconds

Reader Threads 3 (Requires Restart)

Portion Of Look-Ahead Cache Behind Playhead 25 %

Cache Frames Outside In/Out Region

Figure 14.2: Caching Preference Pane

20.1 Internal Software Operations

Some operations occur in software in RV. For example, when you read images in at a reduced resolution, the image has to be resampled in software. When software operations are being performed on incoming images, it's a good idea to use caching. If direct from disk playback performance is important, then these operations should be avoided:

- Image resolution changes
- Pre-Cache LUT
- Color resolution changes (for example, float to 8-bit int color)
- Cropping
- Channel remapping

The use of cropping can either increase or decrease performance depending on the circumstances.

20.1.1 Streaming I/O Image Formats

Streaming IO works with the following image file formats:

- DPX and Cineon
- JPEG
- TARGA (TGA)
- TIFF
- EXR

20.2 Steps to Improve Performances

When RV is installed, the installer sets the RV preferences to default settings. These settings should allow you to play back correctly most media streams. If this is not the case, you can follow these steps to maximize playback performance.

Note: To reset RV to default preference, use the `-resetPrefs` command line option.

1. Verify that your workstation meets the recommended system requirements.
2. In RV Preferences > Caching, set **Default Cache Mode** to **Lookahead Cache**. See *Lookahead Cache* for more details.
3. In RV Preferences > Caching, increase the number of **Reader Threads**. Increase this number by 1 and then test playback. See ***Reader Threads*** for more details.
4. In RV Preferences > Formats, try alternate **I/O Methods** for the format you're trying to play back. See ***I/O Methods*** for more details.
5. In RV Preferences > Formats, select **Prefetch Images**. This doubles the amount of VRAM required, but significantly increases the amount of bandwidth between RV and the graphics card.
6. On Linux, set up RV to make it run real-time. See *Real-time RV on Linux* for more details.

If the frames per second performance of RV are fine, but you have issues with tearing, see *Refresh Rate and V-Sync*.

To test playback performance, see *Testing Playback Performance*.

20.2.1 More Optimization Tips

In addition to the more generic optimizations listed before, the following techniques can help you in some specific cases.

- EXR B44 images are ideally subsampled as 4:2:0. Also, keep in mind that B44/A must be 16 bit. PIZ, ZIP, and ZIPS encoded EXR are CPU intensive and may require more EXR decoder threads.
- If you have recent graphics card try setting the DPX and Cineon 10-bit display depth to **10 Bits/Pixel Reversed** in Preferences > Format. This is the fastest and most color preserving method of dealing with 10-bit data in RV. If you have a "30 bit" capable monitor you can also put the X server or Windows in 30-bit mode to get both high precision color and fast streaming this way.
- Displaying 10-bit DPX in 16-bit mode requires 2x the bandwidth from the I/O device AND to the graphics card that 8-bit mode does. If you can use the 10-bit mode for DPX/Cineon try 8-bit first.
- If your I/O device has significant latency, you may need to increase the number of reader threads dramatically to amortize the delay. It some cases it may be beneficial to use more threads than you have cores. This can happen with network storage.
- Make sure RV's v-sync is not on at the same time that the driver's GL v-sync is on. For more information, see *Refresh Rate and V-Sync*.
- DPX files which are written so that pixel data starts 4096 bytes into the file, are little-endian, and which use four channel 8 bit, 3 channel 10 bit, or 4 channel 16 bit, and which have a resolution width divisible by 8.
- For speed tests, be sure you're in "Play All Frames" mode (Control menu) as opposed to "Realtime". In this context, "Realtime" means that RV skips video frames to keep pace with the audio.

20.3 System Requirements

While high performance playback cannot be guaranteed, these minimum system requirements should get you reasonable performances.

Component	Recommended Minimum
CPU	64-bit, 6 cores, 12 logical processors
Memory (RAM)	16 GB
Memory (GPU)	8 GB
Storage	RAID or nvme
Network	1 Gb/s.

Since RV aggressively caches its media, the network performance does not directly affect playback performance. It does affect how fast media can be cached, so a faster network allows you to start playback earlier.

20.4 Cache Settings

RV has a three cache settings that you can select: off, cache the current in/out range, or look-ahead buffer. In addition to selecting the cache mode, you can also set the size of the cache for each mode.

The **Look-Ahead Cache** attempts to smooth out playback by pre-caching frames right before they are played. This is the best mode to use when RV can read the files from disk at close to the frame rate, or faster. This is why a fast storage is important as it allows RV to fill the cache faster than it's emptied, ensuring smooth playback. If playback catches up to the look-ahead cache, playback pauses until it fills the cache, or for a length of time specified in the Caching preferences under Max Look-Ahead Wait Time. If RV often pauses playback because it catches up to the look-ahead cache, increase Preferences > Caching > Look-Ahead Cache Size. You can increase this size in increments of 1 GB and see if this solves your issue.

The **Region Cache** reads frames starting at the in point and attempts to fill the cache up to the out point. If there is not enough room in the cache, RV stops caching. This is the mode to use when RV struggles to read files from disk because it tries to cache as much content as possible. This cache works well when playing back media sequentially, but struggles when you jump around in a sequence.

In a review context, the Look-Ahead cache mode is the best solution.

20.4.1 RAM and Playback

The fastest playback occurs when frames are cached in your computer's RAM. The more RAM you have, the more frames that RV can cache and the more interactive it becomes. By default RV will load images at their full bit depth and size. For example, a 32-bit RGBA tiff will be loaded, cached, and sent to the graphics card at full resolution and bit depth. This gives artists the ability to inspect images with access to the full range of color information and dynamic range, and makes it possible to work with high-dynamic range imagery.

For playback and review of sequences at speed, you may wish to cache images with different settings to fit more frames into the available RAM. You increase the number of frames that will fit in the cache by having RV read the frames with reduced resolution. For example, reducing resolution by half can result in as many as four times the number of frames being cached.

Similarly, reducing the color resolution can squeeze more frames into memory. For example a 1024x1024 4 channel 8-bit integer image requires 4 Mb of memory internally. The same image as 16-bit floating-point requires 8 Mb and a 32-bit float image requires 16 Mb. So by having RV reduce a 32-bit float image to an 8-bit image, you can pack four times the number of frames into memory without changing the image size.

Not caching the Alpha channel of a 4 channel image will also reduce the memory footprint of the frames. You can tell RV to remap the image channels to R, G, B before caching (See *7.3.2*). This may affect playback speed for other reasons, depending on your graphics card. You will need to experiment to determine if this works well on your system.

20.5 Reader Threads

Multiple reader threads are required for fast I/O streaming, and are used when you set **Default Cache Mode** to Look-Ahead Cache or Region Cache. When caching to the region or look-ahead cache, threads are used to read and process the frames. This significantly improves I/O speed for most formats. You set the number of threads from:

- The command line with `-rthreads`
- The preferences under Preferences > Caching > Reader Threads. Any change to the reader count requires restarting RV to take effect.

Each reader thread commandeers a CPU logical processor. When you install RV, the installer sets the number of reader threads to something appropriate for your computer's CPU.

Number of logical processors	Recommended number of reader threads
8 or fewer	1
12	2
16	3
More than 16	4

When changing the number of reader threads, always increase in increments of 1, and then test the performance. Only increase the number of threads if necessary. Increasing the thread reader count can make other applications unstable, or render the computer less responsive. If that's the case, decrease the number of threads.

If after increasing the thread reader count performances do not improve, try alternate *I/O Methods* for the formats you're trying to play back.

20.6 I/O Methods

The OpenEXR, DPX, JPEG, Cineon, TARGA, and Tiff file readers all allow you to choose between a few different I/O methods. The best method to use depends on the context RV is being used in and can require some experimentation.

You set the I/O methods for each format in Preferences > Formats.

I/O Method	Description
Standard	This uses whatever is considered the standard or normal way to read these files. For example with EXR this uses the “normal” EXR I/O streams that come as part of the EXR libraries. Not available to all formats.
Buffered	The data is “streamed” as a single logical read of all data. The file data is allowed to reside in the filesystem cache if the kernel decides to do so. The file data is decoded after all the data is read.
Unbuffered	Similar to **buffered**, except that RV provides a hint to the kernel that file data should not be put into the filesystem cache. In theory this could lead to faster I/O because a copy of the data is not created during reading. In practice, this has never been observed in the field.
Memory Mapped	The file contents are mapped directly to main memory. This has the advantage that it may not be filesystem cached and the memory is easily reclaimed by the process when no longer needed. In some respects it is similar to the **buffered** method.
Asynchronous Buffered	Similar to **buffered** but the kernel may provide the data to RV in some random order instead of waiting to assemble the data in order itself. In addition the low-level I/O chunk size can be used to tune the I/O to maximize bandwidth.
Asynchronous Unbuffered	Same as Asynchronous buffered, but a hint is provided to the kernel to omit storing the data in the filesystem cache if possible.

Note: Not all I/O methods are supported by all file systems. In particular, the **Unbuffered** I/O method may not be supported by the underlying file system implementation.

Typically, the circumstance in which RV is used dictates which method is optimal.

When using multiple reader threads, asynchronous methods may not scale as well as the synchronous ones.

You can also select the method from the command line. The command line options -exrIOMethod, -dpxIOMethod, -jpegIOMethod, -cinIOMethod, -tgaIOMethod, and -tiffIOMethod are detailed in *Chapter 3 - Command Line Usage*.

20.6.1 Setting EXR Decoding Threads

OpenEXR decoding benefits from more cores. B44 2k full-aperture 4:2:0 sampled files are approximately 4 Mb in size so they don’t require as much bandwidth as DPX files of the same resolution. For 4:4:4 sampled or B44A with an alpha channel more bandwidth and cores may be required. Generally speaking you should have about as much bandwidth as similar DPX playback would require.

When decoding EXR files, you have the option of setting both the number of reader threads in RV and the number of decoder threads used by the EXR libraries. The exact numbers depend on the flavor of B44/A files being decoded.

If you want to stream EXR files you may want to reserve some of your cores for decoding. The number of EXR decoding threads can be changed from the Preferences > Formats > OpenEXR. Always check performances with **Automatic Threads** activated.

If this doesn’t work, deselect **Automatic Threads** and increase the number in Reader/Decoder Threads by 1. Check performances, and increase by thread count by 1. Repeat until performances are acceptable, but don’t go over the number of logical cores of your workstation minus 1 (if you have 16 logical cores, don’t go over 15) or your system will become unstable. If performances are still degraded, you need to look elsewhere, such as upgrading your hardware.

20.6.2 About File I/O and Decoding Latency

When reading frames directly from disk, file I/O is often a huge bottleneck. If your frames live across a network connection (such as an NFS mount) the latency can be even greater. Ideally, if RV is playing frames without caching, those frames would be on a local disk drive, RAID, or SAN sitting on a fast bus.

Part of the I/O process is decoding compressed image formats. If the decoding is compute intensive, the time spent decoding may become a bottleneck. If good playback performance off disk is a requirement, choose a format that does not require extensive decoding (Cineon or DPX), or one that can be parallelized (EXR).

As always, there is a tradeoff between file size and decoding time. If you have a slow network, you might get better performance by using a format with complex expensive compression. If your computer is connected to a local high-speed RAID array or an SSD, then storing files that are easy to decompress but have larger file size may be better.

20.7 Real-Time RV on Linux

On Linux, you can tell RV to run as a real-time application. This mode enables the most stable possible playback on Linux, especially if the machine as been set up with server time slice durations (which is often the case when the machine is tuned for maximum throughput). Ideally, RV runs with **more and smaller** time slices – at least for display and audio threads.

To start RV in this mode use:

```
rv -scheduler SCHED_RR -priorities 99 99
```

RV Linux will try to use either the FIFO or Round Robin scheduler in place of the normal Linux scheduler in this case. To do so, it must have the capability CAP_SYS_NICE. This can be achieved in several ways, including: running `setuid` root on the Linux binary so it has root privileges, or just running RV as root.

On macOS, RV is a real-time application by default and does not require any special privileges.

On Windows, RV elevates its priorities as high as possible without admin privileges.

> Note: When RV runs with higher priorities, this is referring to only two of its threads: the display thread and the audio thread. Neither of these threads do much computational work. They are both usually blocked. So you shouldn't need to worry about RV consuming too many kernel resources.

20.8 Refresh Rate and V-Sync

The biggest hurdle to smooth playback is to recognize the effects of playback FPS coupled with the monitor refresh rate. For example, it's typically the case that an LCD monitor will have a refresh rate of ~60 Hz by default (that is, it refreshes 60 times a second). Playing 24 FPS material on a 60Hz monitor will result in something similar to a 3/2 pull-down. To get smoothest playback, the ideal refresh rate would be 48 Hz or 72 Hz or some other multiple of 24. Also, a "60 Hz" monitor may actually be 59.88 Hz which means that even 30 FPS material will not play back perfectly smoothly. You need 59.88 / 2 (29.94) FPS for best results.

20.8.1 Multi-monitor Systems

Systems with multiple attached monitors have another problem: the monitor that RV is playing on may not be the monitor that it's syncing to. If that happens the play back can become irregular. On Linux for example, the driver can only sync to one monitor – it can't change the sync monitor once RV has started. On Linux you can change which monitor the driver uses for sync by setting the environment variable **__GL_SYNC_DISPLAY_DEVICE.** Here's a relevant passage from Nvidia's driver README:

> When using __GL_SYNC_TO_VBLANK with TwinView, OpenGL can only sync to one of the display devices; this may cause tearing corruption on the display device to which OpenGL is not syncing. You can use the environment variable __GL_SYNC_DISPLAY_DEVICE to specify to which display device OpenGL should sync. Set this environment variable to the name of a display device; for example "CRT-1". Look for the line "Connected displaydevice(s):" in your X log file for a list of the display devices present and their names. You may also find it useful to review Chapter 10, *Configuring TwinView* "Configuring Twinview," and the section on Ensuring Identical Mode Timings in Chapter 16, *Programming Modes*.

20.8.2 V-Sync versus driver V-Sync

When playing 24 frames/second (24Hz) media on a monitor with a refresh rate of 60Hz (60 frames per second) you can only emulate the timing of frames being played on a 24Hz movie projector. It's not possible to literally get the same timing. The reason is simple math: you can't evenly divide 60 by 24.

RV tries to get around this issue by playing back at as close to the device rate as possible (in this case 60 Hz) and by trying to spread out 24 frames over the 60 actual frames in a visually pleasing way. If you were to look at the timing of the frames you would see that some frames are played back 3 times in a row and others only 2.

The only way around this problem is to match the output device rate to the media rate in such a way that the device rate is evenly divisible by the media rate.

Do not run RV with both the driver's GL v-sync on and RV's. This will almost guarantee bad playback. Use one or the other. Experiment to see if one results in better timing than the other on your system. These settings are:

- In RV: Preferences > Rendering > Video Sync
- In nvidia-settings: openGL > Sync to VBlank

NVIDIA recommends using the driver v-sync and disabling RV's v-sync if possible. On Linux, RV tries to detect which monitor the driver is using for sync and warn you if it's not the one RV is playing on (it outputs an INFO message in the shell or console window). In presentation mode, RV shows a message box if the presentation device is not the sync device on Linux.

20.9 Testing Playback Performances and the Filesystem Cache

All operating systems (that RV runs on) try to maximize IO throughput by holding some pages from the filesystem in memory. The algorithms can be hard to predict, but the upshot is that if you're trying to test real-time streaming IO performance this "assistance" from the OS can invalidate your numbers; to be clear, for testing, you want to ensure that you start each run with **none** of the sequence to be played in memory. One way to do this is with a very large test set. Say several sequences, each several thousands of frames. If the frames are big enough and you RAM is small enough, then switching to new sequence for each testing run will ensure that no part of the new sequence is already in the filesystem cache.

But a more certain way to ensure that none of your frames are in the memory, which also lets you use the same sequence over and over for testing, is to forcibly clear the filesystem cache before each testing run.

On Linux you can run this command:

```
sudo echo 1 > /proc/sys/vm/drop_caches
```

On macOS, use the command `purge` on the command line to achieve the same result.

CHAPTER

TWENTYONE

CHAPTER 15 - FILE FORMATS

Each platform has a different set of file formats which RV can read by default. In addition, it's possible to download or purchase additional file format plug-ins which allow RV to read even more. This chapter is an overview of the most important formats and how RV uses them.

You can have RV dump out all of the formats and codecs which it understands on the command like by giving it the -formats option.

```
shell> rv -formats
```

If you don't see a codec or container format in the list, then RV doesn't support it without installing one or more plug-ins.

21.1 15.1 Movie File Formats

Movie files are single files which contain many images and often audio. These are often called *container file formats* because they usually specify how to store data, but not how it should be used. In most cases, that includes compression methods, play back algorithms, or even what the meaning of the data in the container is.

Container file formats have additional sub-formats called *codecs* which determine things like compression and methods of play back. So even though a program like RV might be able to open a container file and look inside it, it might not understand one or more of the codecs which are being used in the container. In some cases the codec might be proprietary or meant to be used with a specific piece of software or hardware.

21.1.1 15.1.1 Stereo Movie Files

Most movie file formats can have multiple tracks. When RV reads a movie file with two or more tracks it uses the first two tracks as the left and right eye when in stereo mode. You can create these files using the Apple QuickTime Player or RVIO on macOS and Windows or RVIO on Linux.

21.1.2 15.1.2 Text Tracks

On Linux, RV will read the text track of a movie file if it exists and put the contents in an image attribute per frame. You can see this with the image info widget. Text tracks can be used to store metadata about the movie contents in a cross platform manner.

21.1.3 15.1.3 QuickTime Movie Files (.mov)

RV uses a plugin which leverages ffmpeg directly to handle as many formats and codecs as possible.

There are literally hundreds of codecs which can appear in a QuickTime file, but only some of them are useful in post-production. RV tries to support the most popular and useful ones.

Photo-JPEG

This codec has qualities which make it popular in film post production. Each frame is stored separately in the QuickTime file so moving to a random frame is fast. JPEG offers a number of ways to compress the image data including sub-sampling of color versus luminance and using sophisticated compression techniques. Color representation can be excellent when using Photo-JPEG. File sizes are moderate.

The ffmpeg plugin does not respect all of the QuickTime atom objects. Usually this is a good thing as Apple's own library produces inconsistent results from platform to platform when displaying movie files with gama and colr atoms. The most notorious of the color atoms which typically affects the color of Photo-JPEG movies is the gama atom. The gama atom causes Apple's Quicktime to apply two gamma corrections to the image before it is displayed. RV will not do that and RVIO does not include the gama atom when writing Quicktime movies.

Motion-JPEG

Motion JPEG is similar to Photo-JPEG in that they both use the same compression algorithm. The main difference is that Motion-JPEG essentially stores a single frame in two parts: all of the even scanlines as a single JPEG image and all of the odd scanlines as a separate image. The codec interlaces the two parts together to form the final image.

Motion-JPEG is supported on all platforms.

H.264 (avc1)

This is the latest and greatest codec which is usually associated with MPEG-4. H.264 stores keyframes and data which helps it generate in-between frames on the fly. Because it doesn't store every frame individually, H.264 compressed files can be much smaller than codecs like Photo-JPEG. The image quality for H.264 can be good depending on how the movie was created. If relatively simple creation software was used (like Apple's QuickTime player or RVIO) the results are usually OK, but not nearly as good as Photo or Motion-JPEG.

RAW Codecs

There are a number of *raw* codecs for QuickTime. Some of these store the pixel data as RGB, others as YUV. The Raw codecs tend to be very fast to read and play back. RV supports a number of raw codecs on all platforms.

Audio

RV supports most raw uncompressed audio codecs across platforms.

RV and RVIO handle stereo audio. RV does not currently handle more than two channels of audio.

21.1.4 15.1.4 MPEG-4 Movie Files (.mp4)

The MPEG-4 container file (.mp4) is almost identical to the QuickTime container file (.mov). The same codecs may be used to store data in either format. However, typically you find files encoded with H.264 or one of its predecessors.

RV supports the MPEG-4 container on all platforms.

21.1.5 15.1.5 Windows AVI Files (.avi)

RV supports AVI files with the same codecs as QuickTime on all platforms.

21.1.6 15.1.6 Windows Media Files (.wmv)

There is no official support for this file format.

21.1.7 15.1.7 RV's movieproc Format (.movieproc)

The movieproc "format" is not really a file format— all of the information about the pixels is encoded in the name of the file. So the file doesn't even need to exist on disk to use it.

You might use a movieproc as a source if you need a procedural movie object like color bars or a hold on a black or other solid color frame in in a sequence.

The syntax of the file name is a follows:

TYPE, *OPTION=VALUE,OPTION=VALUE,OPTION=VALUE* .movieproc

where TYPE is one of solid, smptebars, colorchart, noise, blank, black, white, grey/gray, hramp/hbramp, hwramp, or error, and *OPTION* and *VALUE* are one of those listed in *15.1* . A blank movieproc is one that will render no pixels but can occupy space in a sequence, and so can be used to form sequences with "holes".

Option	Value
start	frame number
end	frame number
fps	frames-per-second (e.g. 30 or 29.97)
inc	frame increment (default is 1)
red, green, blue	floating point value [-inf, inf] (used as one "side" for ramp types)
grey,gray	short hand for red=green=blue image
alpha	floating point value [0, 1]
width	Image width in pixels
height	Image height in pixels
depth	Channel Bit Depth: one of 8i, 16i, 16f, or 32f
interval	interval in seconds for "sync" and "flash" options below
audio	"sine" for continuous tone, or "sync" for once-per-interval chirp
freq	Audio Frequency (pitch), e.g. 440 for concert A
amp	Audio Amplitude [0,1]
rate	Audio Rate in Samples-per-second (e.g, 44100 for CD quality)
hpan	"animate" by shifting hpan pixels to the left each frame
flash	flash one frame every interval
filename	base64 string to spoof filename
attr:NAME	add an attribute named NAME to the Framebuffer with string value

Table 15.1: Movieproc Options

For example the following will show color bars with a 1000Hz tone:

```
smptebars,audio=sine,freq=1000,start=1,end=30,fps=30.movieproc
```

and to make 100 black HD 1080 frames:

```
solid,start=1,end=100,fps=24,width=1920,height=1080.movieproc
```

or for an orange frame:

```
solid,red=1,green=.5,blue=0,width=1920,height=1080.movieproc
```

Anywhere you might use a normal file or sequence name in RV or RVIO you can use a movieproc instead.

21.2 15.2 Image File Formats

Each platform which RV runs on has its own selection of image formats. There are few important ones which are implemented across all platforms. Some of the most important formats are discussed in this chapter.

21.2.1 15.2.1 OpenEXR

OpenEXR (EXR) is a high dynamic range floating point file format developed at ILM. It can store both 32 and 16 bit ``half" floating point values with or without compression. RV supports the EXR half float type natively and when the GPU is capable, will render using type half type directly. RVIO is capable of converting to and from the half and full float formats.

The EXR format is extremely flexible, capable of holding everything from multiple views (for stereo) to rendered layers like isolated diffuse and specular components as separate images. In addition it's possible to store subsampled chroma images or combinations of all the above.

There have been at least two important revisions to the original OpenEXR specification, one the "multi-view" spec adds official support for top-level "view" objects that contain channels, and the most recent, the "multi-part" spec adjusts the file format so that groups of channels can be stored in discrete sections ("parts") for efficient I/O. RV supports all varieties of EXR file supported by the latest specifications, but attempts to hide some of the resulting complexity from the user, as discussed below.

Multiple Views

Multiple view EXR files as defined by the Weta Multiview Extension are supported by RV. When in stereo mode, RV will look for views called ``left" and ``right" by default and can be programmatically told to use other channels (see the Reference Manual).

You can also select one or more views in the UI to be loaded specifically when not in stereo mode. In stereo mode if you specify two views those will become the left and right.

RV defaults to loading the default (or first) EXR layer from the view. The EXR views are independent of EXR layers which are described below — there can be multiple views each of which has multiple layers (or vice versa if you prefer to think of it that way).

RV will recognize the file extensions ``exr", ``sxr" (stereo exr), or ``openexr" as being OpenEXR files. Stereo views may be stored as either ``exr" or ``sxr"; RV does not distinguish between the two extensions.

Layers

A layer in EXR terminology is a collection of channels which share a common channel name prefix separated by a dot character. Although EXR layers can have sub-layers and so on, in RV the layer hierarchy is flattened to a single level. So for example, an EXR channel in a traditional multi-view EXR file (multi-part files are discussed below), might have a channel called “left.keyLight.Diffuse.R”. In RV’s simplified usage, the channel “R” is a member of the layer “keyLight.Diffuse”, which is a member of the view “left”.

EXR layers are usually used to store components of images output by a renderer like Pixar’s prman or Mental Image’s Mental Ray. Often these layers are recombined in compositing software like Nuke which can handle the internal structure of the EXR file. This makes it easy for a compositor to control render output at a fine level to match it into a shot. Note that this not how EXR views are used — they are used for indicating stereo eyes (for examples) and each view may itself contain multiple EXR layers.

RV initially will load the default layer (the one that has no name) or the first layer it finds. If this layer has R, G, B, A, Z, Y, RY, or BY channels, these will be assigned accordingly. If there are no obvious channel assignments to be made to the red, green, and blue channels, RV will take the first four channels it finds in the layer. If there are additional channels, you can assign these explicitly using the channel remap function in the UI under Image → Remap Source Image Channels.

You can view alternate layers by selecting them in the Session Manager, or by selecting a layer on the command line.

Y RY BY Images and Subsampling.

By default, RV will read EXR files as planar images. Normally this distinction does not have any real effect at the user level, but in the case of Y RY BY images — which can be sub-sampled — it can have a big impact on playback performance. EXR has two special lossy compression schemes, B44 and B44A, which allow fast decode of high dynamic range imagery. B44 maintains a fixed size file regardless of the contents while B44A can potentially make smaller file sizes. As of OpenEXR v2.2.0, two further lossy DCT compression schemes were added i.e. DWAA (based on 32 scanlines) and DWAB (based on 256 scanlines). The level compression for DWAA and DWAB can be set through the ‘-quality ’ option in rvio when writing out EXR files. This value is stored in the exr header (single/multipart) as a float attribute called ‘dwaCompressionLevel’ and defaults to 45 for DWAA and DWAB.

When used with Y RY BY images with sub-sampled chroma (the BY and RY channels) RV will use the GPU to resample the chroma on the card resulting in faster throughput. When coupled with one or more multi-core CPUs, RV can get good direct from disk performance for these types of images while keeping the HDR information intact. At some resolutions, RV can even play back stereo HDR imagery in real-time from disk when used with the correct hardware.

The OpenEXR file will only use B44 or B44A with half float images currently. 32 bit float images will not typically get as good performance.

Chromaticities

EXR files may have chromaticities (primaries) included in the image attributes. If RV sees the **Color/Primaries** attribute, it will apply the corresponding transform to the Rec 709 primaries by default. You can turn off this behavior by selecting Color → Ignore File Primaries to turn off the transform.

Channel Inheritance

Some programs may assume channels should be "inherited" in EXR files. For example, a renderer may write a single alpha channel in the default layer and exclude redundant alpha channels from additional layers in the file like diffuse and specular. The idea is that these layers will share the default alpha during compositing.

RV can attempt to aggregate channels assumed to be related like this by using either the command line option -exrInherit or by setting the flag in preferences (under the OpenEXR format).

Data/Display Window Handling

How an EXR image is displayed on screen, under certain data/display window overlap permutations, can vary across commercial and/or in-house viewing tools. To accommodate this and allow RV to emulate these differing behaviors, we provide several exr format preferences; see the table below for typical settings.

The two OpenEXR preferences, found under Preferences-Format-OpenEXR, that define RV's data/display window handling behavior are -exrReadWindow and -exrReadWindowIsDisplayWindow. The flag -exrReadWindow specifies how the final data/display window (i.e ReadWindow) is defined. The choices are "Data Window", "Display Window", "Data Window Inside Display Window" and "Union of Data and Display Window". In addition the flag -exrReadWindowIsDisplayWindow maybe optionally set to always make the ReadWindow the EXR display window; otherwise the display window is determined by source's EXR display window attributes.

Table 15.3: OpenEXR format settings that emulate data/display window handling of various tools*

Read Window	Read Window Is Display Window	Tool
Data Window	Checked	OSX Preview, Adobe Photoshop
Display Window	Not applicable	Nuke, exrdisplay
Data Window Inside Display Window	Not applicable	
Union of Data and Display Window	Checked	Renderman it, exrdisplay -w

* = Subject to change by the vendor.

Parts

As of OpenEXR version 2.0, EXR files can be written in separate sections, called "parts". Each part has it's own header, and the underlying OpenEXR API can quickly skip to parts requested by the reader. Parts can contain layered and unlayered channels and be associated with views, but parts have names and so can also act as "layers" in the sense that unique parts can contain channels of the same name.

In order to merge these possibly multiple and simultaneous uses of "parts" into the standard view/layer/channel hierarchy, RV subsumes EXR "parts" into the definition of "layer" when necessary. That is, as far as RV is concerned, a fully-specified channel name always looks like **view.layer.channel** , where (in the case of an EXR file) the three period-separated components of the name have these meanings:

Table 15.4: Components of a fully-qualified channel name.

view	The name of the stereo view. This may be missing or "default" for "the default view". The view name cannot contain a "." Example: "left", "right", "center", etc.
layer	This component may include part names as well as traditional EXR layer names. The rule is that if there are no channel-name conflicts within a single view, the part names will be ignored (since they are not necessary to distinguish the channels), if there **is** a channel-name conflict, the part names maybe incorporated into the layer names. Sub-layers are also included. To be concrete, a channel called "Diffuse.R" stored in a part called "keyLight" is considered to be a member of a layer called "keyLight.Diffuse" if the channel "Diffuse.R" also appears in another part within the same view.
channel	The last component of a traditional EXR channel name, containing no "." Example: "R", "G", "B", "A", "Z", etc.

If an EXR has multiple parts and no channels are selected, RV will make a best effort to determine the most viable default channels. If multiple layers are similarly viable, RV will prefer a layer with RGB or RGBA channels from the first part (part 0). Unless explicitly directed, the channels chosen will not cross part boundaries unless Guess Channel Inheritance is enabled.

Note: You can obtain uncached playback FPS speedups of between 2-3x (OSX/Linux/Windows) with multipart exr files over single part exr files for deep channeled images.

21.2.2 15.2.2 TIFF

TIFF files come in many flavors some of which are rarely used. RV supports a useful subset of all possible TIFF files. This includes 32 bit floating point and multiple channels (beyond four) in both tiled and scanline. RV can read planar and interleaved TIFF files, but currently only reads the first image directory if there are more than one.

RV will read all TIFF tags including EXIF tags and present them as image attributes.

21.2.3 15.2.3 DPX and Cineon

RV and RVIO currently support 8 and16 bit, and the common 10 and 12 bit DPX and 10 bit Cineon files reading direct from disk on all platforms. The standard header fields are read and reported if they contain useful information (e.g., Motion Picture and TV fields). We do not currently support any of the vendor headers.

RV supports the linear, log, and Rec. 709 transfer functions for DPX natively and others through the use of LUTs.

RV decodes DPX and Cineon to 8 bit integer per channel by default. However, the reader can be configured from the command line or preferences to use 16 bit if needed. In addition the reader can use either planar or interleaved pixel formats. We have found that the combination of OS and hardware determines which format is fastest for playback, but we currently recommend RGB8_PLANAR on systems that can use OpenGL Pixel Buffer Objects (PBOs) and RGBA8 on systems that cannot. If you opt for 16 bit pixels, use the RGB16_PLANAR as the pixel format on PBO equipped hardware for maximum throughput.

For best color fidelity use RV's built in Cineon Log → Linear option and sRGB display (or a particular display LUT if available). This option decodes the log space pixels directly to linear without interpolation inside a hardware shader and results in the full [0 , 13.5] range.

RVIO decodes Cineon and DPX to 16 bit integer by default.

The DPX and Cineon writers are both currently limited to 10 bit output.

21.2.4 15.2.4 IFF (ILBM)

The IFF image format commonly created by Maya or Shake is supported by RV including the 32 bit float version.

21.2.5 15.2.5 JPEG

RV can read JPEG natively as Y U V or R G B formats. The reader can collect any EXIF tags and pass them along as image attributes. The reader is limited to 8 bits per channel files. Like OpenEXR, DPX, and Cineon, JPEG there is a choice of I/O method with JPEG images.

21.2.6 15.2.7 RAW DSLR Camera Formats

There are a number of camera vendor specific formats which RV can read through RV's cross platform image plugin io_raw. This plugin uses the open src package LibRAW. The table below list some of the common raw formats that are read with this plugin.

Table 15.7:

Supported RAW formats

File Extension	File Format
.arw	Sony/Minolta RAW
.cr2	Canon RAW 2
.crw	Canon RAW
.dng	Digital NeGative
.nef	Nikon Electronic Format
.orf	Olympus RAW
.pef	Pentax RAW
.ptx	Pentax RAW 2
.raf	Fuji RAW
.rdc	Ricoh RAW
.rmf	Canon Raw Media Format

21.3 15.3 Audio File Formats

RV supports a number of basic uncompressed audio file formats across platforms. On Windows and macOS a number of compressed formats may be supported. Currently use of Microsoft wave files and Apple's AIFF formats are the best bet for cross platform use. RV does support multichannel audio files for playback to multichannel audio devices. (see Appendix *J*).

21.4 15.4 Simple ASCII EDL Format

Each line of the ASCII file is either blank, a comment, or an edit event. A comment starts with a '#' character and continues until the end of a line. A comment can appear on the same line as an edit event.

The format of an edit event is:

```
"SOURCE" START END
```

where SOURCE is a the path to a logical source movie such as:

```
"/some/path/to/foo.mov"
"/some/path/to/bar.1-100#.exr"
"/some/other_path/baz.1-100#.exr"
```

Note that the SOURCE name must always be in double quotes. If the path includes spaces, you do not need to use special escape characters to make sure they are accepted. So this:

```
"/some/place with spaces/foo.mov"
```

is OK.

START and END are frame numbers in the movie. Note that START is the first frame to be include in the clip, and END is the last frame to be included (not, as in some edl formats, the frame **after** the last frame). For QuickTime .mov files or .avi files, the first frame of the file is frame 1.

Here's an example .rvedl file:

```
#
#  4 source movies
#

"/movies/foo.mov" 1 100
"/movies/bar.mov" 20 50
"/movies/render.1-100#.exr" 25 100
"/movies/with a space.#.exr" 1 25
```

CHAPTER

TWENTYTWO

CHAPTER 16 - RVIO

RVIO is a command line (re)mastering tool–it converts image sequence, audio files, and movie files from one format into another including possible bit depth and image color and size resolution changes. RVIO can also generate custom slate frames, add per-frame information and matting directly onto output images, and change color spaces (to a limited extent).

RVIO supports all of the same movie, image, and audio formats that RV does including the .rv session file. RV session files (.rv) can be used to specify compositing operations, split screens, tiling, color corrections, pixel aspect ratio changes, etc.

Table 16.1: rvio Options

-o *string*	Output sequence or image
-t *string*	Output time range (default=input time range)
-tio	Output time range from rv session files in/out points
-v	Verbose messages
-vv	Really Verbose messages
-q	Best quality for color conversions (slower – mostly unnecessary)
-noRanges	No separate frame ranges (1-10 will be considered a file)
-rthreads *int*	Number of reader/render threads (default=1)
-wthreads *int*	Number of writer threads (default=same as -rthreads)
-formats	Show all supported image and movie formats
-iomethod *int* [*int*]	I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=AsyncUnbuffere
-view *string*	View to render (default=defaultSequence or current view in rv file)
-leader ...	Insert leader/slate (can use multiple time)
-leaderframes *int*	Number of leader frames (default=1)
-overlay ...	Visual overlay(s) (can use multiple times)
-inlog	Convert input to linear space via Cineon Log->Lin
-insrgb	Convert input to linear space from sRGB space
-in709	Convert input to linear space from Rec-709 space
-ingamma *float*	Convert input using gamma correction
-fileGamma *float*	Convert input to linear space using gamma correction
-inchannepmap ...	map input channels
-inpremult	premultiply alpha and color
-inunpremult	un-premultiply alpha and color
-exposure *float*	Apply relative exposure change (in stops)
-scale *float*	Scale input image geometry
-resize *int* [*int*]	Resize input image geometry to exact size on input (0 = maintain image aspect)
-resampleMethod *string*	Resampling method (area, linear, cubic, nearest, default=area)
-floatLUT *int*	Use floating point LUTs (1=yes, 0=no, default=1)

Table 1 – continued from previous page

-flut *string*	Apply file LUT
-dlut *string*	Apply display LUT
-flip	Flip image (flip vertical) (keep orientation flags the same)
-flop	Flop image (flip horizontal) (keep orientation flags the same)
-yryby *int int int*	Y RY BY sub-sampled planar output
-yrybya *int int int int*	Y RY BY A sub-sampled planar output
-yuv *int int int*	Y U V sub-sampled planar output
-outparams …	Codec specific output parameters
-outchannelmap …	map output channels
-outrgb	same as -outChannelMap R G B
-outpremult	premultiply alpha and color
-outunpremult	un-premultiply alpha and color
-outlog	Convert output to log space via Cineon Lin->Log
-outsrgb	Convert output to sRGB ColorSpace
-out709	Convert output to Rec-709 ColorSpace
-outgamma	Apply gamma to output
-outstereo *string*	Output stereo (checker, scanline, anaglyph, lumanaglyph, left, right, pair, mirror, hsqueezed, vsqueezed, de
-outformat *int string*	Output bits and format (e.g. 16 float -or- 8 int)
-outhalf	Same as -outformat 16 float
-out8	Same as -outformat 8 int
-outres *int int*	Output resolution (image will be fit, not stretched)
-outfps	Output FPS
-codec *string*	Output codec (varies with file format)
-audiocodec *string*	Output audio codec (varies with file format)
-audiorate *float*	Output audio sample rate (default from input)
-audiochannels *int*	Output audio channels (default from input)
-quality *float*	Output codec quality 0.0 -> 1.0 (use varies with file format and codec default=0.900000)
-outpa* float*	Output pixel aspect ratio (e.g. 1.33 or 4:3, etc, metadata only) default=1:1
-comment *string*	Output comment (movie files, default=””)
-copyright *string*	Output copyright (movie files, default=””)
-debug *string*	Debug category
-version	Show RVIO version number
-exrcpus *int*	EXR thread count (default=*platform dependant*)
-exrRGBA	EXR use basic RGBA interface (default=false)
-exrInherit	EXR guesses channel inheritance (default=false)
-exrIOMethod int [int]	EXR I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=AsyncUn
-jpegRGBA	Make JPEG four channel RGBA on read (default=no, use RGB or YUV)
-jpegIOMethod int [int]	JPEG I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=AsyncUn
-cinpixel *string*	Cineon/DPX pixel storage (default=RGB16)
-cinchroma	Use file chromaticity values (ignores them by default)
-cinIOMethod int [int]	Cineon I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=AsyncU
-dpxpixel string	DPX pixel storage (default=RGB16)
-dpxchroma	Use DPX chromaticity values (ignores them by default)
-dpxIOMethod int [int]	DPX I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=AsyncUn
-tgaIOMethod int [int]	TARGA I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=Async
-tiffIOMethod int [int]	TIFF I/O Method (0=standard, 1=buffered, 2=unbuffered, 3=MemoryMap, 4=AsyncBuffered, 5=AsyncUn
-init *string*	Override init script
-err-to-out	Output errors to standard output (instead of standard error)
-noprerender	Turn off prerendering optimization
-flags …	Arbitrary flags (flag, or ‘name=value’) for Mu or Python

22.1 16.1 Basic Usage

22.1.1 16.1.1 Image Sequence Format Conversion

RVIO's most basic operation is to convert a sequence of images from one format into another. RVIO uses the same sequence notation as RV to specify input and output sequences. For example:

```
shell> rvio foo.#.exr -o foo.#.tif
shell> rvio foo.#.exr -o foo.#.jpg
```

22.1.2 16.1.2 Image sequence to QuickTime Movie Conversion

RVIO can write out QuickTime movies. The default compression codec is PhotoJPEG with a default quality of 0.9.

```
shell> rvio foo.#.exr -o foo.mov
```

22.1.3 16.1.3 Resizing and Scaling

Rvio does high quality filtering when it resizes images. Output resolution can be specified explicitly, in which case RVIO will conform to the new resolution, padding the image to preserve pixel aspect ratio.

```
shell> rvio foo.1-100@@@@.tga -scale 0.5 -o foo.#.tga
shell> rvio foo.#.exr -outres 640 480
```

Or you can use the -resize x y option, in which case scaling will occur in either dimension (unless the number specified for that dimension is 0).

22.1.4 16.1.4 Adding Audio to Movies

RVIO uses the same layer notation as RV to combine audio with image sequences. Multiple audio sources can be mixed in a single layer.

```
shell> rvio [ foo.1-300#.tif foo.aiff ] -o foo.mov
shell> rvio [ foo.1-300#.tif foo.aiff bar.aiff ] -o foo.mov
```

22.2 16.2 Advanced Usage

22.2.1 16.2.1 Editing Sequences

RVIO can combine multiple sequences and write out a single output sequence or movie. This allows you to quickly edit and conform.

```
shell> rvio foo.25-122#.exr bar.8-78#.exr -o foo.mov
shell> rvio [ foo.#.exr foo.aiff ] [ bar.#.exr bar.aiff ] \
       -o foobar.mov
```

Note that you can cut in and out of movie files as well:

```
shell> rvio [ foo.mov -in 101 -out 120 ] [ bar.mov -in 58 -out 123 ] -o out.mov
```

22.2.2 16.2.2 Processing Open RV Session Files

RV session files can be used as a way to author operations for processing with RVIO. Most operations and settings in RV can be used by RVIO. For example RV can be used in a compositing mode to do an over or difference or split-screen comparison. RV can also be used to set up edits with per source color corrections (e.g. a unique LUT for each sequences, exposure or color adjustments per sequences, an overall output LUT, etc.).

```
shell> rvio foo.rv -o foo.mov
```

Any View in the session file can be used as the source for the RVIO output:

```
shell> rvio foo.rv -o foo.mov -view "latest shots"
```

22.2.3 16.2.3 Advanced Image Conversions

RVIO has flags to handle standard colorspace, gamma, and log/lin conversions for both inputs and outputs

```
shell> rvio foo.#.cin -inlog -o foo.#.exr
shell> rvio foo.#.exr -o foo.#.cin -outlog
shell> rvio foo.#.jpeg -insrgb -o foo.#.exr
shell> rvio foo.#.exr -o foo.#.tiff -outgamma 2.2
shell> rvio foo.#.exr -o foo.#.jpeg -outsrgb
shell> rvio foo.#.cin -inlog -o foo.mov -outsrgb \
       -comment "Movie is in sRGB space"
```

22.2.4 16.2.4 LUTs

RVIO can apply a LUT to input files and an output LUT. RVIO's command line only supports one file LUT and one display LUT. The file LUT will be applied to all the input sources before conversion and and the output LUT will be applied to the entire session on output. If you need to process a sequences with a different file LUT per sequence, you can do that by creating an RV session file with the desired LUTs and color settings to use as input to RVIO.

```
shell> rvio foo.#.cin bar.#.cin -inlog -flut in.cube \
       -dlut out.cube -o foobar.mov
```

22.2.5 16.2.5 Pixel Storage Formats and Channel Mapping

RVIO provides control over pixel storage (floating point or integer, bit depth, planar subsampling) and channel mapping. The planar subsampling options are particularly used to support OpenExr's B44 compression, which is a fixed bandwidth, floating point, high-dynamic range compression scheme.

```
shell> rvio foo.#.exr -outformat 8 int -o foo.#.tif
shell> rvio foo.#.exr -outformat 8 int -o foo.#.tif -outrgb
shell> rvio foo.#.cin -inlog -o foo..#.tiff -outformat 16 float
shell> rvio foo.#.exr -outformat 32 float -o foo.#.tif
shell> rvio foo.#.exr -codec B44A  -yrybya 1 2 2 2 -outformat \
```

(continues on next page)

(continued from previous page)

```
       16 float
shell> rvio foo.#.exr -codec B44A  -yryby 1 2 2  -outformat \
       16 float -outchannelmap R G B
```

22.2.6 16.2.6 Advanced QuickTime Movie Conversions

RVIO uses ffmpeg for reading and writing. You can find out what codecs are available for reading and writing by using the "-formats" flag. This will also tell you the full name of the encoder, e.g. writing Motion JPEG requires "mjpeg" by ffmpeg. RVIO lets you specify the output codec and can collect parameters for the output from "outparams". Please examine the following two examples. The first is with the default settings for mjpeg's Motion JPEG writing, and the second is a popular command derived from using the ffmpeg binary directly. Something of the form `ffmpeg -i foo.%04d.tif -ac 2 -b:v 2000k -c:a pcm_s16be -c:v mjpeg -pix_fmt yuv420p -g 30 -b:a 160k -vprofile high -bf 0 -strict experimental -f mov outfile.mov`.

```
shell> rvio foo.#.tif -codec mjpeg -o foo.mov
shell> rvio foo.#.tif -codec mjpeg -audiocodec pcm_s16be \
       -outparams pix_fmt=yuv420p vcc:b=2000000 acc:b=160000 \
        vcc:g=30 vcc:profile=high vcc:bf=0 -o foo.mov
```

22.2.7 16.2.7 Audio Conversions

RVIO can operate directly on audio files and can also add or extract audio to/from QuickTime movies. RVIO provides flags to set the audio codec, sample rate, storage format (8, 16, and 32 bits) storage type (int, float), and number of output channels. RVIO does high quality resampling using 32 bit floating point operations. RVIO will mix together multiple audio files specified in a layer.

```
shell> rvio foo.mov -o foo.aiff
shell> rvio foo.mov -o foo.aiff -audiorate 22050
shell> rvio [ foo.#.exr foo.aiff bar.wav ] -codec H.264 \
       -quality  1.0 \
       -audiocodec "Apple Lossless" -audiorate 44100 \
       -audioquality 1.0 -o foo.mov
```

22.2.8 16.2.8 Stereoscopic and Multiview Conversions

RV and RVIO support stereoscopic playback and conversions. RVIO can be used to create stereo QuickTime files or multiview OpenExr files (Weta's SXR files) by specifying two input layers and using the -outstereo flag. Stereo QuickTime movies contain multiple video tracks. RV interprets the fist two tracks as left and right views. RVIO will also render output using stereo modes specified in an RV session file–this allows you to output anaglyph images from stereo inputs or to render out scanline or alternating pixel stereo material.

```
shell> rvio [ foo_l.#.exr foo_r.#.exr foo.aiff ] -outstereo \
       -o stereo.mov
shell> rvio [ foo_left.#.exr foo_right.#.exr ] -outstereo \
       -o stereo.#.exr
```

Or you can specify an output stereo format:

```
shell> rvio [ foo_l.#.exr foo_r.#.exr foo.aiff ] -outstereo hsqueezed \
       -o stereo.mov
```

22.2.9 16.2.9 Slates, Mattes, Watermarks, and Burn-ins

RVIO supports script based creation of slates and overlays. The default scripts that come with RV can be used as is, or they can be customized to create any kind of overlay or slate that you need. Customization of these scripts is covered in the RV Reference Manual. RVIO has two command line flags to manage these scripts, -leader and -overlay. -leader scripts are used to create Slates or other frames that will be added to the beginning of a sequence or movie. -overlay scripts will draw on top of the image frames. Multiple overlays can be layered on top of the image, so that you can build up a frame with mattes, frame burn-in, bugs, etc. The scripts that come with RV include:

- simpleslate
- watermark
- matte
- frameburn
- bug

Simpleslate Leader

Simpleslate allows you to build up a slate from a list of attribute/value pairs. It will automatically scale all of your text to fit onto the frame. It works like this:

```
shell> rvio foo.#.exr -leader simpleslate \
       "Acme Post" "Show=My Show" "Shot=foo" "Type=comp" \
       "Artist=John Doe" "Comments=Lighter and Darker as \
       requested by director" -o foo.mov
```

Watermark Overlay

The watermark overlay burns a text comment onto output images. This makes it easy to generate custom watermarked movies for clients or vendors. Watermark takes two arguments, the comment in quotes and the opacity (from 0 to 1).

```
shell> rvio foo.mov -overlay watermark "For Client X Review" 0.1 \
       -o foo_client_x.mov
```

Matte Overlay

The matte overlay mattes your images to the desired aspect ratio. It takes two arguments, the aspect ratio and the opacity.

```
shell> rvio foo.#.exr -overlay matte 2.35 1.0 -o foo.mov
```

Frameburn Overlay

The frameburn overlay renders the source frame number onto each frame. It takes three arguments, the opacity, the luminance, and the size.

```
shell> rvio foo.#.exr -overlay frameburn 0.1 0.1 50 -o foo.mov
```

Bug Overlay

The bug overlay lets you render an image o top of each output frame. It takes three arguments, the image name, the opacity, and the size

```
shell> rvio foo.#.exr -overlay bug "/path/to/logo.tif" 0.5 48
```

22.2.10 16.2.10 EXR Attributes

RVIO can create and pass through header attributes. To create an attribute from the command line use -outparams:

```
shell> rvio in.exr -o foo.exr -outparams NAME:TYPE=VALUE0,VALUE1,VALUE2,...
```

TYPE is one of:

f	float
i	int
s	string
sv	Imf::StringVector
v2i	Imath::V2i
v2f	Imath::V2f
v3i	Imath::V3i
v3f	Imath::V3f
b2i	Imath::Box2i
b2f	Imath::Box2f
m33f	Imath::M33f
m44f	Imath::M44f
c	Imf::Chromaticities

Values are comma separated. For example to create a Imath::V2i attribute called myvec with the value V2i(1,2):

```
shell> rvio in.exr -o out.exr -outparams myvec:v2i=1,2
```

similarily a string vector attribute would be:

```
shell> rvio in.exr -o out.exr -outparams mystringvector:sv=one,two,three,four
```

and a 3 by 3 float matrix attribute would be:

```
shell> rvio in.exr -o out.exr -outparams myfloatmatrix:m33f=1.0,2.0,3.0,4.0,5.0,6.0,7.0,
↪8.0,9.0
```

where the first row of the matrix would be 1.0 2.0 3.0.

If you want to pass through attributes from the incoming image to the output EXR file you can use the passthrough variable. Setting passthrough to a regular expression will cause the writer code to select matching incoming attribute names.

```
shell> rvio in.exr -o out.exr -outparams "passthrough=.*"
```

The name matching includes any non-EXR format identifiers that are created by RV and RVIO.

```
shell> rvio in.jpg -o out.exr -insrgb -outparams "passthrough=.*EXIF.*"
```

The name matching includes any EXIF attributes (e.g. from TIFF or JPEG files).

22.2.11 16.2.11 IIF/ACES Files

RVIO can convert pixels to the very wide gamut ACES color space for output using the -outaces flag. In addition, use of the .aces extension will cause the OpenEXR writer to enforce the IIF container subset of EXR. For example, to convert an existing EXR file to an IIF file:

```
shell> rvio in.exr -o out.aces -outaces
```

It's possible to write to other formats using -outaces, but it's not recommended.

22.2.12 16.2.12 DPX Header Fields

Using -outparams it's possible to set almost any DPX header field. Setting the field will not change the pixels in the final file, just the value of the header field.

There are a couple of fields treated in a special way: the frame_position, tv/time_code, and tv/user_bits fields are all increment automatically when a sequence of frames is output. In the case of tv/user_bits and tv/time_code, the initial value comes either from the output frame number, or starting at the time code passed in.

Table 16.2: DPX Output Parameters

Keyword	Description
transfer	Transfer function (LOG, DENSITY, REC709, USER, VIDEO, SMPTE274M, REC601-625, REC601-52
colorimetric	Colorimetric specification (REC709, USER, VIDEO, SMPTE274M, REC601-625, REC601-525, NTSC
creator	ASCII string
copyright	ASCII string
project	ASCII string
orientation	Pixel Origin string or int (TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, BOTTOM_RIGHT, ROTATED_
create_time	ISO 8601 ASCII string: YYYY:MM:DD:hh:mm:ssTZ
film/mfg_id	2 digit manufacturer ID edge code
film/type	2 digit film type edge code
film/offset	2 digit film offset in perfs edge code
film/prefix	6 digit film prefix edge code
film/count	4 digit film count edge code
film/format	32 char film format (e.g. Academy)
film/frame_position	Frame position in sequence (0 indexed)
film/sequence_len	Sequence length
film/frame_rate	Frame rate (frames per second)

Table 2 – continued from previou

Keyword	Description
film/shutter_angle	Shutter angle in degrees
film/frame_id	32 character frame identification
film/slate_info	100 character slate info
tv/time_code	SMPTE time code as an ASCII string (e.g. 01:02:03:04)
tv/user_bits	SMPTE user bits as an ASCII string (e.g. 01:02:03:04)
tv/interlace	Interlace (0=no, 1=2:1)
tv/field_num	Field number
tv/video_signal	Video signal standard 0-254 (see DPX spec)
tv/horizontal_sample_rate	Horizontal sampling rate in Hz
tv/vertical_sample_rate	Vertical sampling rate in Hz
tv/frame_rate	Temporal sampling rate or frame rate in Hz
tv/time_offset	Time offset from sync to first pixel in ms
tv/gamma	Gamma
tv/black_level	Black level
tv/black_gain	Black gain
tv/break_point	Breakpoint
tv/white_level	White level
tv/integration_times	Integration times
source/x_offset	X offset
source/y_offset	X offset
source/x_center	X center
source/y_center	Y center
source/x_original_size	X original size
source/y_original_size	Y original size
source/file_name	Source file name
source/creation_time	Source creation time YYYY:MM:DD:hh:mm:ssTZ
source/input_dev	Input device name
source/input_dev	Input device serial number
source/border_XL	Border validity left
source/border_XR	Border validity right
source/border_YT	Border validity top
source/border_YB	Border validity bottom
source/pixel_aspect_H	Pixel aspect ratio horizontal component
source/pixel_aspect_V	Pixel aspect ratio vertical component

This example set the start time code of the DPX sequence:

```
shell> rvio in.#.tif -o out.#.dpx -outparams tv/time_code=00:11:22:00
```

It is also possible to set the alignment of pixel data relative to the start of the file using alignment. For example, to force the pixel data to start at byte 4096 in the DPX file:

```
shell> rvio in.#.tif -o out.#.dpx -outparams alignment=4096
```

The smallest value for alignment is 2048 which includes the size of the default DPX headers.

The DPX writer cannot automatically pass through header fields from input DPX images to the output DPX images.

To set the project header value:

```
rvio in.#.exr -o out.#.dpx -outparams "project=THE PROJECT"
```

To set colorspace header values:

```
rvio in.#.exr -o out.#.dpx -outparams transfer=LINEAR colorimetric=REC709
```

CHAPTER

TWENTYTHREE

CHAPTER 17 - RVLS

The RVLS provides command line information about images, sequences, movie files and audio files.

By default rvls with no arguments will list the contents of the current directory. Unlike its namesake ls, rvls will contract image sequences into a single listing. The output image sequences can be used as arguments to rv or RVIO

1

You can use RVLS to create image sequence completions in bash or tcsh for rv and RVIO. See the reference manual for an example.

.

For more detailed information about a file or sequence the -l option can be used: this will show each file or sequence on a separate line with the image width and height, channel bit depth, and number of channels. For movie files and audio files, additional information about the audio like the sample rate and number of channels will be displayed.

```
shell> rvls -l
        w x h      typ   #ch   file
      623 x 261    16i   3     ./16bit.tiff
     1600 x 1071    8i   3     ./2287176218_5514bbc63a_o.jpg
     1024 x 680     8i   3     ./best-picture.jpg
     2048 x 1556   16f   3     ./sheet.hi.90.exr
     5892 x 4800    8i   1     ./BurialMount.psd
       64 x 64      8i   1     ./caust19.bw
     5892 x 4992    8i   1     ./common_sense.psd
      640 x 486    16i   3     ./colorWheel.685.cin
      640 x 480     8i   1     ./colour-bars-smpte-75-640x480.tiff
```

or to see certain files in a directory as a sequence:

```
shell> ls *.iff
render_maya.1.iff
render_maya.10.iff
render_maya.2.iff
render_maya.3.iff
render_maya.4.iff
render_maya.5.iff
render_maya.6.iff
render_maya.7.iff
render_maya.8.iff
render_maya.9.iff
shell> rvls *.iff
render_maya.1-10@.iff
```

It is also possible to list the image attributes using RVLS using the -x option. This will display the same attribute information that is displayed in RV's Image Info Widget:

```
shell> rvls -x unnamed_5_channel.exr
unnamed_5_channel.exr:

             Resolution   640 x 480, 5ch, 16 bits/ch floating point
               Channels
       PixelAspectRatio   1
    EXR/screenWindowWidth   1
   EXR/screenWindowCenter   (0 0)
     EXR/pixelAspectRatio   1
            EXR/lineOrder   INCREASING_Y
        EXR/displayWindow   (0 0) - (639 479)
           EXR/dataWindow   (0 0) - (639 479)
          EXR/compression   ZIP_COMPRESSION
             Colorspace   Linear
```

Additional command line options can be see in table *17.1*

-a	Show hidden files
-s	Show sequences only (no non-sequence member files)
-l	Show long listing
-x	Show extended attributes and image structure
-b	Use brute force if no reader found
-nr	Do not show frame ranges
-ns	Do not infer sequences (list each file separately)
-min *int*	Minimum number of files considered a sequence (default=3)
-formats	List image/movie formats
-version	Show rvls version number

Table 17.1: RVLS Options

CHAPTER

TWENTYFOUR

CHAPTER 18 - RVPUSH

The RVPUSH command-line utility allows you to communicate with a running RV (you can designate a “target” RV with the -tag option). The help output from the command is as follows:

```
usage: rvpush [-tag <tag>] <command> <commandArgs>
       rvpush set <mediaArgs>
       rvpush merge <mediaArgs>
       rvpush mu-eval <mu>
       rvpush mu-eval-return <mu>
       rvpush py-eval <python>
       rvpush py-eval-return <python>
       rvpush py-exec <python>
       rvpush url rvlink://<rv-command-line>

       Examples:

       To set the media contents of the currently running RV:
           rvpush set [ foo.mov -in 101 -out 120 ]

       To add to the media contents of the currently running RV with tag "myrv":
           rvpush -tag myrv merge [ fooLeft.mov fooRight.mov ]

       To execute arbitrary Mu code in the currently running RV:
           rvpush mu-eval 'play()'

       To execute arbitrary Mu code in the currently running RV, and print the result:
           rvpush mu-eval-return 'frame()'

       To evaluate an arbitrary Python expression in the currently running RV:
           rvpush py-eval 'rv.commands.play()'

       To evaluate an arbitrary Python expression in the currently running RV, and print␣
↪the result:
           rvpush py-eval-return 'rv.commands.frame()'

       To execute arbitrary Python statements in the currently running RV:
           rvpush py-exec 'from rv import commands; commands.play()'

       To process an rvlink url in the currently running RV, loading a movie into the␣
↪current session:
           rvpush url 'rvlink:// -reuse 1 foo.mov'
```

(continues on next page)

(continued from previous page)

```
    To process an rvlink url in the currently running RV, loading a movie into a new␣
↪session:
        rvpush url 'rvlink:// -reuse 0 foo.mov'

    Set environment variable RVPUSH_RV_EXECUTABLE_PATH if you want rvpush to
    start something other than the default RV when it cannot find a running
    RV.  Set to 'none' if you want no RV to be started.

    Exit status:
        4: Connection to running RV failed
       11: Could not connect to running RV, and could not start new RV
       15: Cound not connect to running RV, started new one.
```

Any number of media sources and associated per-source options can be specified for the set and merge commands. For the mu-eval command, it's probably best to put all your Mu code in a single quoted string. In the next release we'll have a python-eval command as well.

If RVPUSH cannot find a running RV to talk to, it'll start one with the appropriate command-line options. Any later rvpush commands will use this RV until it exits. Note that the RV that RVPUSH starts will by default be the one in the same bin directory. If you'd rather start a different RV, or start a wrapper, etc, you can set the environment variable RVPUSH_RV_EXECUTABLE_PATH to point to the one you'd prefer. If you want RVPUSH to never start RV, you can set this environment variable to "none".

CHAPTER

TWENTYFIVE

A - TUNING PLATFORM AUDIO FOR LINUX

On Linux, RV's Platform Audio module is based on Qt's QAudioOutput, which is implemented against ALSA's api. As such, we have added several environment variables that allow us to better debug audio issues and which also allows users to tune the audio data IO performance between RV and their chosen playback audio hardware device. Note these environment variables are only supported on Linux.

The environment variables are as follows:

Environment Variable	Variable values
TWK_QTAUD	0 = No debugging msgs (default) 1 = Standard debugging msgs ; displays the ALSA device hardware parameter values used to configure the audio device. It will also display errors like buffer underruns. 2 = Verbose debugging msgs ; use this value for tracing crashes. Messages are displayed with each audio period write to the audio alsa device.
TWK_QTAUD	value in microsecs e.g. 120000 for 120ms (default is 120000)
TWK_QTAUD	value in microsecs e.g. 20000 for 20ms (default is 20000)

Table K.1:

Platform Audio Environment Variables

```
# setenv TWK_QTAUDIOOUTPUT_ENABLE_DEBUG 1
# setenv RV_NO_CONSOLE_REDIRECT 1
# rv myclip.mov

Version 7.1.0, built on Aug 31 29 2016 at 03:05:11 (HEAD=166a76e). (L)
Copyright (c) 2016 Autodesk, Inc. All rights reserved.
INFO: myclip.mov
DEBUG: Number of available audio devices  3
DEBUG: Audio device =  "default"
DEBUG: Audio device actual =  "pulse"
DEBUG: ranges: pmin= 666 , pmax= 7281792 , bmin= 2000 , bmax= 21845334
DEBUG: used: buffer_frames= 5760 , period_frames= 960     # in sample count
DEBUG: used: buffer_size= 23040 , period_size= 3840       # in bytes
DEBUG: used: buffer_time= 120000 , period_time= 20000     # in microsecs
DEBUG: used: chunks= 6                                    # no of periods per buffer
DEBUG: used: max write periods/chunks=0                   # 0 implies max possile.
DEBUG: used: bytesAvailable= 23040                        # amount of free space in the␣
↪audio buffer on device open().
```

The environment variables TWK_QTAUDIOOUTPUT_BUFFER_TIME and TWK_QTAUDIOOUTPUT_PERIOD_TIME can be used to set the buffer size and period size of the audio device. The audio buffer size is always an integral number of period sizes, in other words chose values such that TWK_QTAUDIOOUTPUT_BUFFER_TIME = N * TWK_QTAUDIOOUTPUT_PERIOD_TIME; where N is an integer. Experimentally we have found N = 6 produced a measured audio - video sync < 10ms; while increasing or decreasing N from this value seemed to produce larger and increasingly worst av sync lag numbers.

It is worth remembering that TWK_QTAUDIOOUTPUT_BUFFER_TIME determines overall size of the audio buffer. The audio device will not start playing until the buffer is completely filled first. This means the buffer size can influence the lag at the beginning when play first starts.

So for a given TWK_QTAUDIOOUTPUT_BUFFER_TIME, TWK_QTAUDIOOUTPUT_PERIOD_TIME determines the number of period buffers within the overall buffer size and this influences the average av sync value and if too small <=3 leads to buffer underun errors and crackles in audio.

```
# setenv TWK_QTAUDIOOUTPUT_ENABLE_DEBUG 1
# setenv RV_NO_CONSOLE_REDIRECT 1
# setenv TWK_QTAUDIOOUTPUT_BUFFER_TIME 60000
# setenv TWK_QTAUDIOOUTPUT_PERIOD_TIME 20000
# rv myclip.mov

Version 7.1.0, built on Aug 31 29 2016 at 03:05:11 (HEAD=166a76e). (L)
Copyright (c) 2016 Autodesk, Inc. All rights reserved.
INFO: myclip.mov
DEBUG: Number of available audio devices  3
DEBUG: Audio device =  "default"
DEBUG: Audio device actual =  "pulse"
DEBUG: ranges: pmin= 666 , pmax= 7281792 , bmin= 2000 , bmax= 21845334
DEBUG: used: buffer_frames= 2880 , period_frames= 960     # in sample count
DEBUG: used: buffer_size= 11520 , period_size= 3840       # in bytes
DEBUG: used: buffer_time= 60000 , period_time= 20000      # in microsecs
DEBUG: used: chunks= 3                                    # no of periods per buffer
DEBUG: used: max write periods/chunks=0                   # 0 implies max possile.
DEBUG: used: bytesAvailable= 11520                        # amount of free space in the␣
↪audio buffer on device open().
DEBUG: *** Buffer underrun: -32
DEBUG: *** Buffer underrun: -32
DEBUG: *** Buffer underrun: -32
DEBUG: *** Buffer underrun: -32
DEBUG: *** Buffer underrun: -32
```

The tuning process steps:

1. Launch RV and setup the File->Preferences->Audio tab settings as follows:
    1. Output Module: Platform Audio
    2. Output Device: Default
    3. Output Format: Stereo 32bit float 48000
    4. Enable 'Keep Audio device open when playing'
    5. Turn off 'Hardware Audio/Video Synchronization'
    6. Enable 'Scrubbing on by default'
2. Determine the smallest TWK_QTAUDIOOUTPUT_BUFFER_TIME and TWK_QTAUDIOOUTPUT_PERIOD_TIME settings before buffer underruns occur.

   1. Set Platform Audio debugging messaging on… TWK_QTAUDIOOUTPUT_ENABLE_DEBUG = 1.
   2. Set values for TWK_QTAUDIOOUTPUT_BUFFER_TIME and TWK_QTAUDIOOUTPUT_PERIOD_TIME.
   3. Try the following combinations of TWK_QTAUDIOOUTPUT_BUFFER_TIME / TWK_QTAUDIOOUTPUT_PERIOD_TIME i.e. 60000 / 10000 and 36000 / 6000.
   4. Launch RV with a 48Khz video clip and enable lookahead/region caching. Either cache option would do as long as you size the video cache appropriately so the entire clip is cached. Playback and observe for buffer underruns messages. Note you should also stress test running multiple RVs with the same config.
   5. If no underun errors occur; repeat the previous step, setting smaller values for TWK_QTAUDIOOUTPUT_BUFFER_TIME and TWK_QTAUDIOOUTPUT_PERIOD_TIME keeping in mind that the ratio of TWK_QTAUDIOOUTPUT_BUFFER_TIME/TWK_QTAUDIOOUTPUT_PERIOD_TIME must be greater than 3 and ideally around 6.

3. Once you have found the smallest value of TWK_QTAUDIOOUTPUT_BUFFER_TIME; use an av sync meter to measure the average av sync lag playing back RV's movieproc syncflash clip.

   1. With TWK_QTAUDIOOUTPUT_BUFFER_TIME fixed to the value determined in the previous step; increase and decrease TWK_QTAUDIOOUTPUT_PERIOD_TIME, bearing in mind TWK_QTAUDIOOUTPUT_BUFFER_TIME must be a integer multiple of TWK_QTAUDIOOUTPUT_PERIOD_TIME. Find the multiple with the lowest measure average av sync values (i.e. the average of at least 25 av sync measurements).

The table below provides the smallest values of TWK_QTAUDIOOUTPUT_BUFFER_TIME and TWK_QTAUDIOOUTPUT_PERIOD_TIME that prevents buffer underruns (i.e. audio corruption/static issues), minimizes the lag at play-start and average av sync. Note the default value for buffer time is 120000 and period time is 20000.

Hardware class	Audio Hardware	OS Machine Specs	BUFFEF	PERIOD_T	Avrg AV sync
Gaming machine	onboard intel HDA/realtek 7.1ch	Centos7.2, Asus Rampage Gene IV, iCore7 3.5Ghz, 32GB 2400Mhz RAM, SSD, Quadro K2200 (nv drv 352.55)	36000	6000	~3ms
HP workstation	onboard intel HDA/realtek 2ch	Centos6.6, HPZ820, Xeon 12core, 48GB RAM, SSD, Quadro K6000 (nv 352.63)	TDB	TDB	TDB
Dell workstation	onboard intel HDA/realtek 2ch	Centos7.2	TDB	TDB	TDB
Any	USB Soundblaster XiFi	Centos7.2, Asus Rampage Gene IV, iCore7 3.5Ghz, 32GB 2400Mhz RAM, SSD, Quadro K2200 (nv drv 352.55)	120000	20000	TDB

Table K.2:

Some tuned configurations for 24fps on a 60Hz monitor.

25.1 K.1 Suggestions for resolving audio static issues with PulseAudio for Linux

In this Appendix, we outline some suggestions for configuring your linux distribution's pulseaudio to address the issue of audio static during playback when RV sees heavy and continuous use within the context of a production environment. While this issue is intermittent and hard to reproduce, it can occur with sufficient frequency to become a support burden. Please note the suggestions here should be validated by your systems/video engineering dept before you adopt them.

Settings for /etc/pulse/default.pa.

```
# For RHEL7 equivalent systems
#
# fragments=2 (prevents problems on kvm and teradici host)
# fixed_latency_range=1 (fixes static problem when system is heavily loaded or in swap)
#
load-module module-alsa-card device_id=PCH format=s16le rate=48000 fragments=2 fixed_
↪latency_range=1
```

NB: Many thanks to JayHillard/ChrisMihaly@WDAS from sharing these settings with us.

CHAPTER

TWENTYSIX

B - STEREO SETUP

26.1 B.1 Linux

This is taken from an NVIDIA README file. The portions pertaining to stereo modes are reproduced here:

The following driver options are supported by the NVIDIA X driver. They may be specified either in the Screen or Device sections of the X config file.

26.1.1 Option "Stereo" "integer"

Enable offering of quad-buffered stereo visuals on Quadro. Integer indicates the type of stereo glasses being used

Valu	Equipment
1	DDC glasses. The sync signal is sent to the glasses via the DDC signal to the monitor. These usually involve a passthrough cable between the monitor and video card
2	"Blueline" glasses. These usually involve a passthrough cable between the monitor and video card. The glasses know which eye to display based on the length of a blue line visible at the bottom of the screen. When in this mode, the root window dimensions are one pixel shorter in the Y dimension than requested. This mode does not work with virtual root window sizes larger than the visible root window size (desktop panning).
3	Onboard stereo support. This is usually only found on professional cards. The glasses connect via a DIN connector on the back of the video card.
4	TwinView clone mode stereo (aka "passive" stereo). On video cards that support TwinView, the left eye is displayed on the first display, and the right eye is displayed on the second display. This is normally used in conjuction with special projectors to produce 2 polarized images which are then viewed with polarized glasses. To use this stereo mode, you must also configure TwinView in clone mode with the same resolution, panning offset, and panning domains on each display.

Stereo is only available on Quadro cards. Stereo options 1, 2, and 3 (aka "active" stereo) may be used with TwinView if all modes within each metamode have identical timing values. Please see Appendix J for suggestions on making sure the modes within your metamodes are identical. The identical modeline requirement is not necessary for Stereo option 4 ("passive" stereo). Currently, stereo operation may be "quirky" on the original Quadro (NV10) chip and left-right flipping may be erratic. We are trying to resolve this issue for a future release. Default: Stereo is not enabled.

UBB must be enabled when stereo is enabled (this is the default behavior).

Stereo options 1, 2, and 3 (aka "active" stereo) are not supported on digital flat panels.

26.1.2 Option "AllowDFPStereo" "boolean"

By default, the NVIDIA X driver performs a check which turns off active stereo (stereo options 1, 2, and 3) if the X screen is driving a DFP. The "AllowDFPStereo" option bypasses this check.

ENSURING IDENTICAL MODE TIMINGS

Some functionality, such as Active Stereo with TwinView, requires control over exactly what mode timings are used. There are several ways to accomplish that:

If you only want to make sure that both display devices use the same modes, you only need to make sure that both display devices use the same HorizSync and VertRefresh values when performing mode validation; this would be done by making sure the HorizSync and SecondMonitorHorizSync match, and that the VertRefresh and the SecondMonitorVertRefresh match.

A more explicit approach is to specify the modeline you wish to use (using one of the modeline generators available), and using a unique name. For example, if you wanted to use 1024x768 at 120 Hz on each monitor in TwinView with active stereo, you might add something like: # 1024x768 @ 120.00 Hz (GTF) hsync: 98.76 kHz; pclk: 139.05 MHz Modeline "1024x768_120" 139.05 1024 1104 1216 1408 768 769 772 823 -HSync +Vsync In the monitor section of your X config file, and then in the Screen section of your X config file, specify a MetaMode like this: Option "MetaModes" "1024x768_120, 1024x768_120"

26.1.3 Support for GLX in Xinerama

This driver supports GLX when Xinerama is enabled on similar GPUs. The Xinerama extension takes multiple physical X screens (possibly spanning multiple GPUs), and binds them into one logical X screen. This allows windows to be dragged between GPUs and to span across multiple GPUs. The NVIDIA driver supports hardware accelerated OpenGL rendering across all NVIDIA GPUs when Xinerama is enabled.

To configure Xinerama: configure multiple X screens (please refer to the XF86Config(5x) or xorg.conf(5x) manpages for details). The Xinerama extension can be enabled by adding the line

Option "Xinerama" "True"

to the "ServerFlags" section of your X config file.

Requirements:

It is recommended to use identical GPUs. Some combinations of non-identical, but similar, GPUs are supported. If a GPU is incompatible with the rest of a Xinerama desktop then no OpenGL rendering will appear on the screens driven by that GPU. Rendering will still appear normally on screens connected to other supported GPUs. In this situation the X log file will include a message of the form:

(WW) NVIDIA(2): The GPU driving screen 2 is incompatible with the rest of (WW) NVIDIA(2): the GPUs composing the desktop. OpenGL rendering will (WW) NVIDIA(2): be turned off on screen 2.

The NVIDIA X driver must be used for all X screens in the server.

Only the intersection of capabilities across all GPUs will be advertised.

X configuration options that affect GLX operation (e.g.: stereo, overlays) should be set consistently across all X screens in the X server.

26.2 B.2 macOS and Windows

There are no special requirements (other than having a proper GPU that can produce stereo output). If the macOS or Windows graphical environment can provide RV with a stereo GL context, it will play back in stereo. If not, the console widget will pop up and you will see GL errors.

If you have trouble with the stereo on the Mac, you might have some luck on one of Apple's mailing lists.

CHAPTER

TWENTYSEVEN

C - THE RVLINK PROTOCOL: USING OPEN RV AS A URL HANDLER

RV can act as a protocol handler for URLs using the "rvlink" protocol. These URLs have the form:

```
rvlink://<RV commandline>
```

for example,

```
rvlink:// -l -play /path/to/my/movie.mov
```

Will start RV (or on the mac, create a new session or replace the current session), load movie.mov, turn on the look-ahead cache, and start playback.

27.1 C.1 Using rvlink URLs

You can insert rvlinks into web pages, chat sessions, emails, etc. Of course it is up to each individual application whether it recognizes the protocol. Some applications can be taught to treat anything of the form ``name://" as a link to the name protocol, but others are hard-coded to only recognize ``http://", ``ftp://", etc. Some examples of apps that will recognize rvlinks are

- Firefox
- Safari
- Chrome
- Internet Explorer
- Thunderbird
- Mac Mail
- IChat

One example of an app that will only recognize a hard-coded set of protocol types is Pidgin.

To use an rvlink in HTML, this kind of thing should work:

```
<a href="rvlink: -l -play /path/to/my/movie.mov">play movie</a>
```

Note the quotation marks.

In other settings (like pasting into an email, for example) you may want a ``web-encoded" URL, since an RV command line can contain arbitrary characters. RV will do the web-encoding for you, if you ask it to do so on the command line. For example, if you run:

```
rv -l -play /path/to/my/movie.mov -encodeURL
```

RV will print the encoded url to the terminal:

```
rvlink://%20-l%20-play%20%2Fpath%2Fto%2Fmy%2Fmovie.mov
```

Some browsers, however, like IE and Konqueror, seem to modify even encoded URLs before they get to the protocol handler. If the rvlink URL contains interesting characters, even an encoded URL will not work with these browsers. To address this issue, RV also supports *fully-baked* URLs, that look like this:

rvlink://baked/202d6c202d706c6179202f706174682f746f2f6d792f6d6f7669652e6d6f76

This form of URL has the disadvantage of being totally illegible to humans, but as a last resort, it should ensure that the URL reaches the protocol handler without interference. As with encoded URLs, baked URLs can be generated from command-lines by giving RV the "-bakeURL" command-line option. Note that the "baked" URL is just a hex-encoded version of the command line, so in addition to using RV itself, you can do it programmatically. For example, in python, something like

```
"-play /path/to/file.exr".encode("hex")
```

should do the trick

27.1.1 A Note on Spaces

In general, RV will treat spaces within the URL as delimiting arguments on the command line. If you want to include an argument with spaces (a movie name containing a space, for example) that argument must be enclosed in single quotes ('). For example, if the name of your media is "my movie.mov", the encoded rvlink URL to play it would look like:

```
rvlink://%20'my%20movie.mov'
```

27.2 C.2 Installing the Protocol Handler

RV itself is the program that handles the rvlink protocol, so all that is necessary is to register RV as the designated rvlink handler with the OS or desktop environment. This is a different process on each of the platforms that RV supports.

27.2.1 Windows

On windows the rvlink protocol needs to be added to the registry. If you are using the RV installer for Windows this will happen automatically. If not, you need to edit the "rvlink.reg" file in the "etc" directory of the install to point at the install location, then just double click on this file to edit the registry.

27.2.2 Mac

Run RV once with the ``-registerHandler" command-line option in order to register that executable as the default rvlink protocol handler (this is to prevent confusion between RV and RV64 when both are installed).

27.2.3 Linux

Unlike Windows and Mac, Linux protocols are registered at the desktop environment level, not the OS level. After you've installed RV on your machine, you can run the "rv.install_handler" script in the install's bin directory. This script will register RV with both the KDE and Gnome desktop environments.

Some application-specific notes:

Firefox may or may not respect the gnome settings, in general, I've found that if there is enough of the gnome environment installed that gconfd is running (even if you're using KDE or some other desktop env), Firefox will pick up the gnome settings. If you can't get this to work, you can register the rvlink protocol with Firefox directly.

Konqueror sadly seems to munge URLs before giving them to the protocol handler. For example by swapping upper for lowercase letters. And sometimes it does not pass them on at all. This means some rvlink URLs will work and some won't, so we recommend only "baked" rvlink urls with Konqueror at the moment.

Chrome uses the underlying system defaults to handle protocols. In most cases this means whatever "xdg-open" is configured to use. Running the rv.install_handler should be sufficient

27.3 C.3 Custom Environment Variables

Depending on the browser and desktop environment, **environment variables** set in a user environment may not be available to RV when started from a URL. If RV in your setup requires these **environment variables** (RV_SUPPORT_PATH, for example), it may have problems or not run at all when started from a URL. In order to ensure a consistent environment, you must ensure that these **environment variables** are set at a system-wide (or at least user-independent) level. On Linux, setting this up varies from distribution to distribution. You will want to research the appropriate steps for your distribution. On Windows, the usual **environment variable** techniques should work. MacOS has lately made this harder, but if you set the **environment variables** in /etc/launchd.conf (and reboot after setting), then the values should be picked up by all processes on the system.

27.4 C.4 Testing the Protocol Handler

Once RV is properly configured as your rvlink: protocol handler, copy and paste the following URL into your browser window: `rvlink://smptebars.movieproc`. This should launch RV and display a standard SMPTE colorbar image.

You can also test RV's "one-click sync" with these links (**only** on linux and windows).

- First copy and paste the following link into your browser to start your "sync target" RV (make sure no other RVs are running before you paste this link into your browser): `rvlink:// -reuse 0 -networkPort 45128 -network smptebars,start=1,end=100,fps=24.movieproc`.
- Then copy and paste the following link into your browser to sync with the previously started sync target: `rvlink:// -networkPort 45129 -network -networkConnect 127.0.0.1 45128 -flags syncPullFirst`.

As of RV version 3.8.6, a new "fully baked" URL format will be supported by RV. This form of URL will be much more resistant to munging by evil browsers. Here's an example of such a baked URL: `rvlink://baked/20736d707465626172732e6d6f76696570726f63202d6576616c20277072696e74282532326e6f2070726f626c656d2535436e2`

27.5 C.5 One-click sync

The rvlink protocol allows you to build URLs that start and run RV. You can also set up a network connection to a remote RV from the command line and hence from an rvlink URL. Furthermore, you can direct that after the network connection is established, Sync should be activated, and the remote session information should be pulled over the network so that you'll both be looking at the same media.

To create such a link, use **Edit > Copy Sync Session URL**. It copies the link to the clipboard so you can share it with others.

Imagine this scenario: You're using RV, have some media loaded, when you decide you want your friend to look at it with you. They're in the next building, so rather than walk over there you want to start a RV Sync session with them.

To get connected, you can send them by email the URL from **Copy Sync Session URL**.

So they can click the link, or copy paste it in a browser, to open in RV the linked session. When they click on this link, RV will start on their machine, connect to your running RV, pull the session information (so your media will match), and start Sync. So there you have it: One-Click Sync!

CHAPTER

TWENTYEIGHT

D - USING OPEN RV AS NUKE'S FLIP BOOK PLAYER

Nuke can easily be configured to use RV as a flipbook. Nuke can also be set up to render out OpenEXR temp files instead of the default rgb files.

28.1 D.1 Setting up a custom plugins area for Nuke

In order to configure Nuke to work with RV, you should set up a custom scripts/plugins directory where you can add new custom Nuke functionality without disturbing the default Nuke install. This directory can be anywhere on the NUKE_PATH environment variable, but note that the ``$HOME/.nuke" directory is always on that NUKE_PATH, so we'll assume we're working there for now.

1. In your $HOME/.nuke directory, create a file called ``init.py", if it doesn't already exist, and also one called ``menu.py".
2. Add this line to the init.py file

```
nuke.pluginAddPath('./python')
```

3. Create the directory referenced by the line we just added

```
mkdir $HOME/.nuke/python
```

Done! Now Nuke well pick up new stuff from this area on start-up

28.2 D.2 Adding Open RV support in the custom plugins area

The following assumes that you've setup a custom plugin area as described in the previous section.

1. Download the script ``rv_this.py" from *this forum post*.
2. Move rv_this.py into $HOME/.nuke/python.
3. Add the following line to the $HOME/.nuke/init.py file, **after** the pluginAddPath lines mentioned above:

```
import rv_this
```

4. Add the following to the $HOME/.nuke/menu.py file:

```
menubar = nuke.menu("Nuke");
menubar.addCommand("File/Flipbook Selected in &RV",
        "nukescripts.flipbook(rv_this.rv_this, nuke.selectedNode())", "#R")
```

RV will now appear in the render menu and will be available with the hot key Alt+r.

To add an option to render your flipbook images in Open EXR,

1. Find the file ``nukescripts/flip.py" in your Nuke install and copy it to $HOME/.nuke/python/flipEXR.py.
2. Edit $HOME/.nuke/python/flipEXR.py and find the two lines that specify the output file name (search for ``nuke_tmp_flip") and change the file extension from ``rgb" to ``exr".
3. Add the following line to the $HOME/.nuke/menu.py file:

```
menubar.addCommand("File/Flipbook Selected in &RV (EXR)",
        "flipExr.flipbook(rv_this.rv_this, nuke.selectedNode())", "#E")
```

Now you'll have the option in the File menu to flipbook to EXR, with the hotkey Alt+e.

You can edit the rv_this.py script to specify any rv options you wish to set as your viewing defaults. For example you could un-comment the script lines that will apply -gamma 2.2 or enable -sRGB, or you could specify a file or display LUT for RV to use.

CHAPTER

TWENTYNINE

E - OPEN RV AUDIO ON LINUX

29.1 E.1 Overview

RV provides multiple audio modules so users can get the most audio functionality available given the constraints imposed by the Linux kernel version, the available audio frameworks, and the other audio applications in use.

Potential user issues with audio on Linux are choppy playback (clicking, dead spots, drop-outs), or high latency (it takes a breath's worth of time to start playing). Both of these issues are extremely annoying and it is even worse when they both occur. Ideally latency is nearly 0 for high quality audio and drop-outs never occur.

To make matters worse, there are two completely different audio drivers for Linux: OSS and ALSA. Only one of these can be present at a time (but both provide a ``compatibility" API for the other). ALSA is the official Linux audio API and is shipped with almost all distributions. OSS is a cross platform API (for UNIX based machines).

The Linux desktop projects, KDE and Gnome, both have sound servers build on top of OSS/ALSA. These are called esd and arts. Both of them have become deprecated in recent releases. If either of these processes is running you may find it difficult to use audio with RV (or other commercial products).

Some distributions use a newer desktop sound server called PulseAudio . PulseAudio ships with most distributions and is on by default in RHEL, CentOS, and Fedora, and comes with Debian and Ubuntu. RV can use PulseAudio through the ALSA (Safe) module, however we strongly recommend against using PulseAudio. Though the PulseAudio server is designed to allow multiple applications to simultaneously play audio (and can run in real-time mode), we have found a decrease in stability with each new release of the module. Even for simple read-only API calls that query the state of the device. If possible for your workflow we suggest you remove PulseAudio, if it is not otherwise necessary.

29.1.1 E.1.1 How Open RV Handles Linux Audio

RV has two ALSA audio back ends for Linux: the old and safe versions. The old version is meant to run on distributions which shipped with ALSA versions 1.0.13 or earlier and which do not have PulseAudio installed. The safe version should work well with PulseAudio systems. Unless you know that your system is using PulseAudio you should try the ALSA Old back end to start with.

On some systems, RV will fail to load one or more of its audio modules. This can occur of the API used by the module is newer than the one installed on the system. For example neither CentOS 4.6 or Fedora Core 4 can load the ALSA safe module unless the ALSA client library is upgraded. Any Linux system which uses ALSA 1.0 or newer should function with the ALSA old back end.

29.2 E.2 ALSA (Pre-1.0.14)

The ALSA (Old) audio back uses a subset of the ALSA API. Without modification, it will make hardware devices directly accessible. (In ALSA terms these are the hw:x,y devices). You can also add to the list of devices using an environment variable called RV_ALSA_EXTRA_DEVICES. The syntax is:

```
visiblename1@alsadevicename1|visiblename2@alsadevicename2|...
```

So for example to add the "dmix" device, set the variable to: dmix@dmix.

See the ALSA documentation for the .asoundrc file for more information about devices and how they can be created. Devices defined in the configuration files will not automatically show up in RV; you'll need to add them to the environment variable.

When using the hardware devices other programs will not have access to the audio.

29.3 E.3 ALSA (Safe)

The term "Safe" here refers to the subset of the ALSA API deemed safe to use in the presence of PulseAudio. Programs that use features that PulseAudio cannot intercept will cause inconsistent audio availability or unstable output. Use of hardware devices in the presence of PulseAudio can prevent other applications from using the audio driver whether they be ALSA or OSS based.

The ALSA (Safe) audio back end uses a more modern version of the ALSA client API. If the version of ALSA client library on your system is newer than 1.0.13 this back end might work better on your system. PulseAudio system should definitely be using this audio module.

The ALSA safe back end can use the RV_ALSA_EXTRA_DEVICES variable (see above). However, user/system defined devices will typically show up in the device list automatically.

The safe back end does not give you direct access to the hardware devices by default. Typically you will see devices which look like:

The xxxx and n values correspond to the hardware device values used in the ALSA old back end. You can force the use of the hardware devices by adding them to the environment variable.

```
front:CARD=xxxx,DEV=n
```

For example, the first device on the first card would be hw:0,0 in ALSA parlance. So setting the value of RV_ALSA_EXTRA_DEVICES to:

First Hardware Device@hw:0,0

would add that to the list.

We recommend that you do not add hardware devices when using the safe ALSA module. The hardware devices will typically be accessible via the default device list under different names (like front:CARD=xxxx,DEV=n from above). Unlike the default list of devices, the hardware devices will shut out other software from using the audio even on systems using the PulseAudio sound server.

29.4 E.4 Platform Audio

RV supports a cross platform audio module based on Qt audio (which on Linux is ALSA based). Note Platform Audio supports playback on multichannel audio devices.

CHAPTER

THIRTY

F - TROUBLESHOOTING NETWORKING

30.1 Configure RV's network settings

You can use the Network dialog under the RV Menu to configure RV's network syncing (as in between two or more instances of RV), set the name you show up as in sync mode on other RV's, and establish the port you want to listen on for network connections. From this dialog you can initiate a connection to another and manage your Contacts list.

The Contacts tab gives a list of users you have established connections with previously, but what is more important is the permission drop-down menu immediately below the contacts list. This drop-down allows you to set the default behavior for how to manage incoming connections. Once a user is added to the contacts list, the permission can be set per contact. Lastly, the Connections tab shows your active connections.

At the bottom are two important buttons: "Connect…" and "Start Network." "Connect…" lets you type in another RV's network name and port. RV can only connect by hostname or IP address. It is important that the RV reaching out to connect can see the remote RV through any firewall. You may have to setup port forwarding or some other DMZ configuration for this to work. Please contact your network administrator for these types of advanced setups.

Once you are satisfied with your settings, you can enable networking for RV by pressing the "Start Network" button.

30.2 Connections only work from one direction or are always refused

Some operating systems have a firewall on by default that may be blocking the port RV is trying to use. When you start RV on the machine with the firewall and start networking it appears to be functioning correctly, but no one can connect to it. Check to see if the port that RV wants to use is available through the firewall.

This is almost certainly the case when the connection works from one direction but not the other. The side which can make the connection is usually the one that has the firewall blocking RV (it won't let other machines in).

G - RISING SUN RESEARCH CINESPACE .CSP FILE FORMAT

This is RSR's spec of their file format:

The cineSpace LUT format contains three main sections.

31.1 Header

This section contains the LUT identifier and the LUT type, 3D or 1D.

It is made up of the first two (2) valid lines in the file. See Notes below for a definition of a valid line.

A 3D LUT:

```
CSPLUTV100
3D
```

or a 1D (channel) LUT:

```
CSPLUTV100
1D
```

31.2 Metadata

This section contains metadata not defined by the CSP format itself. RV will ignore everything here except a line containing a conditioningGamma. For example:

```
BEGIN METADATA
conditioningGamma=0.4545
END METADATA
```

The conditioningGamma value is applied (in GLSL) to both the input lattice points and the input data to the LUT (so does not change the mathematical "meaning" of the LUT). Especially for LUTs expecting linear input, this can compensate for the fact that, when the LUT is applied in hardware the even spacing of the lattice points can cause artifacts. In particular, we've found that a conditioningGamma value of 0.4545 allows a ACES-to-ACESlog LUT applied in RV (in HW) to match the results of applying the same LUT in Nuke (in SW).

31.3 1D preLUT data

This section is designed to allow for unevenly spaced data and also to accommodate input data that maybe outside the 0.0 <-> 1.0 range. Each primary channel, red, green and blue has each own 3 line entry. The first line is the number of preLUT data entries for that channel. The second line is the input and the third line is the mapped output that will then become the input for the LUT data section.

It is made up of the valid lines 3 to 11 in the LUT. See Notes below for a definition of a valid line.

Examples …

Map extended input (max. 4.0) into top 10% of LUT

```
11
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 4.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
11
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 4.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
11
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 4.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Access LUT data via a gamma lookup Red channel has gamma 2.0 Green channel has gamma 3.0 but also has fewer points Blue channel has gamma 2.0 but also has fewer points

```
11
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
0.0 0.01 0.04 0.09 0.16 0.25 0.36 0.49 0.64 0.81 1.0
6
0.0 0.2 0.4 0.6 0.8 1.0
0.0 0.008 0.064 0.216 0.512 1.0
6
0.0 0.2 0.4 0.6 0.8 1.0
0.0 0.04 0.16 0.36 0.64 1.0
```

31.4 LUT data

This section contains the LUT data. The input stimuli for the LUT data is evenly spaced and normalized between 0.0 and 1.0. All data entries are space delimited floats. For 3D LUTs the data is red fastest.

It is made up of the valid lines 12 and onwards in the LUT. See Notes below for a definition of a valid line.

Examples …

Linear LUT with cube sides R,G,B = 2,3,4 (ie. a 2x3x4 data set)

```
2 3 4
0.0 0.0 0.0
1.0 0.0 0.0
0.0 0.5 0.0
1.0 0.5 0.0
0.0 1.0 0.0
1.0 1.0 0.0
```

(continues on next page)

(continued from previous page)

```
0.0 0.0 0.33
1.0 0.0 0.33
0.0 0.5 0.33
1.0 0.5 0.33
0.0 1.0 0.33
1.0 1.0 0.33
0.0 0.0 0.66
1.0 0.0 0.66
0.0 0.5 0.66
1.0 0.5 0.66
0.0 1.0 0.66
1.0 1.0 0.66
0.0 0.0 1.0
1.0 0.0 1.0
0.0 0.5 1.0
1.0 0.5 1.0
0.0 1.0 1.0
1.0 1.0 1.0
```

31.5 Notes:

All lines starting with white space are considered not valid and are ignored. Lines can be escaped to the next line with "\" (but note that RV does not actually implement this part of the spec, so lines should not be broken in CSP files for RV). All values on a single line are space delimited.

The first line must contain the LUT type and version identifier "CSPLUTV100"

The second line must contain either "3D" or "1D".

The third valid line (after any METADATA section) is the number of entries in the red 1D preLUT. It is an integer.

The fourth valid line contains the input entries for the red 1D preLUT. These are floats and the range is not limited. The number of entries must be equal to the value on the third valid line.

The fifth valid line contains the output entries for the red 1D preLUT. These are floats and the range is limited to 0.0 <-> 1.0. The number of entries must be equal to the value on the third valid line.

The sixth valid line is the number of entries in the green 1D preLUT. It is an integer.

The seventh valid line contains the input entries for the green 1D preLUT. These are floats and the range is not limited. The number of entries must be equal to the value on the sixth valid line.

The eighth valid line contains the output entries for the green 1D preLUT. These are floats and the range is limited to 0.0 <-> 1.0. The number of entries must be equal to the value on the sixth valid line.

The ninth valid line is the number of entries in the blue 1D preLUT. It is an integer.

The tenth valid line contains the input entries for the blue 1D preLUT. These are floats and the range is not limited. The number of entries must be equal to the value on the ninth valid line.

The eleventh valid line contains the output entries for the blue 1D preLUT. These are floats and the range is limited to 0.0 <-> 1.0. The number of entries must be equal to the value on the ninth valid line.

The twelfth valid line in a "3D" LUT contains the axis lengths of the 3D data cube in R G B order.

The twelfth valid line in a "1D" LUT contains the 1D LUT length

The thirteenth valid line and onwards contain the LUT data. For 3D LUTs the order is red fastest. The data are floats and are not range limited. The data is evenly spaced.

The LUT file should be named with the extension .csp

CHAPTER

THIRTYTWO

H - CRASH REPORTING

RV includes google's crash reporting mechanism called breakpad on Linux. If RV crashes it will produce a file in /tmp with an extension of .dmp.

CHAPTER

THIRTYTHREE

I - PYSIDE EXAMPLE USAGE

RV ships with PySide on all platforms. In this section, we present two simple examples of PySide usage. We also demonstrate how to access the RV session window in the second example.

The first example, shown below, is a simple executable python/pyside file that uses RV py-interp.

```
#!/Applications/RV64.app/Contents/MacOS/py-interp

# Import PySide classes
import sys
from PySide.QtCore import *
from PySide.QtGui import *

# Create a Qt application.
# IMPORTANT: RV's py-interp contains an instance of QApplication;
# so always check if an instance already exists.
app = QApplication.instance()
if app == None:
    app = QApplication(sys.argv)

# Display the file path of the app.
print app.applicationFilePath()

# Create a Label and show it.
label = QLabel("Using RV's PySide")
label.show()

# Enter Qt application main loop.
app.exec_()

sys.exit()
```

The second example, shown below, is a RV python package that uses pyside for building its UI with Qt widgets that control properties values on an RVLensWarp node. Note too that in this example, the current RV session QMainWindow obtained from rv.qtutils.sessionWindow() and we use it to change the session window's opacity with the "Enable" checkbox.

This "PySide Example" can be loaded in RV through Preferences->Packages.

```
from PySide.QtCore import QFile
from PySide.QtGui import QDoubleSpinBox, QDial, QCheckBox
from PySide.QtUiTools import QUiLoader

```

(continues on next page)

(continued from previous page)

```
import types
import os
import math

import rv
import rv.qtutils

import pyside_example # need to get at the module itself


class PySideDockTest(rv.rvtypes.MinorMode):
    "A python mode example that uses PySide"

    def checkBoxPressed(self, checkbox, prop):
        def F():
            try:
                if checkbox.isChecked():
                    if self.rvSessionQObject is not None:
                        self.rvSessionQObject.setWindowOpacity(1.0)
                    rv.commands.setIntProperty(prop, [1], True)
                else:
                    if self.rvSessionQObject is not None:
                        self.rvSessionQObject.setWindowOpacity(0.5)
                    rv.commands.setIntProperty(prop, [0], True)
            except:
                pass
        return F


    def dialChanged(self, index, last, spins, prop):
        def F(value):
            diff = float(value - last[index])
            if diff < -180:
                diff = value - last[index] + 360
            elif diff > 180:
                diff = value - last[index] - 360
            diff /= 360.0
            last[index] = float(value)
            try:
                p = rv.commands.getFloatProperty(prop, 0, 1231231)
                p[0] += diff
                if p[0] > spins[index].maximum() :
                    p[0] = spins[index].maximum()
                if p[0] <  spins[index].minimum()  :
                    p[0] = spins[index].minimum()
                spins[index].setValue(p[0])
                rv.commands.setFloatProperty(prop, p, True)
            except:
                pass
        return F

    def spinChanged(self, index, spins, prop):
```

(continues on next page)

(continued from previous page)

```
        def F(value):
            try:
                rv.commands.setFloatProperty(prop, [p], True)
            except:
                pass

        def F():
            try:
                p = spins[index].value()
                commands.setFloatProperty(prop, [p], True)
            except:
                pass

        return F

    def findSet(self, typeObj, names):
        array = []
        for n in names:
            array.append(self.dialog.findChild(typeObj, n))
            if array[-1] == None:
                print "Can't find", n
        return array

    def hookup(self, checkbox, spins, dials, prop, last):
        checkbox.released.connect(self.checkBoxPressed(checkbox, "%s.node.active"%prop))
        for i in range(0,3):
            dial = dials[i]
            spin = spins[i]
            propName = "%s.warp.k%d" % (prop,i+1)
            dial.valueChanged.connect(self.dialChanged(i, last, spins, propName))
            spin.valueChanged.connect(self.spinChanged(i, spins, propName))
            last[i] = dial.value()


    def __init__(self):
        rv.rvtypes.MinorMode.__init__(self)
        self.init("pyside_example", None, None)

        self.loader = QUiLoader()
        uifile = QFile(os.path.join(self.supportPath(pyside_example, "pyside_example"),
↪"control.ui"))
        uifile.open(QFile.ReadOnly)
        self.dialog = self.loader.load(uifile)
        uifile.close()

        self.enableCheckBox  = self.dialog.findChild(QCheckBox, "enableCheckBox")

        #
        # To retrieve the current RV session window and
        # use it as a Qt QMainWindow, we do the following:
        self.rvSessionQObject = rv.qtutils.sessionWindow()
```

(continues on next page)

(continued from previous page)

```
        # have to hold refs here so they don't get deleted
        self.radialDistortDials = self.findSet(QDial, ["k1Dial", "k2Dial", "k3Dial"])

        self.radialDistortSpins = self.findSet(QDoubleSpinBox, ["k1SpinBox", "k2SpinBox",
↪ "k3SpinBox"])

        self.lastRadialDistort = [0,0,0]

        self.hookup(self.enableCheckBox, self.radialDistortSpins, self.
↪radialDistortDials, "#RVLensWarp", self.lastRadialDistort)


    def activate(self):
        rv.rvtypes.MinorMode.activate(self)
        self.dialog.show()

    def deactivate(self):
        rv.rvtypes.MinorMode.deactivate(self)
        self.dialog.hide()

def createMode():
    "Required to initialize the module. RV will call this function to create your mode."
    return PySideDockTest()
```

CHAPTER

THIRTYFOUR

J - SUPPORTED MULTICHANNEL AUDIO LAYOUTS

Multichannel audio devices are supported by RV audio output module choice “Platform Audio”. The “Platform Audio” choice is available on all RV platforms ie. OSX, Linux and Windows.

On OSX, you might need to enable the multichannel (e.g. 5.1) capability of your audio device using the OSX utility “Audio Midi Setup”.

The list of possible channel layouts that RV recognises is listed in the table below.

Table J.1:

Channel Layout	Layout and Speaker Description FL = Front Left FR = Front Right FC = Front Center LF = Lower Frequency/Subwoofer BL = Back Left BR = Back Right SL = Side Left SR = Side Right BC = Back Center FLC = Front Left of Center FRC = Front Right of Center LH = Left Height RH = Right Height
Mono	FC
Stereo	FL:FR
3.1	FL:FR:LF
Quadrophonic	FL:FR:BL:BR
5.1	FL:FR:FC:FL:SL:SR
5.1 (Back)	FL:FR:FC:FL:BL:BR
5.1 (Swap)	FL:FR:BL:BR:FC:FL
5.1 (AC3)	FL:FC:FR:SL:SR:LF
5.1 (DTS)	FC:FL:FR:SL:SR:LF
5.1 (AIFF)	FL:BL:FC:FR:BR:LF
6.1	FL:FR:FC:LF:BL:BR:BC
7.1 (SDDS	FL:FR:FC:LF:SL:SR:FLC:FRC
7.1	FL:FR:FC:LF:SL:SR:BL:BR
7.1 (Back)	FL:FR:FC:LF:BL:BR:SL:SR
9.1	FL:FR:FC:LF:BL:BR:SL:SR:LH:RH
16	

Supported Multichannel Layouts

Note that RV will mix down, mix up or reorder channels for any given media to match the intended output device channel layout format.

For example, playing back 5.1 media to a stereo audio device will see the 5.1 audio channels mixed downed to two channels.

Similarly, playing back stereo media to a 5.1 device will see the media's stereo FL and FR content mixed up to the 5.1 device's FL, FR and FC only.

For the case where the media and device have the same channel count and speaker types but different layout e.g. for 5.1 media and 5.1 (AC3) device, the media's channel layout is reordered to match the device channel layout when RV reads the media.

For the case where the media and device have the same channel count but non-matching channel/speaker types, the channel layout of media is passed to the device as is; for example 5.1 (Back) media and 5.1 device.

The audio channel layout for any given media can be determined from RV's image info tool.

CHAPTER

THIRTYFIVE

K - LOCALIZING MEDIA PATHS WITH RV_OS_PATH OR RV_PATHSWAP

35.1 RV_OS_PATH Variables

RV_OS_PATH variables work similarly to **RV_PATHSWAP** variables, but they can be used to adjust paths of media files (and LUTs etc) that are more arbitrary, and are specifically targeted at supporting multi-OS environments.

For example, if your site supports Linux/Windows/OSX machines, and the same basic file system structure appears on all OSs but at different roots, setting a group of three environment variables on all machines will allow some interoperability. Suppose you have the following three "roots" under which your directory structure is identical on all OSs:

OS	Root
OSX	/shows
LINUX	/net/shows
WINDOWS	c:/shows

Then if you can ensure that all users have the following environment variables set, paths will automatically be converted on input to RV:

Variable Name	Variable Value
RV_OS_PATH_OSX	/shows
RV_OS_PATH_LINUX	/net/shows
RV_OS_PATH_WINDOWS	c:/shows

For example if RV is running on Windows and receives a path like "/net/shows/sw4/trailer.mov" it will convert it to "c:/shows/sw4/tralier.mov" before use. (**Note:** UNC paths are also supported).

If some of your production data appears in a separate hierachy, you can add additional variables to support exceptions. For example, suppose that all show data is stored as above, but reference data is stored separately under these "roots":

OS	Root
OSX	/ref
LINUX	/net/reference
WINDOWS	c:/global/reference

In that case, you can set another triplet of environment variables to allow for that exception:

Variable Name	Variable Value
RV_OS_PATH_OSX_REF	/ref
RV_OS_PATH_LINUX_REF	/net/reference
RV_OS_PATH_WINDOWS_REF	c:/global/reference

Some additional details about RV_OS_PATH variables:

- If you only use two different OSs, you only need to specify the corresponding pairs of environment variables.
- As above, additional sets of environment variables are considered to refer to the same "root" if the portion following the OS name matches ("REF" in the above example).
- You can set any number of sets of environment variables.
- RV_OS_PATH variables affect all incoming filenames **except** those which contain RV_PATHSWAP variables.
- RV_OS_PATH variables do not affect outgoing filenames.
- If more than one match is found, the variable that matches the largest number of characters in the incoming path will be used.

  Note: If you need more dynamic control over your path remapping, you can author an RV package to handle transforming your paths with the ' *incoming-source-path* '.

  Note: Due to how environments propagate, it is highly recommended to restart your computer after defining an environment variable on your system.

35.2 Open RV PATHSWAP Variables

If you're comfortable with environment variables, **RV_PATHSWAP** variables can provide a way to share session files across platforms and/or studio locations.

Note: Due to the difficult nature involving changing file paths, we do not recommend RV_PATHSWAP variables unless absolutely necessary.

Suppose that you're working on a project called 'myshow' in two locations, but with shared or mirrored data. Location 'Win' is windows-based and at that location all the media lives in paths that have names that start with `\\projects\myshow`. The other location, 'Lin', is linux-based and at that site all the media for myshow appears in paths that start with '/shows/myshow'.

To localize the media you can define a site-wide environment variable in each location called RV_PATHSWAP_MYSHOW, but with the corresponding value for that site:

At location 'Lin',

```
RV_PATHSWAP_MYSHOW = "/shows/myshow"
```

And at location 'Win',

```
RV_PATHSWAP_MYSHOW = "//projects/myshow"
```

(Note the forward slashes in the above windows path. The PATHSWAP variables operate on internal RV paths, and at this point backward slashes have been converted to forward slashes.)

You can have any number of these variables (they just all need to start with "RV_PATHSWAP_" so you could have one per show, for example. But note that if the above path pattern holds for all your projects, you can localize them all at once with variables like this one:

```
RV_PATHSWAP_ROOT = "/shows"
```

Once RV is running in an environment with these variables, it'll look for them in incoming paths (in session file, on the command line, in rvlink URLs, etc), and add them to paths it writes into session files (and network packets between synchronized RVs).

The up-shot is that a session file written at either site can be read at either site, and a sync session between sites can refer to the same media with the appropriate path for that site.

And of course these same benefits apply to using RV at a single site, but on several different platforms.

35.3 Hand-written Session Files or RVLINK URLs

If you're writing your own session files to feed to RV "by hand", note that the format RV expects is something like this:

```
string movie = "${RV_PATHSWAP_MYSHOW}/myseq/myshot/mymov.mov"
```

Similarly, a URL to play that meda could look like:

```
rvlink://${RV_PATHSWAP_MYSHOW}/myseq/myshot/mymov.mov
```

35.4 Remote Sync

RV automatically swaps the values of appropriate PATHSWAP variables in and out of the names of media transmitted across a sync connection, so once you have them set up, these variables can also make Remote Sync seamless across sites or platforms.

CHAPTER

THIRTYSIX

CHAPTER 1 - OVERVIEW

RV comes with the source code to its user interface. The code is written in a language called Mu which is not difficult to learn if you know Python, MEL, or most other computer languages used for computer graphics. RV can use Python in a nearly interchangeable manner. If you are completely unfamiliar with programming, you may still glean information about how to customize RV in this manual; but the more complex tasks like creating a special overlay or slate for RVIO or adding a new heads-up widget to RV might be difficult to understand without help from someone more experienced.This manual does not assume you know Mu to start with, so you can dive right in. For Python, some assumptions are made. The chapters are organized with specific tasks in mind.The reference chapters contain detailed information about various internals that you can modify from the UI.Using the RV file format (.rv) is detailed in Chapter *6*.

36.1 1.1 The Big Picture

RV is two different pieces of software: the core (written in C++) and the interface (written in Mu and Python). The core handles the following things:

- Image, Movie, and Audio I/O
- Caching Images and Audio
- Tracking Dependencies Among Image and Audio Operations
- Basic Image Processing in Software
- Rendering Images
- Feeding Audio to Audio Output Devices

The interface — which is available to be modified — is concerned with the following:

- Handling User Events
- Rendering Additional Information and Heads-up Widgets
- Setting and Getting State in the Image Processing Graph
- Interfacing to the Environment
- Handling User Defined Setup of Incoming Movies/Images/Audio
- High Level Features

RVIO shares almost everything with RV including the UI code (if you want it to). However it will not launch a GUI so its UI is normally non-existent. RVIO does have additional hooks for modification at the user level: overlays and leaders. Overlays are Mu scripts which allow you to render additional visual information on top of rendered images before RVIO writes them out. Leaders are scripts which generate frames from scratch (there is nothing rendered under them) and are mainly there to generate customized flexible slates automatically.

36.2 1.2 Drawing

In RV's user interface code or RVIO's leader and overlays it's possible draw on top of rendered frames. This is done using the industry standard API OpenGL. There are Mu modules which implement OpenGL 1.1 functions including the GLU library. In addition, there is a module which makes it easy to render true type fonts as textures (so you can scale, rotate, and composite characters as images). For Python there is PyOpenGL and related modules.Mu has a number of OpenGL friendly data types which include native support for 2D and 3D vectors and dependently typed matrices (e.g., float[4,4], float[3,3], float[4,3], etc). The Mu GL modules take the native types as input and return them from functions, but you can use normal GL documentation and man pages when programming Mu GL. In this manual, we assume you are already familiar with OpenGL. There are many resources available to learn it in a number of different programming languages. Any of those will suffice to understand it.

36.3 1.3 Menus

The menu bar in an RV session window is completely controlled (and created) by the UI. There are a number of ways you can add menus or override and replace the existing menu structure.Adding one or more custom menus to RV is a common customization. This manual contains examples of varying complexity to show how to do this. It is possible to create static menus (pre-defined with a known set of menu items) or dynamic menus (menus that are populated when RV is initialized based on external information, like environment variables).

CHAPTER

THIRTYSEVEN

CHAPTER 2 - IMAGE PROCESSING GRAPH

The UI needs to communicate with the core part of RV. This is done in two ways: by calling special command functions (commands) which act directly on the core (e.g. play() causes it to start playing), or by setting variables in the underlying image processing graph which control how images will be rendered.Inside each session there is a *directed acyclic graph* (DAG) which determines how images and audio will be evaluated for display. The DAG is composed of *nodes* which are themselves collections of *properties* .A node is something that produces images and/or audio as output from images and audio inputs (or no inputs in some cases). An example from RV is the *color* node; the color node takes images as input and produces images that are copies of the input images with the hue, saturation, exposure, and contrast potentially changed.A *property* is a state variable. The node's properties as a whole determine how the node will change its inputs to produce its outputs. You can think of a node's properties as *parameters* that change its behavior.RV's session file format (.rv file) stores all of the nodes associated with a session including each node's properties. So the DAG contains the complete state of an RV session. When you load an .rv file into RV, you create a new DAG based on the contents of the file. Therefore, *to change anything in RV that affects how an image looks, you must change a property in some node in its DAG* .There are a few commands which RV provides to get and set properties: these are available in both Mu and Python.Finally, there is one last thing to know about properties: they are arrays of values. So a property may contain zero values (it's empty) or one value or an array of values. The get and set functions above all deal with arrays of numbers even when a property only has a single value.*16* lists all properties and their function for each node type.

37.1 2.1 Top-Level Node Graph

When RV is started with e.g. two media (movies, file sequences) it will create two top-level group nodes: one for each media source. These are called RVSourceGroup nodes. In addition, there are four other top-level group nodes created and one display group node for each output device present on the system.

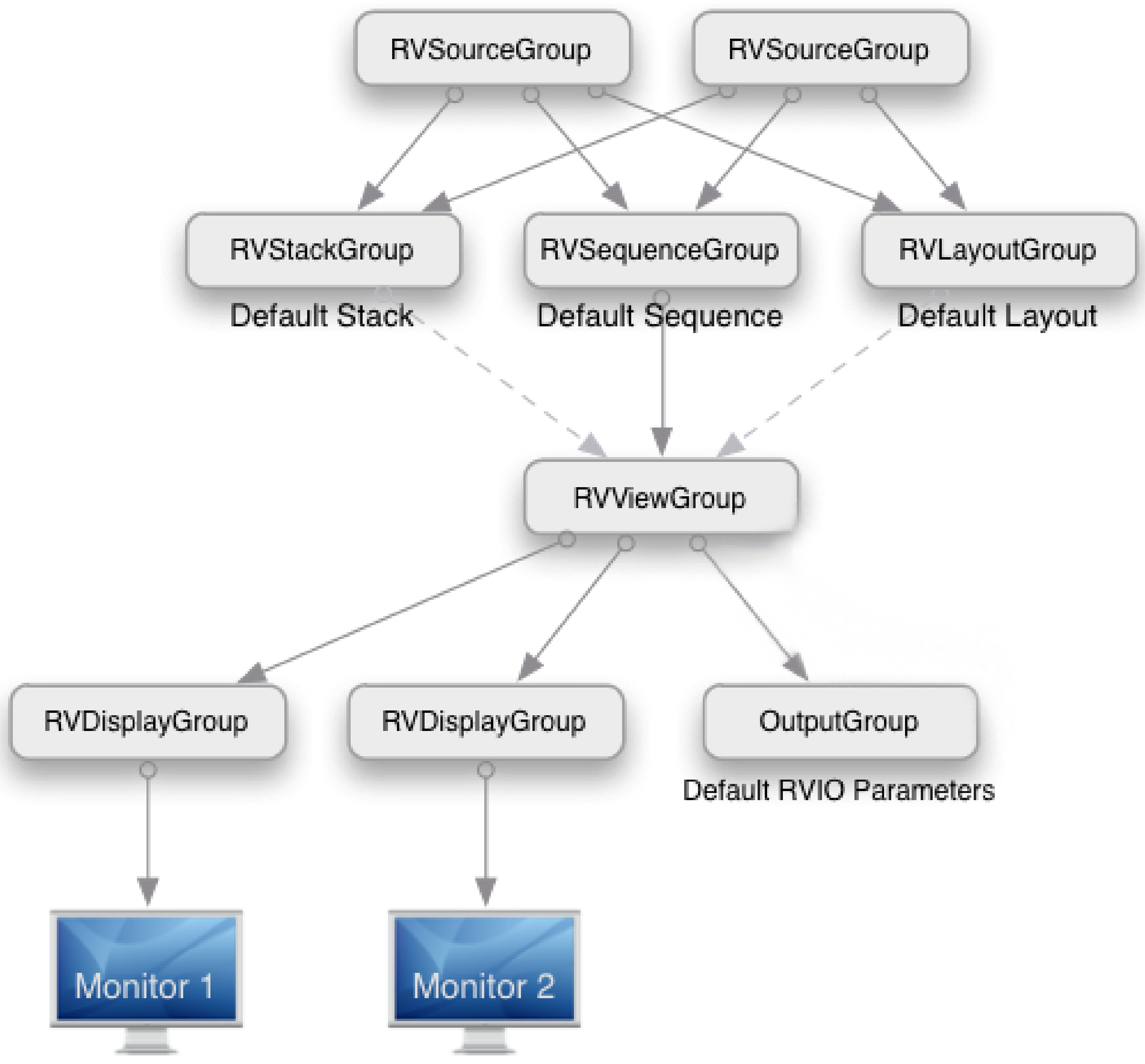


Figure 2.1:Top-Level node graph when two sources are present.

There is always a default layout (RVLayoutGroup), sequence (RVSequenceGroup), and stack node (RVStackGroup) as well as a view group node (RVViewGroup). The view group is connected to each of the active display groups (RVDisplayGroup). There is only one input to the view group and that input determines what the user is seeing in the image viewer. When the user changes views, the view group input is switch to the node the user wishes to see. For example, when the user looks at one of the sources (not in a sequence) the view group will be connected directly to that source group.New top level nodes can be created by the user. These nodes can have inputs of any other top level node in the session other than the view group and display groups.In the scripting languages, the nodes are referred to by internal name. The internal name is not normally visible to the user, but is used extensively in the session file. Most of the node graph commands use internal node names or the name of the node's type.

Command	Mu Return Type	Python Return Type	Description
nodes()	string[]	Unicode String List	Returns an array of all nodes in the graph.
nodesOfType (string *typename*)	string[]	Unicode String List	Returns all nodes in the graph of the specified type
nodeTypes()	string[]	Unicode String List	Returns an array of all node types known to the application
nodeType (string *nodename*)	string	Unicode String	Returns the type of the node specified by *nodename*
deleteNode (string *nodename*)	void	None	Deletes the node specified by *nodename*
setViewNode (string *nodename*)	void	None	Connect the specified node to the view group
newNode (string *typename* , string *nodename* = nil)	string	Unicode String	Create a node of type *typename* with name *nodename* or if *nodename* is nil use a default name
nodeConnections (string *nodename* , bool *traverseGroups* = false)	(string[	Tuple of two Unicode String Lists	Return a tuple of nodes connected to the specified node: the first as an array of input node names, the second as an array of output node names.
nodesExists (string *nodename*)	bool	Bool	Returns true if the specified node exists false otherwise
setNodeInputs (string *nodename* , string[] *inputNodes*)	void	None	Connect a nodes inputs to an array of node names
testNodeInputs (string *nodename* , string[] *inputNodes*)	string	Unicode String	Test the validity of a set of input nodes for *nodename* . If nil is returned than the inputs are valid. If a string is returned than the inputs are not valid and the string contains a user readable reason why they are not.

Table 2.1:Commands used to manage nodes in the graph

37.1.1 2.2 Group Nodes and Pipeline Groups

A group node is composed of multiple member nodes. The graph connectivity is determined by the value of the group node's properties or it is fixed. Group nodes can contain other group nodes. The member nodes are visible to the user interface scripting languages and their node names are unique in the graph. Nodes may only be connected to nodes that are members of the same group. In the case of top level nodes they can be connected to other top level nodes.

A pipeline group is a type of group node that connects it members into a single pipeline (no branches). Every pipeline group has a string array property called pipeline.nodes which determines the types of the nodes in the pipeline and the order in which they are connected. Any node type other than view and display group nodes can be specified in the pipeline.nodes property.Each type of pipeline group has a default pipeline. Except for the RVLinearizePipelineGroup which has two nodes in its default pipeline all others have a single node with the exception of the view pipeline which is empty by default. By modifying the pipeline.nodes property in any of these pipeline groups the default member nodes can either be swapped out, removed completely, or additional nodes can be inserted.For example. the following python code will set the view pipeline to use a user defined node called "FilmLook":

```
setStringProperty("#RVViewPipelineGroup.pipeline.nodes", ["FilmLook"], True)
```

37.1.2 2.3 Source Group Node

The source group node (RVSourceGroup) has fixed set of nodes and three pipeline groups which can be modified to customize the source color management.

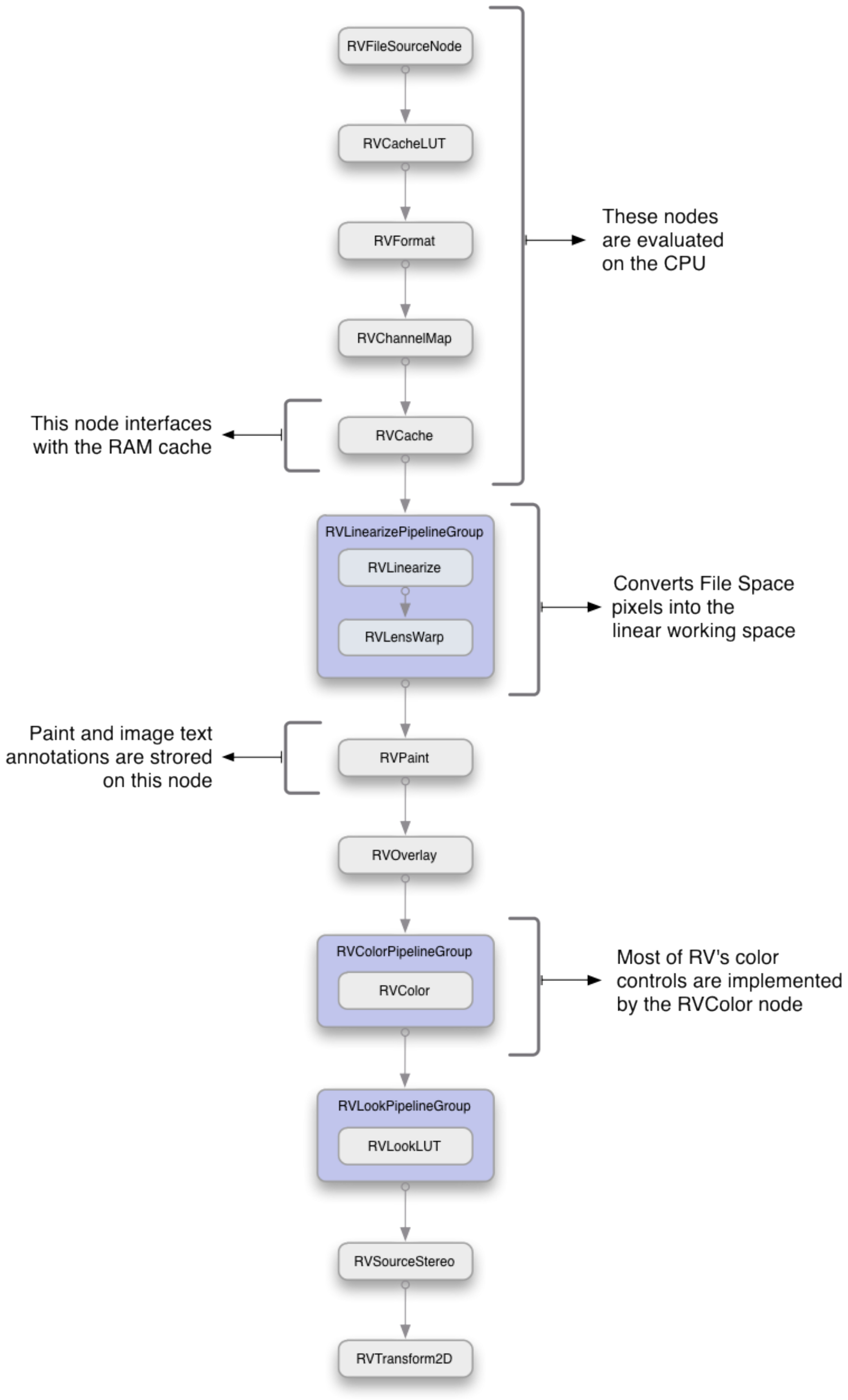

RVFileSourceNode
RVCacheLUT
RVFormat
RVChannelMap
RVCache
These nodes are evaluated on the CPU
This node interfaces with the RAM cache
RVLinearizePipelineGroup
RVLinearize
RVLensWarp
Converts File Space pixels into the linear working space
Paint and image text annotations are strored on this node
RVPaint
RVOverlay
RVColorPipelineGroup
RVColor
Most of RV's color controls are implemented by the RVColor node
RVLookPipelineGroup
RVLookLUT
RVSourceStereo
RVTransform2D

Figure 2.2: Source Group Internals

The source group takes no inputs. There is eiher an RVFileSource or an RVImageSource node at the leaf position of the source group. A file source contains the name of the media that is provided by the source. An image source contains raw pixels of its media (usually obtained directly from a renderer etc).The source group is responsible for linearizing the incoming pixel data, possibly color correcting it and applying a look, and holding per-source annotation and transforms. Any of these operations can be modified by changing property values on the member nodes of the source group.Pixels are expected to be in the working space (normally linear) after existing the source group.

Command	Mu Return Type	Pyth Return Type	Description
sources ()	(strin	Same	Returns an array of media info for all loaded media.
sourcesAtFrame (int *frame*)	strin	Strin Array	Returns array of source node names (RVFileSource and/or RVImageSource). This is equivalent to nodesOfType("RVSource")
sourceAttributes (string *nodename* , string *medianame* = nil)	(strin	Array of (Stri	Returns an array of image attribute name/value pairs at the current frame. The sourceName can be the node name or the source path as returned by PixelImageInfo, etc. The optional media argument can be used to constraint the attributes to that media only.
sourceMediaInfo (string *nodename* , string *medianame* = nil)	Sourc MediaInfo	Dictionary	Returns a SourceMediaInfo structure for the given source and optional media. The SourceMediaInfo supplies geometric and timing information about the image and sequence
sourceDisplayChannelNames (string *nodename*)	strin	Strin Array	Returns the names of channels in a source which are mapped to the display RGBA channels
addSource (string *filename* , string tag = nil)	void	None	Creates a new source group with the specified media
addSource (string[] *filename* s, string tag = nil)	void	None	Creates a new source group with the specified media
addSourceBegin()	void	None	Optional call providing a fast add source mechanism when adding multiple sources which postpones connecting the added sources to the default views' inputs until after the corresponding addSourceEnd() is called. The way to enable this optimization is to call addSourceBegin() first, followed by a bunch of addSource() calls, and then end with addSourceEnd().
addSourceEnd()	void	None	Optional call providing a fast add source mechanism when adding multiple sources which postpones connecting the added sources to the default views' inputs until after the corresponding addSourceEnd() is called. The way to enable this optimization is to call addSourceBegin() first, followed by a bunch of addSource() calls, and then end with addSourceEnd().
addSources(string[] sources, string tag = "", bool processOpts = false, bool merge = false)	void	None	Add a new source group to the session (see addSource). This function adds the requested sources asynchronously. In addition to the "incoming-source-path" and "new-source" events generated for each source. A "before-progressive-loading" and "after-progressive-loading" event pair will be generated at the appropriate times. An optional tag can be provided which is passed to the generated internal events. The tag can indicate the context in which the addSource() call is occurring (e.g. drag, drop, etc). The optional processOpts argument can be set to true if there are 'option' states like -play, that should be processed after the loading is complete.Note that sources can be single movies/sequences or you can use the "[]" notation from the command line to specify multiple files for the source, like stereo layers, or additional audio files. Per-source command-line flags can also be used here, but the flags should be marked by a "+" rather than a "-". Also note that each argument is a separate element of the input string array. For example a single stereo source might look like string[] {"[", "left.mov", "right.mov", "+rs", "1001", "]" }
addSourceVerbose (string *filename* [], string tag = nil)	strin	Strin	Creates a new source group with the specified media. Returns the name of the source node created.
addToSource (string *filename* , string tag = nil)	void	None	Adds media to an existing source group
addToSource (string[] *filename* s, string tag = nil)	void	None	Adds media to an existing source group

Table 2.2:Commands used to manage and create source groups

2.3.1 Progressive Source Loading

If you need to, RV can load sources asynchronously, also known as progressive source loading. Progessive source loading is turned off by default.

Use one of these methods to enable asynchronous source loading:

- Set `setProgressiveSourceLoading = true`
- With the command line argument: `-progressiveSourceLoading 1`
- With the environment variable: `RV_PROGRESSIVE_SOURCE_LOADING`

  Note: `setProgressiveSourceLoading` affects the behaviour of the following scripting commands:

  - addSource()
  - addSources()
  - addSourceVerbose()
  - addSourcesVerbose()

When progressive source loading is disabled (default setting), the sources are loaded synchronously. This creates the sequence of events:

1. before-progressive-loading
2. source-group-complete media 1
3. source-group-complete media 2
4. after-progressive-loading

When progressive source loading is enabled, RV loads the sources asynchronously:

1. RV creates a movie placeholder in the graph called a movieProxie which has a default duration of 20 frames.
2. It dispatches the actual loading of the media as a work item to a pool of worker threads.
3. Once the media completes the loading operation, the movieProxie placeholder is replaced with the actual movie.

You can expect this sequence of events:

1. before-progressive-loading
2. before-progressive-proxy-loading
3. source-group-proxy-complete media_1
4. source-group-proxy-complete media_2
5. after-progressive-proxy-loading
6. source-group-complete media 1
7. source-group-complete media 2
8. after-progressive-loading

Enabling progressive source loading significantly increases the complexity of scripting. Consider the following example:

```
rv.commands.addSource('/my/clip/1')
rv.commands.setFPS(60.00)
```

- With progressive source loading disabled (default setting), the script behaves as expected: the source is loaded with a frame rate set at 60 fps.
- With progressive source loading enabled, the script doesn't behave as expected. While the source is loaded asynchronously, the frame rate is set temporarily to 60 fps. It's only once the source is completely loaded that the frame rate is set to the source's native rate.

`progressiveSourceLoading` returns the loading state of the current progressive source.

2.3.2 Deleting a Source

When you allocate a new source with `addSource()`, or one of its variants such as `addSourceVerbose()`, RV actually allocates a source group under the hood but only returns the RVFileSource node. Using `deleteNode()` on the returned RVFileSource node deletes the node but not the source group that was created.

To properly delete a source, you must delete its associated source group by using the following:

```
rv.commands.deleteNode(rv.commands.nodeGroup(<loaded_node>))
```

If you prefer, you can instead clear all the sources from the current session with `rv.commands.clearSession()`.

37.1.3 2.4 View Group Node

The view group (RVViewGroup) is responsible for viewing transforms and is the final destination for audio in most cases. The view group is also responsible for rendering any audio waveform visualization.Changing the view in RV is equivalent to changing the input of the view group. There is only one view group in an RV session.The view group contains a pipeline into which arbitrary nodes can be inserted for purposes of QC and visualization. By default, this pipeline is empty (it has no effect).

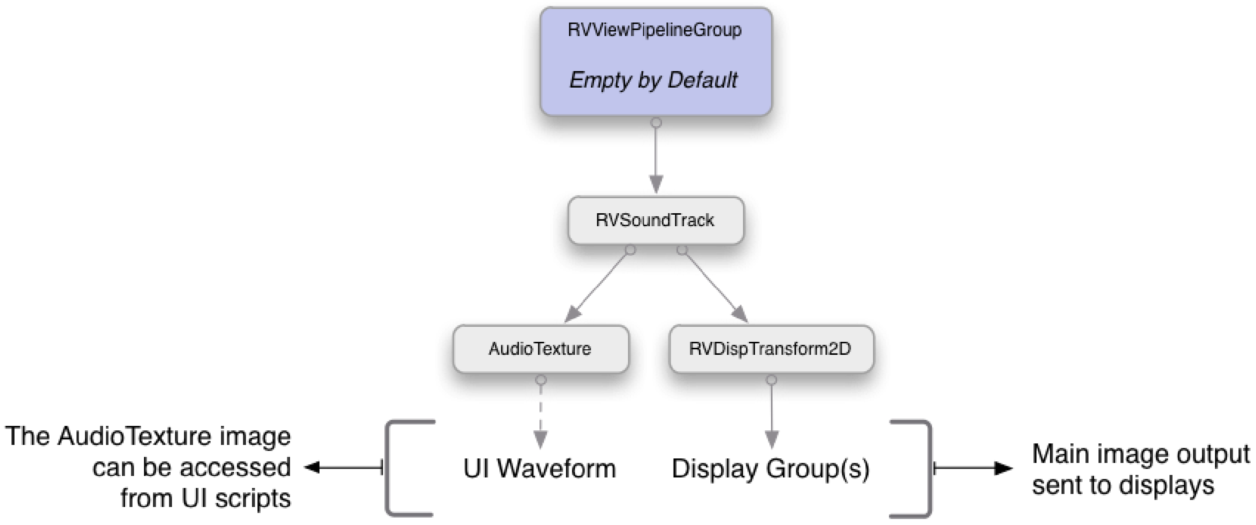


Figure 2.3:View Group Internals

Command	Mu Return Type	Python Return Type	Description
setViewNode (string *nodename*)	void	None	Connect the specified node to the view group
nextViewNode ()	void	None	Switch to next view in the view history (if there is one)
prevViewNode ()	void	None	Switch to next previous in the view history (if there is one)

Table 2.3:High level commands used to change the view group inputs

37.1.4 2.5 Sequence Group Node

The sequence group node causes its inputs to be rendered one after another in time.The internal RVSequence node contains an EDL data structure which determines the order and possibly the frame ranges for its inputs. By default the EDL is automatically created by sequencing the inputs in order from the first to last with their full frame ranges. The automatic EDL function can be turned off in which case arbitrary EDL data can be set including cuts back to a single source multiple times.Each input to a sequence group has a unique sub-graph associated with it that includes an RVPaint node to hold annotation per input and an optional retime node to force all input media to the same FPS.

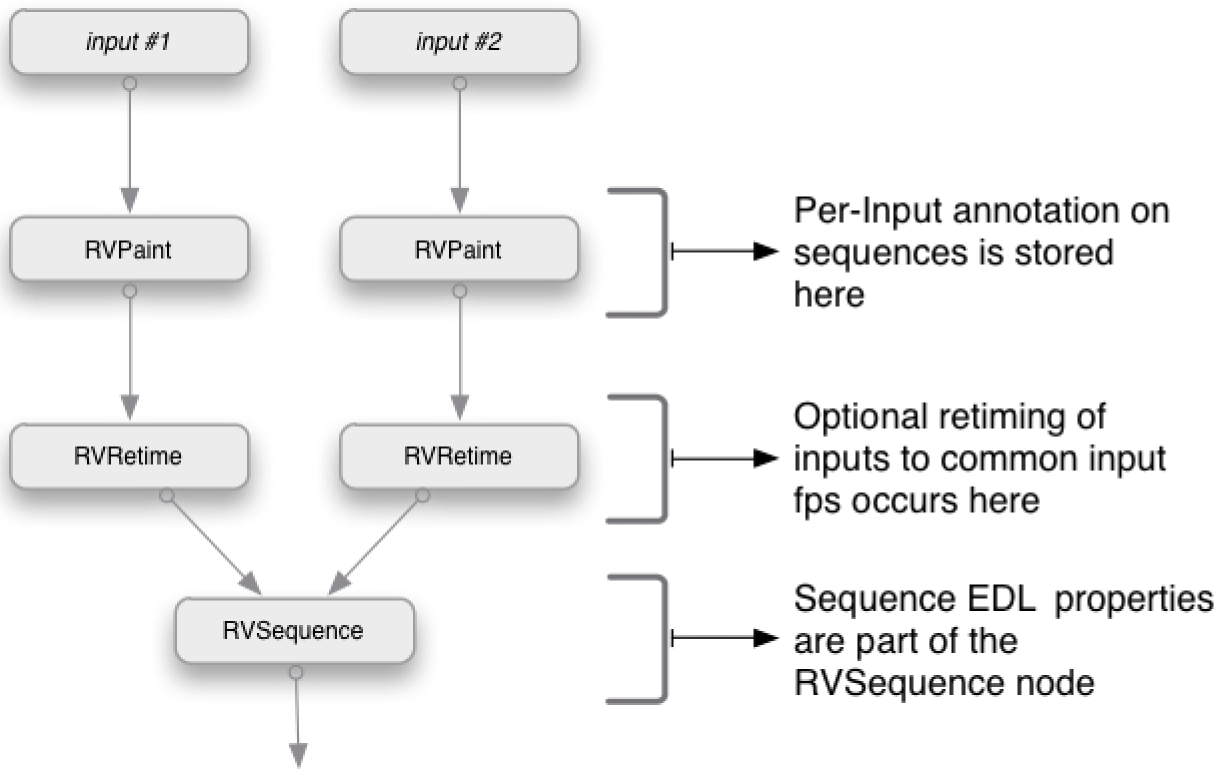

input #1
input #2
RVPaint
RVPaint
RVRetime
RVRetime
RVSequence
Per-Input annotation on sequences is stored here
Optional retiming of inputs to common input fps occurs here
Sequence EDL properties are part of the RVSequence node

Figure 2.4:Sequence Group Internals

37.1.5 2.6 Stack Group Node

The stack group node displays its inputs on top of each other and can control a crop per input in order to allow pixels from lower layers to be seen under upper layers. Similar to a sequence group, the stack group contains an optional retime node per input in order to force all of the input FPS' to the same value.Unlike the sequence group, the stack group's paint node stores annotation after the stacking so it always appears on top of all images.

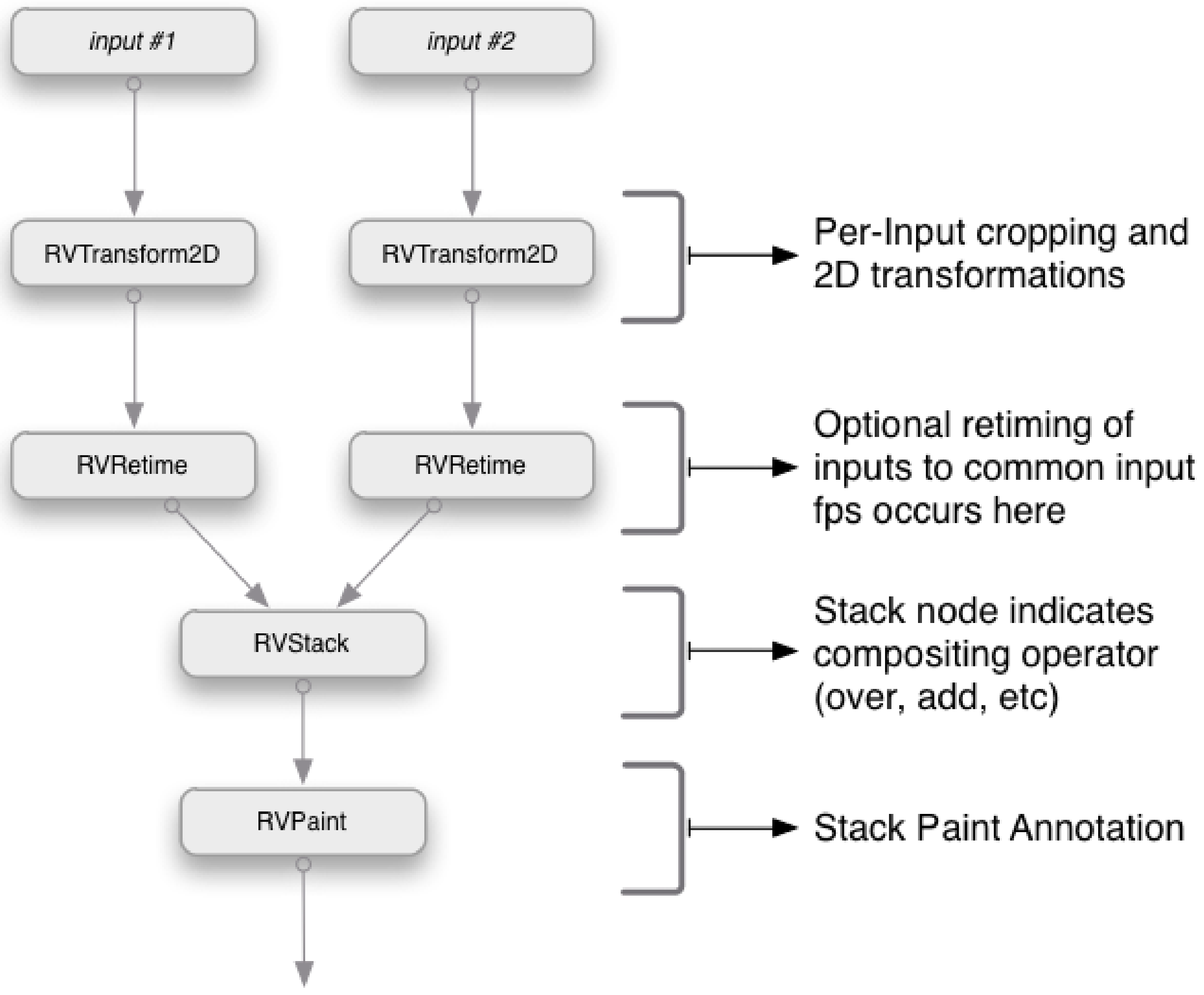

input #1
input #2
RVTransform2D
RVTransform2D
Per-Input cropping and 2D transformations
RVRetime
RVRetime
Optional retiming of inputs to common input fps occurs here
RVStack
Stack node indicates compositing operator (over, add, etc)
RVPaint
Stack Paint Annotation

Figure 2.5:Stack Group Internals

37.1.6 2.7 Layout Group Node

The layout group is similar to a stack group, but instead of showing all of its inputs on top of one another, the inputs are transformed into a grid, row, column, or under control of the user (manually). Like the other group nodes, there is an optional retime node to force all inputs to a common FPS. Annotations on the layout group appear on top of all images regardless of their input order.

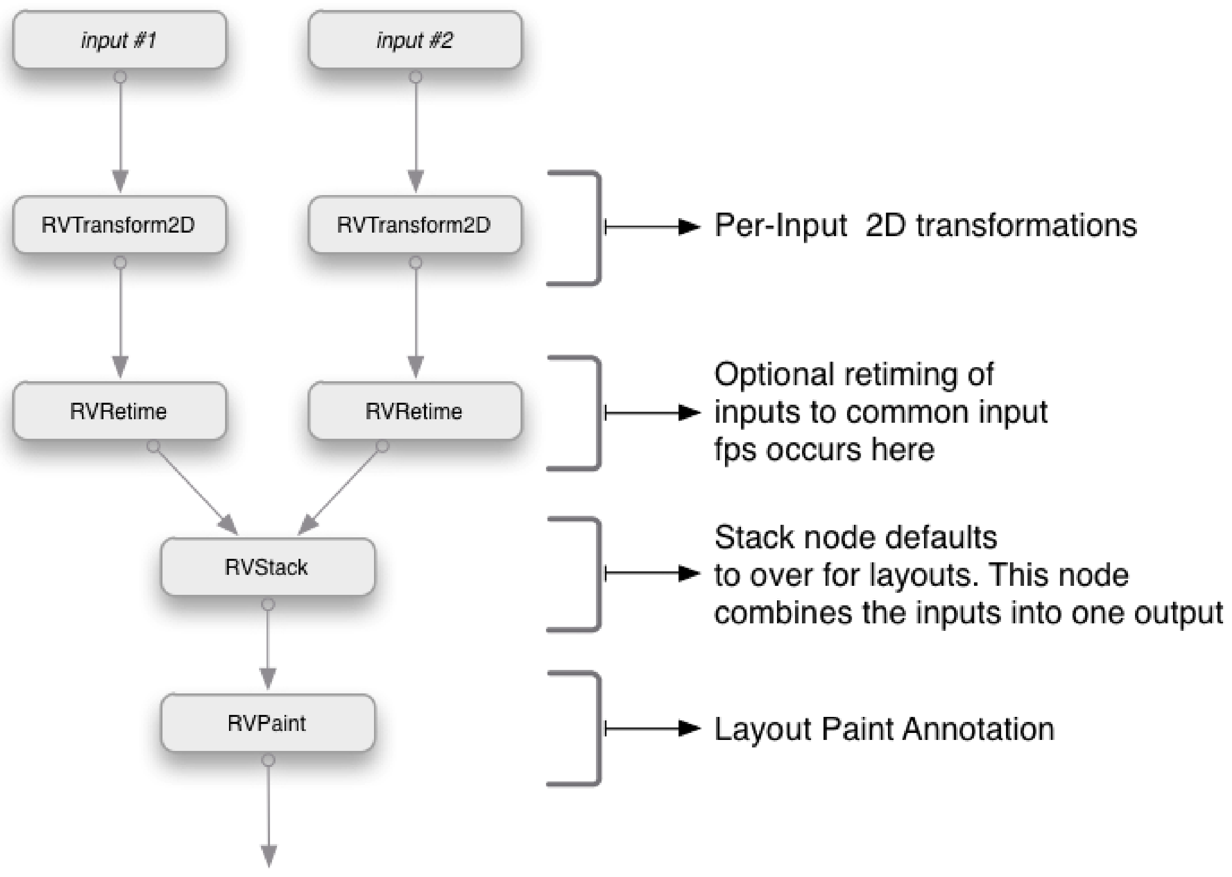

input #1
input #2
RVTransform2D
RVTransform2D
Per-Input 2D transformations
RVRetime
RVRetime
Optional retiming of inputs to common input fps occurs here
RVStack
Stack node defaults to over for layouts. This node combines the inputs into one output
RVPaint
Layout Paint Annotation

Figure 2.6:Layout Group Internals

37.1.7 2.8 Display Group Node

There is one display group for each video device accessible to RV. For example in the case of a dual monitor setup, there would be two display groups: one for each monitor. The display group has two functions: to prepare the working space pixels for display on the associated device and to set any stereo modes for that device.By default the display group's pipeline uses an RVDisplayColor node to provide the color correction. The user can use any node for that purpose instead or in addition to the existing RVDisplayColor. For example, when OpenColorIO is being used a DisplayOCIONode is used in place of the RVDisplayColor.For a given desktop setup with multiple monitors only one of the RVDisplayGroups is active at a time: the one corresponding to the monitor that RV's main window is on. In presentation mode, two RVDisplayGroups will be active: one for RV's main window and one for the presentation device. Each display group has properties which identify their associated device.Changes to a display group affect the color and stereo mode for the associated device only. In order to make a global color change that affects all devices, a node should be inserted into the view group's pipeline or earlier in the graph.

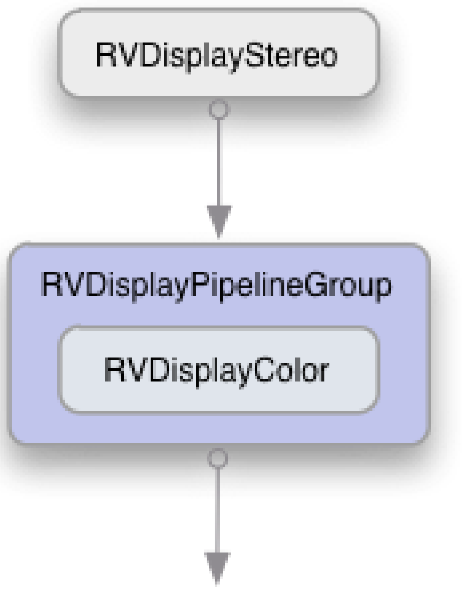


Figure 2.7:Display Group Internals

37.1.8 2.9 Addressing Properties

A full property name has three parts: the node name, the component name, and the property name. These are concatenated together with dots like *nodename.componentname.propertyname* . Each property has its own type which can be set and retrieved with one of the set or get functions. You must use the correct get or set function to access the property. For example, to set the display gamma, which is part of the ``display" node, you need to use setFloatProperty() like so in Mu:

```
setFloatProperty("display.color.gamma", float[] {2.2, 2.2, 2.2}, true)
```

or in Python:

```
setFloatProperty("display.color.gamma",  [2.2, 2.2, 2.2], True)
```

In this case the value is being set to 2.2.

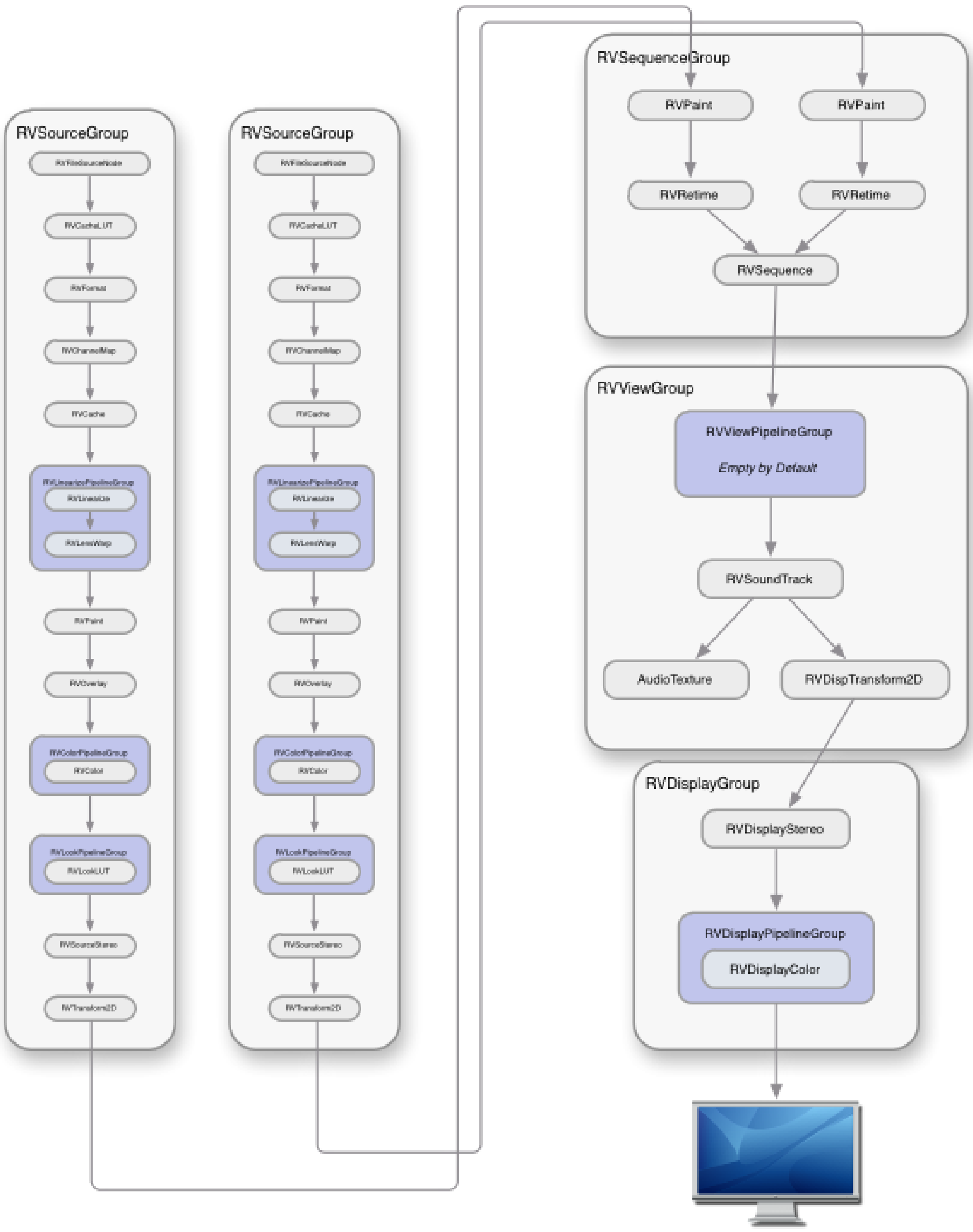


Figure 2.8:Conceptual diagram of RV Image and Audio Processing Graph for a session with a single sequence of two sources. The default stack and layout are not included in this diagram, but would be present.

In an RV session, some node names will vary per the source(s) being displayed and some will not. Figure *2.8* shows a pipeline diagram for one possible configuration and indicates which are per-source (duplicated) and which are not.At any point in time, a subset of the graph is active. For example if you have three sources in a session and RV is in sequence mode, at any given frame only one source branch will be active. There is a second way to address nodes in RV: by their types. This is done by putting a hash (#) in front of the type name. Addressing by node type will affect all of the currently active nodes of the given type. For example, a property in the color node is exposure which can be addressed directly like this in Mu:

```
color.color.exposure
```

or using the type name like this:

```
#RVColor.color.exposure
```

When the "#" type name syntax is used, and you use one of the set or get functions on the property, only nodes that are currently active and which are the first reachable of the given type will be considered. So in this case, if we were to set the exposure using type-addressing:

```
setFloatProperty("#RVColor.color.exposure", float[] {2.0, 2.0, 2.0}, true)
```

or in Python:

```
setFloatProperty("#RVColor.color.exposure", [2.0, 2.0, 2.0], True)
```

In sequence mode (i.e. the default case), only one RVColor node is usually active at a time (the one belonging to the source being viewed at the current frame). In stack mode, the RVColor nodes for all of the sources could be active. In that case, they will all have their exposure set. In the UI, properties are almost exclusively addressed in this manner so that making changes affects the currently visible sources only. See figure *2.9* for a diagrammatic explanation.

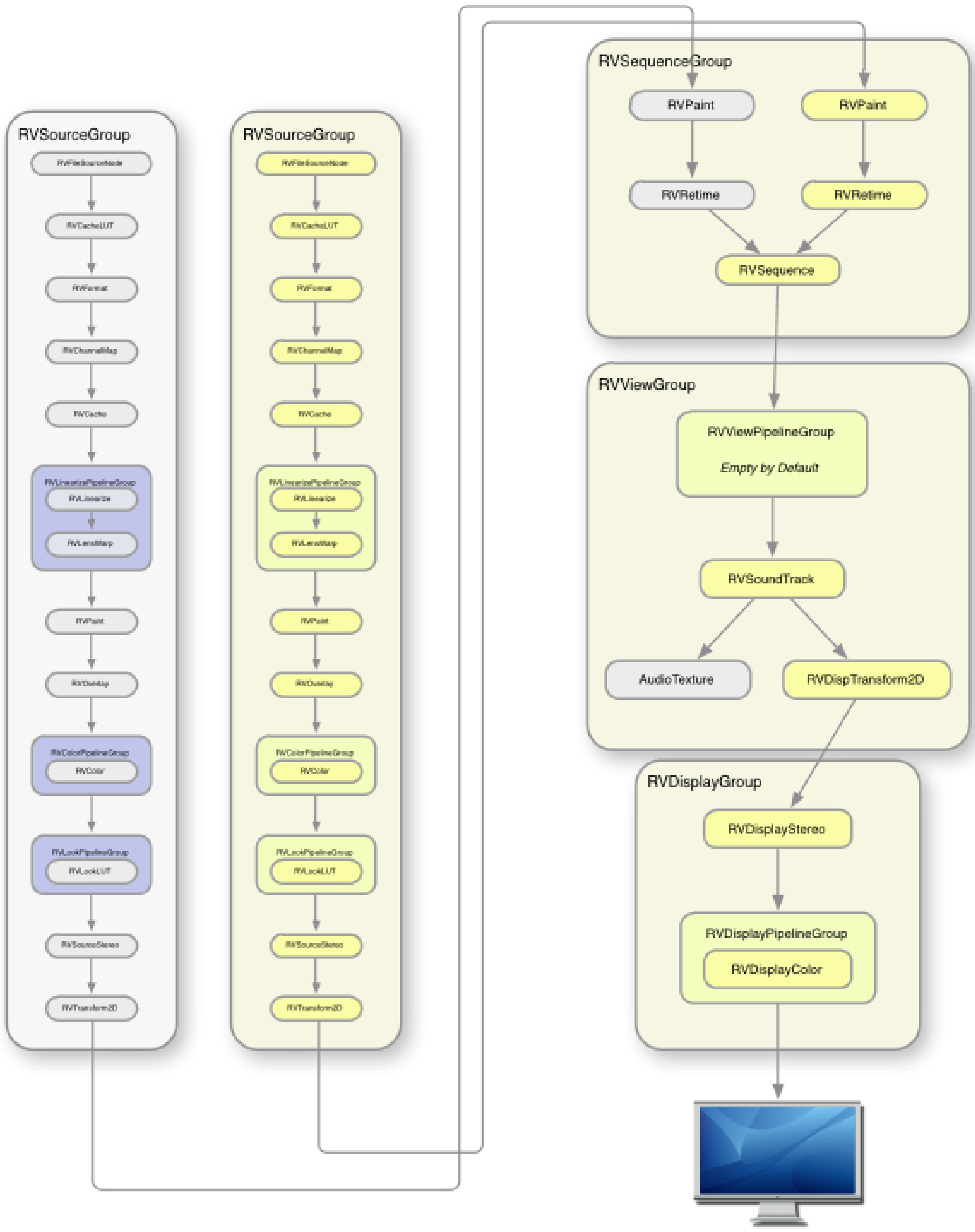


Figure 2.9: Active Nodes in the Image Processing Graph

The active nodes are those nodes which contribute to the rendered view at any given frame. In this configuration, when the sequence is active, there is only one source branch active (the yellow nodes). By addressing properties using their node's type name, you can affect only active nodes with that type without needing search for the exact node(s).There is an additional shorthand using "@" in front of a type name:

```
@RVDisplayColor.color.brightness
```

The above would affect only the first RVDisplayColor node it finds instead of all RVDisplayColor nodes of depth 1 like "#" does. This is useful with presentation mode for example because setting the brightness would be confined to the first RVDisplayColor node which would be the one associated with the presentation device. If "#" was used, all devices would have their brightness modified. The utility of the "@" syntax is limited compared to "#" so if you are unsure of which to use try "#" first.Chapter *16* has all the details about each node type.

37.1.9 2.10 User Defined Properties

It's possible to add your own properties when creating an RV file from scratch or from the user interface code using the newProperty() function.Why would you want to do this? There are a few reasons to add a user defined property:

1. You wish to save something in a session file that was created interactively by the user.
2. You're generating session files from outside RV and you want to include additional information (e.g. production tracking, annotations) which you'd like to have available when RV plays the session file.

Some of the packages that come with RV show how to implement functionality for the above.

37.1.10 2.11 Getting Information From Images

RV's UI often needs to take actions that depend on the context. Usually the context is the current image being displayed. Table *2.4* shows the most useful command functions for getting information about displayed images.

Command	Description
sourceAtPixel	Given a point in the view, returns a structure with information about the source(s) underneath the point.
sourcesRendered	Returns information about all sources rendered in the current view (even those that may have been culled).
sourceLayers	Given the name of a source, returns the layers in the source.
sourceGeometry	Given the name of a source, returns the geometry (bounding box) of that source.
sourceMedia	Given the name of a source, returns the a list of its media files.
sourcePixelValue	Given the name of a source and a coordinate in the image, returns an RGBA pixel value at that coordinate. This function may convert chroma image pixels to Rec709 primary RGB in the process.
sourceAttributes	Given the name of a source and optionally the name a particular media file in the source, returns an array of tuples which contain attribute names and values.
sourceStructure	Given the name of a source and optionally the name a particular media file in the source, returns information about image size, bit depth, number of channels, underlying data type, and number of planes in the image.
sourceDisplayChannelNames	Given the name of a source, returns an array of channel names current being displayed.

Table 2.4: Command Functions for Querying Displayed Images

For example, when automating color management, the color space of the image or the origin of the image may be required to determine the best way to view the image (e.g., for a certain kind of DPX file you might want to use a particular display or file LUT). The color space is often stored as image attribute. In some cases, image attributes are misleading–for example, a well known 3D software package renders images with incorrect information about pixel aspect ratio—usually other information in the image attributes coupled with the file name and origin are enough to make a good guess.

CHAPTER

THIRTYEIGHT

CHAPTER 3 - WRITING A CUSTOM GLSL NODE

RV can use custom shaders to do GPU accelerated image processing. Shaders are authored in a superset of the GLSL language. These GLSL shaders become image processing Nodes in RV's session. Note that nodes can be either "signed" or "unsigned". As of RV6, nodes can be loaded by any product in the RV line (RV, RVIO). The most basic workflow is as follows:

- Create a file with a custom GLSL function (the Shader) using RV's extended GLSL language.
- Create a GTO node definition file which references the Shader file.
- Test and adjust the shader/node as necessary.
- Place the node definition and shader in the RV_SUPPORT_PATH under the Nodes directory for use by other users.

38.1 3.1 Node Definition Files

Node definition files are GTO files which completely describe the operation of an image processing node in the image/audio processing graph.A node definition appears as a single GTO object of type IPNodeDefinition. This makes it possible for a node definition to appear in a session file directly or in an external definition file which can contain a library of definition objects.The meat of a node definition is source code for a kernel function written in an augmented version of GLSL which is described below.The following example defines a node called "Gamma" which takes a single input image and applies a gamma correction:

```
 GTOa (4)

Gamma : IPNodeDefinition (1)
{
    node
    {
        string evaluationType = "color"
        string defaultName = "gamma"
        string creator = "Tweak Software"
        string documentation = "Gamma"
        int userVisible = 1
    }

    render
    {
        int intermediate = 0
    }

```

(continues on next page)

(continued from previous page)

```
    function
    {
        string name = "main" # OPTIONAL
       string glsl = "vec4 main (const in inputImage in0, const in vec3 gamma) { return␣
↪vec4(pow(in0().rgb, gamma), in0().a); }"
    }

    parameters
    {
        float[3] gamma = [ [ 0.4545 0.4545 0.4545 ] ]
    }
}
```

38.2 3.2 Fields in the IPNodeDefinition

node.evaluationType

one of:

color	one input, per-pixel operations only
filter	one input, multiple input pixels sampled to create one output pixel
transition	two inputs, an animated transition
merge	one or more inputs, typically per-pixel operation
combine	one input to node, many inputs to function, pulls views, layers, eyes, multiple frames from input

node.defaultName

the default name prefix for newly instantiated nodes

node.creator

documentation about definition author

node.documentation

Possibly html documentation string. In practice this may be quite large

node.userVisible

if non-0 a user can create this node directly otherwise only programmatically

render.intermediate

if non-0 the node results are forced to be cached

function.name

the name of the entry point in source code. By default this is main

function.fetches

approximate number of fetches performed by function. This is meaningful for filters. E.g. a 3x3 blur filter does 9 fetches.

function.glsl

Source code for the function in the augmented GLSL language. Alternately this can be a file URL pointing to the location of a separate text file containing the source code. See below for more details on file URL handling.

parameters

bindable parameters should be given default values in the parameters component. Special variables need not be given default values. (e.g. input images, the current frame, etc).

38.2.1 3.2.1 The "combine" Evaluation Type

A "combine" node will evaluate its single input once for each parameter to the shader of type "inputImage".The names of the inputImage parameters in the shader may be chosen to be meaningful to the shader writer; they are not meaningful to the evaluation of the combine node. The order of the inputImage parameters in the shader parameter list will correspond to the multiple evaluations of the node's input (see below).Each time the input is evaluated, there are a number of variations that can be made in the context by way of properties specified in the node definition. To be clear, these properties are specified in the "parameters" section of the node definition, but they are "evaluation parameters" not shader parameters. These are:

eye	stereo eye, int, 0 for left, 1 for right
channel	color channel, string, eg "R" or "Z"
layer	named image layer, string, typically from EXR file
view	named image view, string, typically from EXR file
frame	absolute frame number, int
offset	frame number offset, int

A context-modifying property has 3 parts: the name (see above), an "inputImage index" (the int tacked onto the name), and the value. The affect of the parameter is that the context of the evaluation of the input specified by the index will be modified by the value. So for example "int eye0 = 1" means that the "eye" parameter of the context used in the first evaluation of the input will be set to "1".So for example, suppose a "StereoDifference" has this definition:

```
StereoDifference : IPNodeDefinition (1)
{
    node
    {
        string evaluationType = "combine"
    }
    function
    {
        string glsl = "file://${HERE}/StereoQC.glsl"
    }
    parameters
    {
        int eye0 = 0
        int eye1 = 1
    }
}
```

And this shader parameter list:

```
vec4 main (const in inputImage left, const in inputImage right)
```

Then:

- It's a Combine node, so it's single input can be evaluated multiple times.
- The shader has two inputImage parameters so the input will be evaluated twice.
- The node definition contains "eye" parameters, so the eye value of the evaluation context will differ in different evaluations of the input.
- In the first evaluation of the input (index = 0), the context's eye value will be set to 0, and in the second it will be set to 1.
- The results of each evaluation of the input is made available to the shader in the corresponding inputImage parameter.

As another example, here's a "FrameBlend" node:

```
 FrameBlend : IPNodeDefinition (1)
{
    node
    {
        string evaluationType = "combine"
    }
    function
    {
        string glsl = "file://${HERE}/FrameBlend.glsl"
    }
    parameters
    {
        int offset0 = -2
        int offset1 = -1
        int offset2 = 0
        int offset3 = 1
        int offset4 = 2
    }
}
```

And this shader parameter list:

```
vec4 main (const in inputImage in0,
          const in inputImage in1,
          const in inputImage in2,
          const in inputImage in3,
          const in inputImage in4)
```

So the result is that the input to the FrameBlend node will be evaluated 5 times, and in each case the evaluation context will have a frame value that is equal to the incoming frame value, plus the corresponding offset. Note that the shader doesn't know anything about this, and from it's point of view it has 5 input images.

38.3 3.3 Alternate File URL

Language source code can be either inlined for a self contained definition or can be a modified file URL which points to an external file. An example file URL might be:

```
file:///Users/foo/glsl/foo_shader_source.glsl
```

If the node definition reader sees a file URL it will also perform variable substitution from the environment and any special predefined variables. For example if the $HOME environment variable exists the following would be equivalent on a Mac:

```
file://${HOME}/glsl/foo_shader_source.glsl
```

There is currently one special variable defined called $HERE which has the value of the directory in which the definition file lives. So if for example the node definition file lives in the filesystem like so:

```
/Users/foo/nodes/my_nodes.gto
/Users/foo/nodes/glsl/node1_source_code.glsl
/Users/foo/nodes/glsl/node2_source_code.glsl
/Users/foo/nodes/glsl/node3_source_code.glsl
```

and it references the GLSL files mentioned above then valid file URLs for the source files would like this:

```
file://${HERE}/glsl/node1_source_code.glsl
file://${HERE}/glsl/node2_source_code.glsl
file://${HERE}/glsl/node3_source_code.glsl
```

38.4 3.4 Augmented GLSL Syntax

GLSL source code can contain any set of functions and global static data but may not contain any uniform block definitions. Uniform block values are managed by the underlying renderer.

38.4.1 3.4.1 The main() Function

The name of the function which serves as the entry point must be specified if it's not main().The main() function must always return a vec4 indicating the computed color at the current pixel.For each input to a node there should be a parameter of type inputImage. The parameters are applied in the order they appear. So the first node image input is assigned to the first inputImage parameter and so on.There are four special parameters which are supplied by the renderer:

float frame	The current frame number (local to the node)
float fps	The current frame rate (local to the node)
float baseFrame	The current global frame number
float stereoEye	The current stereo eye (0=left,1=right,2=default)

Table 3.1:Special Parameters to main() FunctionAny additional parameters are searched for in the 'parameters' component of the node. When a node is instantiated this will be populated by additional parameters to the main() function.For example, the Gamma node defined above has the following main() function:

```
vec4 main (const in inputImage in0, const in vec3 gamma)
{
    vec4 P = in0();
    return vec4(pow(P.rgb, gamma), P.a);
}
```

In this case the node can only take a single input and will have a property called parameters.gamma of type float[3]. By changing the gamma property, the user can modify the behavior of the Gamma node.

38.4.2 3.4.2 The inputImage Type

A new type inputImage has been added to GLSL. This type represents the input images to the node. So a node with one image argument must take a single inputImage argument. Likewise, a two input node should take two such arguments.There are a number of operations that can be done on a inputImage object. For the following examples the parameter name will be called i.

i()	Returns the current pixel value as a vec4. Functions which only call this operator on inputImage parameters can be of type "color"
i(vec2 *OFF*)	If *P* is the current pixel location this returns the pixel at *OFF* + *P*
i.size()	Returns a vec2 (width,height) indicating the size of the input image
i.st	Returns the absolute current pixel coordinates ([0,width], [0,height]) with swizzling

Table 3.2:Type inputImage OperationsUse of the inputImage type as a function argument is limited to the main() function. Use of the inputImage type as a function should be minimized where possible e.g. the result should be stored into a local variable and the local variable used there after. For example:

```
vec4 P = i();
return vec4(P.rgb * 0.5, P.a);
```

NOTE : The *st* value return by an inputImage has a value ranging from 0 to the width in X and 0 to the height in Y. So for example, the pixel value of the first pixel in the image is located at (0.5, 0.5) not at (0, 0). Similarily, the last pixel in the image is located at (width-0.5, height-0.5) not (width-1, height-1) as might be expected. See ARB_texture_rectangle for information on why this is. In GLSL 1.5 and greater the *rectangle* coordinates are built into the language.

38.4.3 3.4.3 The outputImage Type

The type outputImage has also been added. This type provides information about the output framebuffer.The main() function may have a single outputImage parameter. You cannot pass an outputImage to auxiliary functions nor can you have outputImage parameter to an auxiliary function. You can pass the results of operations on the outputImage object to other functions.outputImage has the following operations:

w.st	Returns the absolute fragment coordinate with swizzling
w.size()	Returns the size of the output framebuffer as a vec2

Table 3.3:Type outputImage Operations

38.4.4 3.4.4 Use of Samplers

Samplers can be used as inputs to node functions. The sampler name and type must match an existing parameter property on the node. So for example a 1D sampler would correspond to a 1D property the value of which is a scalar array. A 3D sampler would have a type like float[3,32,32,32] if it were an RGB 32^3 LUT.

sampler1D	*type* [*D* , *X*]
sampler2D	*type* [*D* , *X* , *Y*]
sampler2DRect	*type* [*D* , *X* , *Y*]
sampler3D	*type* [*D* , *X* , *Y* , *Z*]

Table 3.4:Sampler to Parameter Type Correspondences

In the above table, *D* would normally 1, 3, or 4 for scalar, RGB, or RGBA. A value of 2 is possible but unusual.Use the new style texture() call instead of the non-overloaded pre GLSL 1.30 function calls like texture3D() or texture2DRect(). This should be the case even when the driver only supports 1.20.

38.5 3.5 Testing the Node Definition

Once you have a NodeDefinition GTO file that contains or references your shader code as described above, you can test the node as follows:

1. Add the node definition file to the Nodes directory on your RV_SUPPORT_PATH. For example, on Linux, you can put it in $HOME/.rv/Nodes. If the GLSL code is in a separate file, it should be in the location specified by the URL in the Node Definition file.You can use the ${HERE}/myshader.glsl notation (described above) to indicate that the GLSL is to be found in the same directory.
2. Start RV and from the Session Manager add a node with the "plus" button or the right-click menu ("New Viewable") by choosing "Add Node by Type" and entering the type name of the new node ("Gamma" in the above example).
3. At this point you might want to save a Session File for easy testing.
4. You can now iterate by changing your shader code or the parameter values in the Session File and re-running RV to test.

38.6 3.6 Publishing the Node Definition

When you have tested sufficiently in RV and would like to make the new Node Definition available to other users running RV, RVIO, etc, you need to:**Make the Node Definition available to users** . RV will pick up Node Definition files from any Nodes sub-directory along the RV_SUPPORT_PATH. So your definitions can be distributed by simply inserting them into those directories, or by including them in an RV Package (any GTO/GLSL files in an RV Package will be added to the appropriate "Nodes" sub-directory when the Package is installed). With some new node types, you may want to distribute Python or or Mu code to help the user manage the creation and parameter-editing of the new nodes, so wrapping all that up in an RV Package would be appropriate in those cases.

CHAPTER

THIRTYNINE

CHAPTER 4 - PYTHON

Which language should you use to customize RV? In short, we recommend using Python. You can use Python3 in RV in conjunction with Mu, or in place of it. It's even possible to call Python commands from Mu and vice versa. Python is a full peer to Mu as far as RV is concerned.

Note: there are some slight differences that need to be noted when translating code between the two languages: In Python the modules names required by RV are the same as in Mu. As of this writing, these are commands, extra_commands, rvtypes, and rvui. However, the Python modules all live in the rv package. This package is in-memory and only available at RV's runtime. You can access these commands via writing your own custom MinorMode package. So while in Mu, you can:

```
use commands
```

```
require commands
```

to make the commands visible in the current namespace. In Python you need to include the package name:

```
from rv.commands import *
```

or

```
import rv.commands
```

39.1 Open RV Python quickstart

In order to extend RV using Python you will be making a "mode" as part of an rvpkg package—this is identical to the way it's done in Mu and this is the method that we use internally to add new functions to RV's interface. Creation of a modes and packages is documented later in this chapter. Here is a very simple mode written in Python which is part of the RV packages as `pyhello-1.1.rvpkg`.

```
import rv.rvtypes
import rv.commands

class PyHello(rv.rvtypes.MinorMode):
    "A simple example that shows how to make shift-Z start/stop playback"

    def togglePlayback(self, event):
        if rv.commands.isPlaying():
            rv.commands.stop()
        else:
```

(continues on next page)

(continued from previous page)

```
            rv.commands.play()

    def __init__(self):
        rv.rvtypes.MinorMode.__init__(self)
        self.init("pyhello",  [("key-down--Z", self.togglePlayback, "Z key")],  None)

    def createMode():
        "Required to initialize the module. RV will call this function to create your␣
↪mode."
        return PyHello()
```

39.1.1 Documentation

The command API is nearly identical to Mu. There are a few modules which are important to know about: `rv.rvtypes`, `rv.commands`, `rv.extra_commands`, and `rv.rvui`. These implement the base Python interface to RV.

There is no separate documentation for RV's command API in Python (e.g., via Pydoc), but you can use the existing Mu Command API Browser available under RV's Help menu. The commands and extra_commands modules are basically identical between the two languages.

39.2 4.1 Calling Mu From Python

It's possible to call Mu code from Python, but in practice you will probably not need to do this unless you need to interface with existing packages written in Mu. To call a Mu function from Python, you need to import the MuSymbol type from the pymu module. In this example, the play function is imported and called F on the Python side. F is then executed:

```
from pymu import MuSymbol
F = MuSymbol("commands.play")
F()
```

If the Mu function has arguments you supply them when calling. Return values are automatically converted between languages. The conversions are indicated in Figure *4.3*.

```
from pymu import MuSymbol
F = MuSymbol("commands.isPlaying")
G = MuSymbol("commands.setWindowTitle")
if F() == True:
    G("PLAYING")
```

Once a MuSymbol object has been created, the overhead to call it is minimal. All of the Mu commands module is imported on start up or reimplemented as native CPython in the Python rv.commands module so you will not need to create MuSymbol objects yourself; just import rv.commands and use the pre-existing ones.

When a Mu function parameter takes a class instance, a Python dictionary can be passed in. When a Mu function returns a class, a dictionary will be returned. Python dictionaries should have string keys which have the same names as the Mu class fields and corresponding values of the correct types. For example, the Mu class Foo { int a; float b; } as instantiated as Foo(1, 2.0) will be converted to the Python dictionary {'a' : 1, 'b' : 2.0} and vice versa. Existing Mu code can be leveraged with the rv.runtime.eval call to evaluate arbitrary Mu from Python. The second argument to the eval function is a list of Mu modules required for the code to execute and the result of the evaluation will be returned as a string. For example, here's a function that could be a render method on a mode; it uses the Mu gltext module to draw the name of each visible source on the image:

```
def myRender (event) :
    event.reject()

    for s in rv.commands.renderedImages() :
        if (rv.commands.nodeType(rv.commands.nodeGroup(s["node"])) != "RVSourceGroup") :
            continue
        geom    = rv.commands.imageGeometry(s["name"])

        if (len(geom) == 0) :
            continue

        x       = geom[0][0]
        y       = (geom[0][1] + geom[2][1]) / 2.0
        domain  = event.domain()
        w       = domain[0]
        h       = domain[1]

        drawCode = """
       {
           rvui.setupProjection (%d, %d);
           gltext.color (rvtypes.Color(1.0,1.0,1.0,1));
           gltext.size(14);
           gltext.writeAt(%f, %f, extra_commands.uiName("%s"));
       }
       """
       rv.runtime.eval(drawCode % (w, h, float(x), float(y), s["node"]), ["rvui",
↪"rvtypes", "extra_commands"])
```

Note: Python code in RV can assume that default parameters in Mu functions will be supplied if needed.

39.3 4.2 Calling Python From Mu

There are two ways to call Python from Mu code: a Python function being used as a call back function from Mu or via the "python" Mu module. In order to use a Python callable object as a call back from Mu code simply pass the callable object to the Mu function. The call back function's arguments will be converted according to the Mu to Python value conversion rules show in Figure *4.3* . There are restrictions on which callable objects can be used; only callable objects which return values of None, Float, Int, String, Unicode, Bool, or have no return value are currently allowed. Callable objects which return unsupported values will cause a Mu exception to be thrown after the callable returns. The Mu "python" module implements a small subset of the CPython API. You can see documentation for this module in the Mu Command API Browser under the Help menu. Here is an example of how you would call os.path.join from Python in Mu.

```
require python;

let pyModule = python.PyImport_Import ("os");

python.PyObject pyMethod = python.PyObject_GetAttr (pyModule, "path");
python.PyObject pyMethod2 = python.PyObject_GetAttr (pyMethod, "join");
```

(continues on next page)

(continued from previous page)

```
string result = to_string(python.PyObject_CallObject (pyMethod2, ("root","directory",
↪"subdirectory","file")));

print("result: %s\n" % result); // Prints "result: root/directory/subdirectory/file"
```

If the method you want to call takes no arguments like os.getcwd, then you will want to call it in the following manner.

```
require python;

let pyModule = python.PyImport_Import ("os");

python.PyObject pyMethod = python.PyObject_GetAttr (pyModule, "getcwd");

string result = to_string(python.PyObject_CallObject (pyMethod, python.PyTuple_New(0)));

print("result: %s\n" % result); // Prints "result: /var/tmp"
```

If the method you want to call require the python class instance "self" as an argument, you can get it by using the ModeManager as in the following exemple

```
let pyModule = python.PyImport_Import ("sgtk_bootstrap");
python.PyObject pyMethod = python.PyObject_GetAttr (pyModule, "ToolkitBootstrap");
python.PyObject pyMethod2 = python.PyObject_GetAttr (pyMethod, "queue_launch_import_cut_
↪app");

State state = data();
ModeManagerMode manager = state.modeManager;
ModeManagerMode.ModeEntry entry = manager.findModeEntry ("sgtk_bootstrap");

if (entry neq nil)
{
   PyMinorMode sgtkMode = entry.mode;
   python.PyObject_CallObject (pyMethod2, (sgtkMode._pymode, "no event"));
}
```

If you are interested in retrieving an attribute alone then here is an example of how you would call sys.platform from Python in Mu.

```
require python;

let pyModule = python.PyImport_Import ("sys");

python.PyObject pyAttr = python.PyObject_GetAttr (pyModule, "platform");

string result = to_string(pyAttr);

print("result: %s\n" % result); // Prints "result: darwin"
```

39.4 4.3 Python Mu Type Conversions

Python Type	Converts to Mu Type	Converts To Python Type	
Str or Unicode	string	Unicode string	Normal byte strings and unicode strings are both converted to Mu's unicode string. Mu strings always convert to unicode Python strings.
Int	int, short, or byte	Int	
Long	int64	Long	
Float	float or half or double	Float	Mu double values may lose precision. Python float values may lose precision if passed to a Mu function that takes a half.
Bool	bool	Bool	
(Float, Float)	vector float[2]	(Float, Float)	Vectors are represented as tuples in Python
(Float, Float, Float)	vector float[3]	(Float, Float, Float)	
(Float, Float, Float, Float)	vector float[4]	(Float, Float, Float, Float)	
Event	Event	Event	
MuSymbol	runtime.symb	MuSymbol	
Tuple	tuple	Tuple	Tuple elements each convert independently. NOTE: two to four element Float tuples will convert to vector float[N] in Mu. Currently there is no way to force conversion of these Float-only tuples to Mu float tuples.
List	type[] or type[N]	List	Arrays (Lists) convert back and forth
Dictionary	Class	Dictionary	Class labels become dictionary keys
Callable Object	Function Object	Not Applicable	Callable objects may be passed to Mu functions where a Mu function type is expected. This allows Python functions to be used as Mu call back functions.

Table 4.1:Mu-Python Value Conversion

39.5 4.4 PySide Example

You can use PySide2 to make Qt interface components (RV is a Qt Application). Below is a simple pyside example using RV's py-interp.

```
#!/Applications/RV.app/Contents/MacOS/py-interp

# Import PySide2 classes
import sys
from PySide2.QtCore import *
from PySide2.QtGui import *
from PySide2.QtWidgets import *

# Create a Qt application.
# IMPORTANT: RV's py-interp contains an instance of QApplication;
# so always check if an instance already exists.
app = QApplication.instance()
if app == None:
    app = QApplication(sys.argv)

# Display the file path of the app.
print(f"{app.applicationFilePath()}")

# Create a Label and show it.
label = QLabel("Using RV's PySide2")
label.show()

# Enter Qt application main loop.
app.exec_()

sys.exit()
```

To access RV's essential session window Qt QWidgets, i.e. the main window, the GL view, top tool bar and bottom tool bar, import the Python module 'rv.qtutils'.

```
import rv.qtutils

# Gets the current RV session windows as a PySide QMainWindow.
rvSessionWindow = rv.qtutils.sessionWindow()

# Gets the current RV session GL view as a PySide QGLWidget.
rvSessionGLView = rv.qtutils.sessionGLView()

# Gets the current RV session top tool bar as a PySide QToolBar.
rvSessionTopToolBar = rv.qtutils.sessionTopToolBar()

# Gets the current RV session bottom tool bar as a PySide QToolBar.
rvSessionBottomToolBar = rv.qtutils.sessionBottomToolBar()
```

39.6 4.5 Open RV Python Implementation FAQ

39.6.1 Can I draw on the view the way Mu does using OpenGL?

Yes. If you bind to a render event you can draw using PyOpenGL.

39.6.2 Module XXX is missing. Where can I get it?

Use `py-interp -m pip` to get the missing package, like you would any other python package.

If the module is not included and it's a CPython module (written in C) you will need to compile it yourself. The compiled module must be added to the `Python` plug-ins folder.

39.6.3 The commands.bind() function in Python doesn't work the same way as in Mu? How do I use it?

Python currently requires all arguments to bind(). So to make it the "short form" do:

```
bind("default", "global", event, func, event_doc_string).
```

39.6.4 Does the Python <-> Mu bridge slow things down?

The Python <-> Mu bridge does not slow things down. The MuSymbol type used to interface between them completely skips interpreted Mu code if it's calling a "native" Mu function from Python. All of the RV commands are native Mu functions. So there's a thin layer between the Python call and the actual underlying RV command (which is largely language agnostic).

The Mu calling into Python bridge is roughly the cost of calling a Python function from C.

39.7 Why does my external Python process (which I call from Open RV) now behave differently?

This is probably caused by the fact that RV modifies the PYTHONPATH to incorporate the Python plug-in folder and RV's python standard libraries to run. Forked processes will inherit the PYTHONPATH. If you are using QProcess to launch the external process you can call `QProcess.setEnvironment()` to set the PYTHONPATH before calling `QProcess.start()`.

CHAPTER

FORTY

CHAPTER 5 - EVENT HANDLING

Aside from rendering, the most important function of the UI is to handle events. An event can be triggered by any of the following:

- The mouse pointer moved or a button on the mouse was pressed
- A key on the keyboard was pressed or released
- The window needs to be re-rendered
- A file being watched was changed
- The user became active or inactive
- A supported device (like the apple remote control) did something
- An internal event like a new source or session being created has occurred

Each specific event has a name may also have extra data associated with it in the form of an event object. To see the name of an event (at least for keyboard and mouse pointer events) you can select the Help → Describe… which will let you interactively see the event name as you hit keys or move the mouse. You can also use Help → Describe Key.. to see what a specific key is bound to by pressing it.Table *5.1* shows the basic event type prefixes.

Event Prefix	Description
key-down	Key is being pressed on the keyboard
key-up	Key is being released on the keyboard
pointer	The mouse moved, button was pressed, or the pointer entered (or left) the window
dragdrop	Something was dragged onto the window (file icon, etc)
render	The window needs updating
user	The user's state changed (active or inactive, etc)
remote	A network event

Table 5.1: Event Prefixes for Basic Device Events

When an event is generated in RV, the application will look for a matching event name in its bindings. The bindings are tables of functions which are assigned to certain event names. The tables form a stack which can be pushed and popped. Once a matching binding is found, RV will execute the function.When receiving an event, all of the relevant information is in the Event object. This object has a number of methods which return information depending on the kind of event.

Method	Events	Description
pointer (Vec2;)	pointer-* dragdrop *	Returns the location of the pointer relative to the view.
relativePointer (Vec2;)	pointer-* dragdrop *	Returns the location of the pointer relative to the current widget or view if there is none.
reference (Vec2;)	pointer-* dragdrop *	Returns the location of initial button mouse down during dragging.
domain (Vec2;)	pointer-* render-* dragdrop *	Returns the size of the view.
subDomain (Vec2;)	pointer-* render-* dragdrop *	Returns the size of the current widget if there is one. relativePointer() is positioned in the subDomain().
buttons (int;)	pointer-* dragdrop *	Returns an int or'd from the symbols: Button1, Button2, and Button3.
modifiers (int;)	pointer-* key-* dragdrop *	Returns an int or'd from the symbols: None, Shift, Control, Alt, Meta, Super, CapLock, NumLock, ScrollLock.
key (int;)	key-*	Returns the "keysym" value for the key as an int
name (string;)	any	Returns the name of the event
contents (string;)	internal eventsdra *	Returns the string content of the event if it has any. This is normally the case with internal events like new-source, new-session, etc. Pointer, key, and other device events do not have a contents() and will throw if it's called on them. Drag and drop events return the data associated with them. Some render events have contents() indicating the type of render occurring.
contentsArray (string[];)	internal events	Same as contents(), but in the case of some internal events ancillary information may be present which can be used to avoid calling additional commands.
sender (string;)	any	Returns the name of the sender
contentType (int;)	dragdrop *	Returns an int describing the contents() of a drag and drop event. One of: UnknownObject, BadObject, FileObject, URLObject, TextObject.
timeStamp (float;)	any	Returns a float value in seconds indicating when the event occurred
reject (void;)	any	Calling this function will cause the event to be send to the next binding found in the event table stack. Not calling this function stops the propagation of the event.
setReturnCon	internal events	Events which have a contents may also have return content. This is used by the remote network events which can have a response.

Table 5.2:Event Object Methods. Python methods have the same names and return the same value types.

40.1 5.1 Binding an Event

In Mu (or Python) you can bind an event using any of the bind() functions. The most basic version of bind() takes the name of the event and a function to call when the event occurs as arguments. The function argument (which is called when the event occurs) should take an Event object as an argument and return nothing (void). Here's a function that prints hello in the console every time the ``j" key is pressed:

> **Note:** If this is the first time you've seen this syntax, it's defining a Mu function. The first two characters \: indicate a function definition follows. The name comes next. The arguments and return type are contained in the parenthesis. The first identifier is the return type followed by a semicolon, followed by an argument list.

E.g, : add (int; int a, int b) { return a + b; }

```
\: my_event_function (void; Event event)
{
    print("Hello!\n");
}

bind("key-down--j", my_event_function);
```

or in Python:

```
 def my_event_function (event):
    print ("Hello!")

bind("default", "global", "key-down--j", my_event_function);
```

There are more complicated bind() functions to address binding functions in specific event tables (the Python example above is using the most general of these). Currently RV's user interface has one default global event table an couple of other tables which implement the parameter edit mode and help modes.Many events provide additional information in the event object. Our example above doesn't even use the event object, but we can change it to print out the key that was pressed by changing the function like so:

```
 \: my_event_function (void; Event event)
{
    let c = char(event.key());
    print("Key pressed = %c\n" % c);
}
```

or in Python:

```
 def my_event_function (event):
    c = event.key()
    print ("Key pressed = %s\n" % c)
```

In this case, the Event object's key() function is being called to retrieve the key pressed. To use the return value as a key it must be cast to a char. In Mu, the char type holds a single unicode character. In Python, a string is unicode. See the section on the Event class to find out how to retrieve information from it. At this point we have not talked about *where* you would bind an event; that will be addressed in the customization sections.

40.2 5.2 Keyboard Events

There are two keyboard events: key-down and key-up. Normally the key-down events are bound to functions. The key-up events are necessary only in special cases.The specific form for key down events is key-down– *something* where *something* uniquely identifies both the key pressed and any modifiers that were active at the time.So if the ``a" key was pressed the event would be called: key-down–a. If the control key were held down while hitting the ``a" key the event would be called key-down–control–a.There are five modifiers that may appear in the event name: alt, caplock, control, meta, numlock, scrolllock, and shift in that order. The shift modifier is a bit different than the others. If a key is pressed with the shift modifier down and it would result in a different character being generated, then the shift modifier will not appear in the event and instead the result key will. This may sound complicated but these examples should explain it:For control + shift + A the event name would be key-down–control–A. For the ``*" key (shift + 8 on American keyboards) the event would be key-down–*. Notice that the shift modifier does not appear in any of these. However, if you hold down shift and hit enter on most keyboards you will get key-down–shift–enter since there is no character associated with that key sequence.Some keys may have a special name (like enter above). These will typically be spelled out. For example pressing the ``home" key on most keyboards will result in the event key-down–home. The only way to make sure you have the correct event name for keys is to start RV and use the Help → Describe… facility to see the true name. Sometimes keyboards will label a key and produce an unexpected event. There will be some keyboards which will not produce an event all for some keys or will produce a unicode character sequence (which you can see via the help mechanism).

40.3 5.3 Pointer (Mouse) Events

The mouse (called pointer from here on) can produce events when it is moved, one of its buttons is pressed, an attached scroll wheel is rotated, or the pointer enters or leaves the window.The basic pointer events are move, enter, leave, wheelup, wheeldown, push, drag, and release. All but enter and leave will also indicate any keyboard modifiers that are being pressed along with any buttons on the mouse that are being held down. The buttons are numbered 1 through 5. For example if you hold down the left mouse button and movie the mouse the events generated are:

```
pointer-1--push
pointer-1--drag
pointer-1--drag
...
pointer-1-release
```

Pointer events involving buttons and modifiers always come in there parts: push, drag and release. So for example if you press the left mouse, move the mouse, press the shift key, move the mouse, release everything you get:

```
pointer-1--push
pointer-1--drag
pointer-1--drag
...
pointer-1-release
pointer-1--shift--push
pointer-1--shift--drag
pointer-1--shift--drag
...
pointer-1--shift--release
```

Notice how the first group without the shift is released before starting the second group with the shift even though you never released the mouse button. For any combination of buttons and modifiers, there will be a push-drag-release sequence that is cleanly terminated.It is also possible to hold multiple mouse buttons and modifiers down at the same time. When multiple buttons are held (for example, button 1 and 2) they are simply both included (like the modifiers) so for buttons 1 and 2 the name would be pointer-1-2–push to start the sequence.The mouse wheel behaves more like a button: when the wheel moves you get only a wheelup or wheeldown event indicating which direction the wheel was rotated. The buttons and modifiers will be applied to the event name if they are held down. Usually the motion of the wheel on a mouse will not be smooth and the event will be emitted whenever the wheel ``clicks". However, this is completely a function of the hardware so you may need to experiment with any particular mouse.There are three more pointer events that can be generated. When the mouse moves with no modifiers or buttons held down it will generate the event pointer–move. When the pointer enters the view pointer–enter is generated and when it leaves pointer–leave. Something to keep in mind: when the pointer leaves the view and the device is no longer in focus on the RV window, any modifiers or buttons the user presses will not be known to RV and will not generate events. When the pointer returns to the view it may have modifiers that became active when out-of-focus. Since RV cannot know about these modifiers and track them in a consistent manner (at least on X Windows) RV will assume they do not exist.Pointer events have additional information associated with them like the coordinates of the pointer or where a push was made. These will be discussed later.

40.4 5.4 The Render Event

The UI will get a render event whenever it needs to be updated. When handling the render event, a GL context is set up and you can call any GL function to draw to the screen. The event supplies additional information about the view so you can set up a projection.At the time the render event occurs, RV has already rendered whatever images need to be displayed. The UI is then called in order to add additional visual objects like an on-screen widget or annotation.Here's a render function that draws a red polygon in the middle of the view right on top of your image.Listing 5.1:Example Render Function

```
 \: my_render (void; Event event)
{
    let domain = event.domain(),
        w      = domain.x,
        h      = domain.y,
        margin = 100;

    use gl;
    use glu;

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, w, 0, h);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Big red polygon
    glColor(Color(1,0,0,1));
    glBegin(GL_POLYGON);
    glVertex(margin, margin);
    glVertex(w-margin, margin);
    glVertex(w-margin, h-margin);
    glVertex(margin, h-margin);
    glEnd();
}
```

Note that for Python, you will need to use the PyOpenGL module or bind the symbols in the gl Mu module manually in order to draw in the render event.The UI code already has a function called render() bound the render event; so this function basically turns off existing UI rendering.

40.5 5.5 Remote Networking Events

RV's networking generates a number of events indicating the status of the network. In addition, once a connection has been established, the UI may generate sent to remote programs, or remote programs may send events to RV. These are typically uniquely named events which are specific to the application that is generating and receiving them.For example the sync mechanism generates a number of events which are all named remote-sync-something.

40.6 5.6 Internal Events

Some events will originate from RV itself. These include things like new-source or new-session which include information about what changed. The most useful of these is new-source which can be used to manage color and other image settings between the time a file is loaded and the time it is first displayed. (See Color Management Section). Other internal events are functional, but are placeholders which will become useful with future features.The current internal events are listed in table *5.3* .

Event	Event.(data/contents)	Ancillary Data (contentsArray)	Description
render			Main view render
pre-render			Before rendering
post-render			After rendering
per-render-event-processing			Qt Event processing betw
layout			Main view layout used to
new-source	nodename;;RVSource;;filename		**DEPRECATED** A new s
source-group-complete	group nodename;;action_type		A new or modified source
source-modified	nodename;;RVSource;;filename		An existing source was ch
media-relocated	nodename;;oldmedia;;newmedia		A movie, image sequence
source-media-set	nodename;;tag		
before-session-read	filename		Session file is about to be
after-session-read	filename		Session file was read
before-session-write	filename		Session file is about to be
after-session-write	filename		Session file was just writt
before-session-write-copy	filename		A copy of the session is a
after-session-write-copy	filename		A copy of a session was ju
before-session-deletion			The session is about to be
before-graph-view-change	nodename		The current view node is
after-graph-view-change	nodename		The current view node ch
new-node	nodename		A new view node was cre
graph-new-node	nodename	nodename protocol version groupname	A new node of any kind w
before-progressive-loading			Loading will start
after-progressive-loading			Loading is complete (sent
graph-layer-change			**DEPRECATED** use after
frame-changed			The current frame change
fps-changed			Playback FPS changed
play-start			Playback started
play-stop			Playback stopped
incoming-source-path	infilename;;tag		A file was selected by the
missing-image			An image could not be lo
cache-mode-changed	buffer or region or off		Caching mode changed
view-size-changed			The viewing area size cha
new-in-point	frame		The in point changed
new-out-point	frame		The out point changed

Table 1 – continued from previous page

Event	Event.(data/contents)	Ancillary Data (contentsArray)	Description
before-source-delete	nodename		Source node will be delet
after-source-delete	nodename		Source node was deleted
before-node-delete	nodename		View node will be deleted
after-node-delete	nodename		View node was deleted
after-clear-session			The session was just clear
after-preferences-write			Preferences file was writte
state-initialized			Mu/Python init files read
session-initialized			All modes toggled, comm
realtime-play-mode			Playback mode changed t
play-all-frames-mode			Playback mode changed t
before-play-start			Play mode will start
mark-frame	frame		Frame was marked
unmark-frame	frame		Frame was unmarked
pixel-block	Event.data()		A block of pixels was rec
graph-state-change			A property in the image p
graph-node-inputs-changed	nodename		Inputs of a top-level node
range-changed			The time range changed
narrowed-range-changed			The narrowed time range
margins-changed	left right top bottom		View margins changed
view-resized	old-w new-w	old-h new-h	
preferences-show			Pref dialog will be shown
preferences-hide			Pref dialog was hidden
read-cdl-complete	cdl_filename;;cdl_nodename		CDL file has been loaded
read-lut-complete	lut_filename;;lut_nodename		LUT file has been loaded
remote-eval	code		Request to evaluate exter
remote-pyeval	code		Request to evaluate exter
remote-pyexec	code		Request to execute extern
remote-network-start			Remote networking starte
remote-network-stop			Remote networking stopp
remote-connection-start	contact-name		A new remote connection
remote-connection-stop	contact-name		A remote connection has
remote-contact-error	contact-name		A remote connection erro

Table 5.3:Internal Events

40.6.1 5.6.1 File Changed Event

It is possible to watch a file from the UI. If the watched file changes in any way (modified, deleted, moved, etc) a file-changed event will be generated. The event object will contain the name of the watched file that changed. A function bound to file-changed might look something like this:

```
\: my_file_changed (void; Event event)
{
    let file = event.contents();
    print("%s changed on disk\n" % file);
}
```

In order to have a file-changed event generated, you must first have called the command function watchFile().

40.6.2 5.6.2 Incoming Source Path Event

This event is sent when the user has selected a file or sequence to load from the UI or command line. The event contains the name of the file or sequence. A function bound to this event can change the file or sequence that RV actually loads by setting the return contents of the event. For example, you can cause RV to check and see if a single file is part of a larger sequence and if so load the whole sequence like so:

```
 \: load_whole_sequence (void; Event event)
{
    let file        = event.contents(),
        (seq,frame) = sequenceOfFile(event.contents());

    if (seq != "") event.setReturnContent(seq);
}

bind("incoming-source-path", load_whole_sequence);
```

or in Python:

```
 def load_whole_sequence (event):

    file = event.contents();
    (seq,frame) = rv.commands.sequenceOfFile(event.contents());

    if seq != "":
         event.setReturnContent(seq);


bind("default", "global", "incoming-source-path", load_whole_sequence, "Doc string");
```

40.6.3 5.6.3 Missing Images

Sometimes an image is not available on disk when RV tries to read. This is often the case when looking at an image sequence while a render or composite is ongoing. By default, RV will find a nearby frame to represent the missing frame if possible. The missing-image event will be sent once for each image which was expected but not found. The function bound to this event can render information on on the screen indicating that the original image was missing. The default binding display a message in the feedback area.The missing-image event contains the domain in which rendering can occur (the window width and height) as well as a string of the form ``frame;source" which can be obtained by calling the contents() function on the event object.The default binding looks like this:

```
 \: missingImage (void; Event event)
{
    let contents = event.contents(),
        parts = contents.split(";"),
        media = io.path.basename(sourceMedia(parts[1])._0);

    displayFeedback("MISSING: frame %s of %s"
                     % (parts[0], media), 1, drawXGlyph);
}

bind("missing-image", missingImage);
```

CHAPTER

FORTYONE

CHAPTER 6 - OPEN RV FILE FORMAT

The RV file format (.rv) is a text GTO file. GTO is an open source file format which stores arbitrary data — mostly for use in computer graphics applications. The text GTO format is meant to be simple and human readable. It's helpful to have familiarized yourself with the GTO documentation before reading this section. The documentation should come with RV, or you can read it on line at the *GTO*.

41.1 6.1 How Open RV Uses GTO

RV defines a number of new GTO object protocols (types of objects). The GTO file is made up of objects, which contain components, which contain properties where the actual data resides. RV's use of the format is to store nodes in an image processing graph as GTO objects. How the nodes are connected is determined by RV and is not currently arbitrary so there are no connections between the objects stored in the file.Some examples of RV object types include:

- The **RVSession** object (one per file) which stores information about the session. This includes the full frame range, currently marked frames, the playback FPS, and whether or not to use real time playback among other things.
- **RVLayoutGroup** , **RVFolderGroup, RVSwitchGroup, RVSourceGroup** , **RVRetimeGroup** , **RVStack-Group** , **RVDisplayGroup** and **RVSequenceGroup** nodes which form the top-level of the image processing graph.
- One or more **RVFileSource** objects each within an **RVSourceGroup** which specify all of the media (movies, audio files, image sequences) which are available in the session.
- Color correction objects like **RVColor** nodes which are members of **RVSourceGroup** objects.
- Image format objects like **RVFormat** or **RVChannelMap** which are also members of **RVSourceGroup** objects.
- An **RVDisplayColor** object (one per file) which indicates monitor gamma, any display LUT being used (and possibly the actual LUT data) which is part of the **RVDisplayGroup** .
- A **connections** object which contains connections between the top-level group nodes. The file only stores the top-level connections — connections within group nodes are determined by the group node at runtime.

Normally, RV will write out all objects to the session file, but it does not require all of them to create a session from scratch. For example, if you have a file with a single RVFileSource object in it, RV will use that and create default objects for everything else. So when creating a file without RV, it's not a bad idea to only include information that you need instead of replicating the output of RV itself. (This helps make your code future proof as well).The order in which the objects appear in the file is not important. You can also include information that RV does not know about if you want to use the file for other programs as well.

41.2 6.2 Naming

The names of objects in the session are not visible to the user, however they must follow certain naming naming conventions. There is a separate user interface name for top level nodes which the user does see. The user name can be set by creating a string property on a group node called ui.name.

- If the object is a group node type other than a source or display group its name can be anything, but it must be unique.
- If there is an **RVDisplayGroup** in the file it must be called displayGroup.
- If the object is a member of a group its name should have the pattern: groupName_nodeName where groupName is the name of the group the node is a member of. The nodeName can be anything, but RV will use the type name in lowercase without the "RV" at the front.
- If the object is a **RVFileSourceGroup** or **RVImageSourceGroup** it should be named sourceGroupXXXXXX where the Xs form a six digit zero padded number. RV will create source groups starting with number 000000 and increment the number for each new source group it creates. If a source group is deleted, RV may reuse its number when creating a new group.
- The **connection** object should be named connections.
- The **RVSession** object can have any name.

41.3 6.3 A Simple Example

The simplest RV file you can create is a one which just causes RV to load a single movie file or image. This example loads a QuickTime file called "test.mov" from the directory RV was started in:

```
 GTOa (3)

sourceGroup000000_source : RVFileSource (0)
{
    media
    {
        string movie =  "test.mov"
    }
}
```

The first line is required for a text GTO file: it indicates the fact that the file is text format and that the GTO file version is 3. All of the other information (the frame ranges, etc) will be automatically generated when the file is read. By default RV will play the entire range of the movie just as if you dropped it into a blank RV session in the UI. You should name the first **RVFileSource** object sourceGroup000000_source and the second sourceGroup000001_source and the third sourceGroup000002_source, and so on. Eventually we'll want to make an EDL which will index the source objects so the names mostly matter (but not the order in which they appear).Now suppose we have an image sequence instead of a movie file. We also have an associated audio file which needs to be played with it. This is a bit more complicated, but we still only need to make a single **RVFileSource** object:Here we've got test.#.dpx as an image layer and soundtrack.aiff which is an audio layer.

```
 GTOa (3)

sourceGroup000000_source : RVFileSource (0)
{
    media
```

(continues on next page)

(continued from previous page)

```
    {
        string movie = [ "test.#.dpx" "soundtrack.aiff" ]
    }

    group
    {
        float fps =  24
        float volume =  0.5
        float audioOffset = 0.1
    }
}
```

You can have any number of audio and image sequence/movie files in the movie list. All of them together create the output of the **RVFileSource** object. If we were creating a stereo source, we might have left.#.dpx and right.#.dpx instead of test.#.dpx. When there are multiple image layers the first two default to the left and right eyes in the order in which they appear. You can change this behavior per-source if necessary. The format of the various layers do not need to match.The **group** component indicates how all of media should be combined. In this case we've indicated the FPS of the image sequence, the volume of all audio for this source and an audio slip of 0.1 (one tenth) of a second. Keep in mind that FPS here is for the image sequence(s) in the source it has nothing to do with the playback FPS!. The playback FPS is independent of the input sources frame rate.

41.3.1 Aside: What is the FPS of an RVFileSource Object Anyway?

If you write out an RV file from RV itself, you'll notice that the group FPS is often 0! This is a special cookie value which indicates that the FPS should be taken from the media. Movie file formats like QuickTime or AVI store this information internally. So RV will use the frame rate from the media file as the FPS for the source.However, image sequences typically do not include this information (OpenEXR files are a notable exception).. When you start RV from the command line it will use the playback FPS as a default value for any sources created. If there is no playback FPS on startup either via the command line or preferences, it will default to 24 fps. So it's not a bad idea to include the group FPS when creating an RV file yourself when you're using image sequences. If you're using a movie file format you should either use 0 for the FPS or not include it and let RV figure it out.What happens when you get a mismatch between the source FPS and the playback FPS? If there's no audio, you won't notice anything; RV always plays back every frame in the source regardless of the source FPS. But if you have audio layers along with your image sequence or if the media is a movie file, you will notice that the audio is either compressed or expanded in order to maintain synchronization with the images.This is a very important thing to understand about RV: it will always playback every image no matter what the playback FPS is set to; and it will always change the audio to compensate for that and maintain synchronization with the images.So the source FPS is really important when there is audio associated with the images.

41.4 6.4 Per-Source and Display Color Settings and LUT Files

If you want to include per-source color information – such as forcing a particular LUT to be applied or converting log to linear – you can include only the additional nodes you need with only the parameters that you wish to pass in. For example, to apply a file LUT to the first source (e.g. sourceGroup000000_source) you can create an **RVColor** node similarly named sourceGroup000000_color.

```
 sourceGroup000000_color : RVColor (1)
{
    lut
    {
        string file = "/path/to/LUTs/log2sRGB.csp"
```

(continues on next page)

(continued from previous page)

```
        int active = 1
    }
}
```

This is a special case in the rv session file: you can refer to a LUT by file. If you have a new-source event bound to a function which modifies incoming color settings based on the image type, any node properties in your session file override the default values created there. To state it another way: values you omit in the session file still exist in RV and will take on whatever values the function bound to new-source made for them. To ensure that you get exactly the color you want you can specify all of the relevant color properties in the **RVColor, RVLinearize,** and **RVDisplayColor** nodes:

```
 sourceGroup000000_colorPipeline_0 : RVColor (2)
{
    color
    {
        int invert = 0
        float[3] gamma = [ [ 1 1 1 ] ]
        string lut = "default"
        float[3] offset = [ [ 0 0 0 ] ]
        float[3] scale = [ [ 1 1 1 ] ]
        float[3] exposure = [ [ 0 0 0 ] ]
        float[3] contrast = [ [ 0 0 0 ] ]
        float saturation = 1
        int normalize = 0
        float hue = 0
        int active = 1
    }

    CDL
    {
        float[3] slope = [ [ 1 1 1 ] ]
        float[3] offset = [ [ 0 0 0 ] ]
        float[3] power = [ [ 1 1 1 ] ]
        float saturation = 1
        int noClamp = 0
    }

    luminanceLUT
    {
        float lut = [ ]
        float max = 1
        int size = 0
        string name = ""
        int active = 0
    }

    "luminanceLUT:output"
    {
        int size = 256
    }
}
```

(continues on next page)

(continued from previous page)

```
sourceGroup000000_tolinPipeline_0 : RVLinearize (1)
{
    lut
    {
        float[16] inMatrix = [ [ 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 ] ]
        float[16] outMatrix = [ [ 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 ] ]
        float lut = [ ]
        float prelut = [ ]
        float scale = 1
        float offset = 0
        string type = "Luminance"
        string name = ""
        string file = ""
        int size = [ 0 0 0 ]
        int active = 0
    }

    color
    {
        string lut = "default"
        int alphaType = 0
        int logtype = 0
        int YUV = 0
        int invert = 0
        int sRGB2linear = 1
        int Rec709ToLinear = 0
        float fileGamma = 1
        int active = 1
        int ignoreChromaticities = 0
    }

    cineon
    {
        int whiteCodeValue = 0
        int blackCodeValue = 0
        int breakPointValue = 0
    }

    CDL
    {
        float[3] slope = [ [ 1 1 1 ] ]
        float[3] offset = [ [ 0 0 0 ] ]
        float[3] power = [ [ 1 1 1 ] ]
        float saturation = 1
        int noClamp = 0
    }
}

defaultOutputGroup_colorPipeline_0 : RVDisplayColor (1)
{
    lut
    {
```

(continues on next page)

(continued from previous page)

```
        float[16] inMatrix = [ [ 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 ] ]
        float[16] outMatrix = [ [ 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 ] ]
        float lut = [ ]
        float prelut = [ ]
        float scale = 1
        float offset = 0
        string type = "Luminance"
        string name = ""
        string file = ""
        int size = [ 0 0 0 ]
        int active = 0
    }

    color
    {
        string lut = "default"
        string channelOrder = "RGBA"
        int channelFlood = 0
        int premult = 0
        float gamma = 1
        int sRGB = 0
        int Rec709 = 0
        float brightness = 0
        int outOfRange = 0
        int dither = 0
        int active = 1
    }

    chromaticities
    {
        int active = 0
        int adoptedNeutral = 0
        float[2] white = [ [ 0.3127 0.329 ] ]
        float[2] red = [ [ 0.64 0.33 ] ]
        float[2] green = [ [ 0.3 0.6 ] ]
        float[2] blue = [ [ 0.15 0.06 ] ]
        float[2] neutral = [ [ 0.3127 0.329 ] ]
    }
}
```

The above example values assume default color pipeline slots for a single source session. Please see section *12.3* to learn more about the specific color pipeline groups.

41.5 6.5 Information Global to the Session

Now let's add an **RVSession** object with in and output points. The session object should be called rv in this version. There should only be one **RVSession** object in the file.

> **Note:** From now on we're just going to show fragments of the file and assume that you can put them all together in your text editor.

```
rv : RVSession (1)
{
    session
    {
        string viewNode = "defaultSequence"
        int marks = [ 1 20 50 80 100 ]
        int[2] range = [ [ 1 100 ] ]
        int[2] region = [ [ 20 50 ] ]
        float fps =  24
        int realtime =  1
        int currentFrame =  30
    }
}
```

Assuming this was added to the top of our previous file with the source in it, the session object now indicates the frame range (1-100) and an in and out region (20-50) which is currently active. Frames 1, 20, 50, 80, and 100 are marked and the default frame is frame 30 when RV starts up. The realtime property is a flag which indicates that RV should start playback in real time mode. The view node indicates what will be viewed in the session when the file is opened.Note that it's usually a good idea to skip the frame range boundaries unless an EDL is also specified in the file (which is not the case here). RV will figure out the correct range information from the source media. If you force the range information to be different than the source media's you may get unexpected results. The marks and range can also be stored on each viewable top-level object. For example the defaultLayout and defaultSequence can have different marks and in and out points:

```
defaultStack : RVStackGroup (1)
{
    session
    {
        float fps = 24
        int marks = [ ]
        int[2] region = [ [ 100 200 ] ]
        int frame = 1
    }
}
```

If a group has a session component than the contents can provide an in/out region, marks, playback fps and a current frame. When the user views the group node these values will become inherited by the session.

41.6 6.6 The Graph

Internally, RV holds a single image processing graph per session which is represented in the session file. The graph can have multiple nodes which determine how the sources are combined. These are the top-level nodes and are always group nodes. The user can create new top-level nodes (like sequences, stacks, layouts, retimings, etc). So the inputs for each node need to be stored in order to reproduce what the user created. The connections between the top-level group nodes are stored in the connections object and includes a list of the top level nodes. For example, this is what RV will write out for a session with a single source in it:

```
 connections : connection (1)
{
    evaluation
    {
        string lhs = [ "sourceGroup000000"
                       "sourceGroup000000"
                       "sourceGroup000000" ]

        string rhs = [ "defaultLayout"
                       "defaultSequence"
                       "defaultStack" ]
    }

    top
    {
        string nodes = [ "sourceGroup00000",
                         "defaultLayout",
                         "defaultStack",
                         "defaultSequence" ]
    }
}
```

The connections should be interpreted as arrows between objects. The lhs (left hand side) is the base of the arrow. The rhs (right hand side) is the tip. The base and tips are stored in separate properties. So in the case, the file has three connections:

Note: RV may write out a connection to the display group as well. However, that connection is redundant and may be overridden by the value of the view node property in the RVSession.

1. sourceGroup000000 → defaultLayout
2. sourceGroup000000 → defaultSequence
3. sourceGroup000000 → defaultStack

The nodes property, if it exists, will determine which nodes are considered top level nodes. Otherwise, nodes which include connections and nodes which have user interface name are considered top level.

41.6.1 6.6.1 Default Views

There are three default views that are always created by RV: the default stack, sequence, and layout. Whenever a new source is added by the user each of these will automatically connect the new source as an input. When a new viewing node is created (a new sequence, stack, layout, retime) the default views will not add those —- only sources are automatically added.When writing a .rv file you can co-opt these views to rearrange or add inputs or generate a unique EDL but it's probably a better idea to create a new one instead; RV will never automatically edit a sequence, stack, layout, etc, that is not one of the default views.

41.7 6.7 Creating a Session File for Custom Review

One of the major reasons to create session files outside of RV is to automatically generate custom review workflows. For example, if you want to look at an old version of a sequence and a new version, you might have your pipeline output a session file with both in the session and have pre-constructed stacked views with wipes and a side-by-side layout of the two sequences.To start with lets look at creating a session file which creates a unique sequence (not the default sequence) with plays back sources in a particular order. In this case, no EDL creation is necessary — we only need to supply the sequence with the source inputs in the correct order. This is analogous to the user reordering the inputs on a sequence in the user interface.This file will have an RVSequenceGroup object as well as the sources. Creating sources is covered above so we'll skip to the creation of the RVSequenceGroup. For this example we'll assume there are three sources and that they all have the same FPS (so no retiming is necessary). We'll let RV handle creation of the underlying RVSequence and its EDL and only create the group:

```
// define sources ...

reviewSequence : RVSequenceGroup (1)
{
    ui { string name = "For Review" }
}

connections : connection (1)
{
    evaluation
    {
        string lhs = [ "sourceGroup000002"
                       "sourceGroup000000"
                       "sourceGroup000001" ]

        string rhs = [ "reviewSequence"
                       "reviewSequence"
                       "reviewSequence" ]
    }
}
```

RV will automatically connect up the default views so we can skip their inputs in the connections object for clarity. In this case, the sequence is connected up so that by default it will play sourceGroup000002 followed by sourceGroup000000 followed by sourceGroup000001 because the default EDL of a sequence just plays back the inputs in order. Note that for basic ordering of playback, no EDL creation is necessary. We could also create additional sequence groups with other inputs. Also note the use of the UI name in the sequence group.Of course, the above is not typical in a production environment. Usually there are handles which need to (possibly) be edited out. There are two ways to do this with RV: either set the cut points in each source and tell the sequence to use them, or create an EDL in the sequence which excludes the handles.To start with we'll show the first method: set the cut points. This method is easy to implement and the sequence interface has a button on it that lets the user toggle the in/out cuts on/off in realtime. If the user reorders the sequence, the cuts will be maintained. When using this method any sequence in the session can be made to use the same cut information — it propagates down from the source to the sequence instead of being stored for each sequence.Setting the cut in/out points requires adding a property to the RVFileSource objects and specifying the in and out frames:

```
sourceGroup000000_source : RVFileSource (1)
{
    media { string movie = "shot00.mov" }

    cut
    {
        int in = 8
        int out = 55
    }
}

sourceGroup000001_source : RVFileSource (1)
{
    media { string movie = "shot01.mov" }

    cut
    {
        int in = 5
        int out = 102
    }
}

sourceGroup000002_source : RVFileSource (1)
{
    media { string movie = "shot02.mov" }

    cut
    {
        int in = 3
        int out = 22
    }
}
```

Finally, the most flexibly way to control playback is to create an EDL. The EDL is stored in an RVSequence node which is a member of the RVSequenceGroup. Whenever an RVSequenceGroup is created, it will create a sequence node to hold the EDL. If you are not changing the default values or behavior of the sequence node it's not necessary to specify it in the file. In this case, however we will be creating a custom EDL.

41.7.1 6.7.1 RVSequence

The sequence node can be in one of two modes: auto EDL creation or manual EDL creation. This is controlled by the mode.autoEDL property. If the property is set to 1 then the sequence will behave like so:

- If a new input is connected, the existing EDL is erased and a new EDL is created.
- Each input of the sequence will have a cut created for it in the order that they appear. If mode.useCutInfo is set, the sequence will use the cut information coming from the input to determine the cut in the EDL. Otherwise it will use the full range of the input.
- If cut info changes on any input to the sequence, the EDL will be adjusted automatically.

When auto EDL is not on, the sequence node behavior is not well-defined when the inputs are changed. In future, we'd like to provide more interface for EDL modification (editing) but for the moment, a custom EDL should only be created programmatically in the session file.For this next example, we'll use two movie files: a.mov and b.mov. They have audio so there's nothing interesting about their source definitions: just the media property with the name of the movie . They are both 24 fps and the playback will be as well:

Note: The example RV file has fewer line breaks than one which RV would write. However, it's still valid.

```
 GTOa (3)

rv : RVSession (2)
{
    session
    {
        string viewNode = "mySequence"
    }
}

sourceGroup000000_source : RVFileSource (0) { media { string movie =  "a.mov" } }
sourceGroup000001_source : RVFileSource (0) { media { string movie =  "b.mov" } }

connections : connection (1)
{
    evaluation
    {
        string lhs = [ "sourceGroup000000"
                       "sourceGroup000001" ]
        string rhs = [ "mySequence"
                       "mySequence" ]
    }
}

mySequence : RVSequenceGroup (0)
{
    ui
    {
        string name = "GUI Name of My Sequence"
    }
}

mySequence_sequence : RVSequence (0)
{
    edl
```

(continues on next page)

(continued from previous page)

```
    {
        int frame  = [  1 11 21 31 41 ]
        int source = [  0  1  0  1  0 ]
        int in     = [  1  1 11 11  0 ]
        int out    = [ 10 10 20 20  0 ]
    }

    mode
    {
        int autoEDL = 0
    }
}
```

The source property indexes the inputs to the sequence node. So index 0 refers to sourceGroup000000 and index 1 refers to sourceGroup000001. This EDL has four edits which are played sequentially as follows:

1. a.mov, frames 1-10
2. b.mov, frames 1-10
3. a.mov, frames 11-20
4. b.mov, frames 11-20

You can think of the properties in the sequence as forming a transposed matrix in which the properties are columns and edits are rows as in *6.1*. Note that there are only 4 edits even though there are 5 rows in the matrix. The last edit is really just a boundary condition: it indicates how RV should handle frames past the end of the EDL. To be well formed, an RV EDL needs to include this.Note that the in frame and out frame may be equal to implement a "held" frame.

	global start frame	source	in	out
edit #1	1	a.mov	1	10
edit #2	11	b.mov	1	10
edit #3	21	a.mov	11	20
edit #4	31	b.mov	11	20
past end	41	a.mov	0	0

Table 6.1:EDL as Matrix

41.7.2 6.7.2 RVLayoutGroup and RVStackGroup

The stack and layout groups can be made in a similar manner to the above. The important thing to remember is the inputs for all of these must be specified in the connections object of the file. Each of these view types uses the input ordering; in the case of the stack it determines what's on top and in the case of the layout it determines how automatic layout will be ordered.

41.7.3 6.7.3 RVOverlay

Burned in metadata can be useful when creating session files. Shot status, artist, name, sequence, and other static information can be rendered on top of the source image directly by RV's render. Figure *6.1* shows an example of metadata rendered by the RVOverlay node.

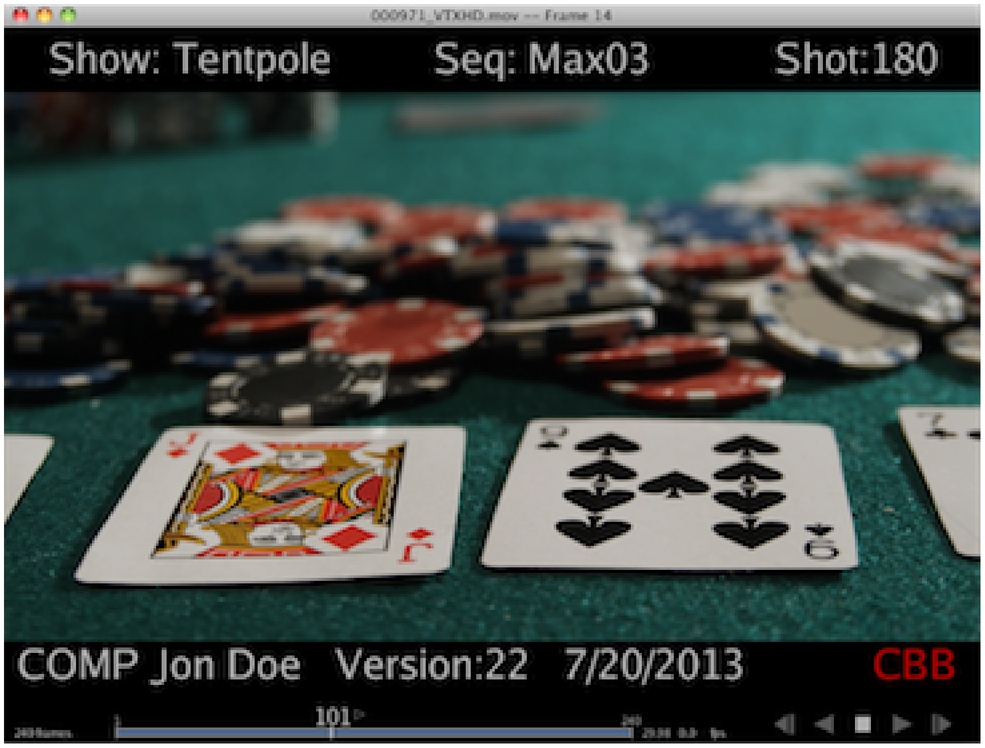


Figure 6.1:Metadata Rendered By RVOverlay Node From Session File

Each RVSourceGroup can have an RVOverlay node. The RVOverlay node is used for matte rendering by user interface, but it can do much more than that. The RVOverlay node currently supports drawing arbitrary filled rectangles and text in addition to the mattes. The text and filled rectangle are currently limited to static shapes and text. Text and rectangles rendered in this fashion are considered part of the image by RV. If you pass a session file with an active RVOverlay node to rvio it will render the overlay the same way RV would. This is completely independent of any rvio overlay scripts which use a different mechanism to generate overlay drawings and text.Figure *6.2* shows an example which draws three colored boxes with text starting at each box's origin.

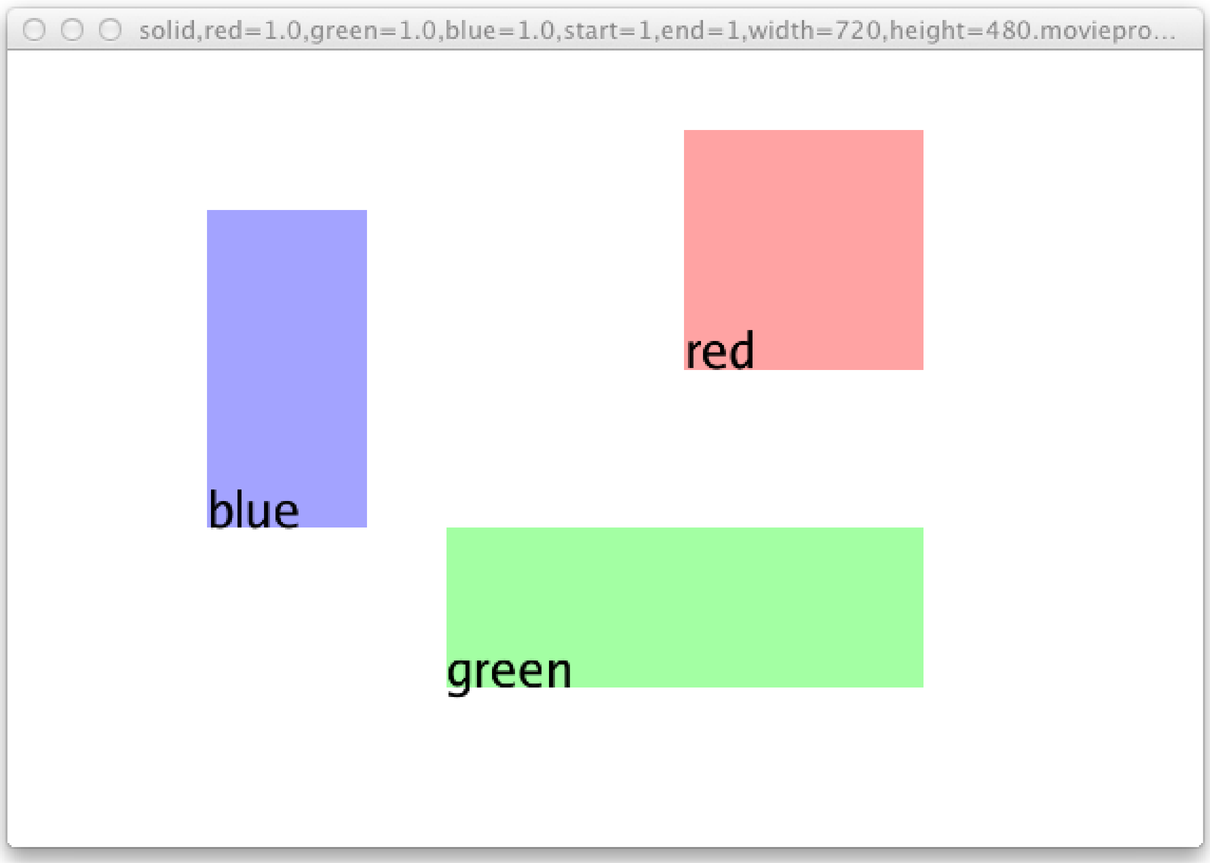

solid,red=1.0,green=1.0,blue=1.0,start=1,end=1,width=720,height=480.moviepro...
red
blue
green

Figure 6.2:RVOverlay Node Example

The session file used to create the example contains a movieproc source (white 720x480 image) with the overlay rendered on top of it. Note that the coordinates are normalized screen coordinates relative to the source image:

```
 GTOa (3)

sourceGroup1_source : RVFileSource (1)
{
    media
    {
        string movie = "solid,red=1.0,green=1.0,blue=1.0,start=1,end=1,width=720,
↪height=480.movieproc"
    }
}

sourceGroup1_overlay : RVOverlay (1)
{
    overlay
    {
        int show = 1
    }
    "rect:red"
    {
        float width = 0.3
```

(continues on next page)

(continued from previous page)

```
        float height = 0.3
        float[4] color = [ [ 1.0 0.1 0.1 0.4 ] ]
        float[2] position = [ [ 0.1 0.1 ] ]
    }
    "rect:green"
    {
        float width = 0.6
        float height = 0.2
        float[4] color = [ [ 0.1 1.0 0.1 0.4 ] ]
        float[2] position = [ [ -0.2 -0.3 ] ]
    }
    "rect:blue"
    {
        float width = 0.2
        float height = 0.4
        float[4] color = [ [ 0.1 0.1 1.0 0.4 ] ]
        float[2] position = [ [ -0.5 -0.1 ] ]
    }
    "text:red"
    {
        float[2] position = [ [ 0.1 0.1 ] ]
        float[4] color = [ [ 0 0 0 1 ] ]
        float spacing = 0.8
        float size = 0.005
        float scale = 1
        float rotation = 0
        string font = ""
        string text = "red"
        int debug = 0
    }
    "text:green"
    {
        float[2] position = [ [ -0.2 -0.3 ] ]
        float[4] color = [ [ 0 0 0 1 ] ]
        float spacing = 0.8
        float size = 0.005
        float scale = 1
        float rotation = 0
        string font = ""
        string text = "green"
        int debug = 0
    }
    "text:blue"
    {
        float[2] position = [ [ -0.5 -0.1 ] ]
        float[4] color = [ [ 0 0 0 1 ] ]
        float spacing = 0.8
        float size = 0.005
        float scale = 1
        float rotation = 0
        string font = ""
        string text = "blue"
```

(continues on next page)

(continued from previous page)

```
        int debug = 0
    }
}
```

Components in the RVOverlay which have names starting with "rect:" are used to render filled rectangles. Components starting with "text:" are used for text. The format is similar to that used by the RVPaint node, but the result is rendered for all frames of the source. The reference manual contains complete information about the RVOverlay node's properties and how the control rendering.

41.8 6.8 Limitations on Number of Open Files

RV does not impose any artificial limits on the number of source which can be in an RV session file. However, the use of some file formats, namely Quicktime .mov, .avi, and .mp4, require that the file remain open while RV is running.Each operating system (and even shell on Unix systems) has different limits on the number of open files a process is allowed to have. For example on Linux the default is 1024 files. This means that you cannot open more than 1000 or so movie files without changing the default. RV checks the limit on startup and sets it to the maximum allowed by the system.There are a number of operating system and shell dependent ways to change limits. Your facility may also have limits imposed by the IT department for accounting reasons.

41.9 6.9 What's the Best Way to Write a .rv (GTO) File?

GTO comes in three types: text (UTF8 or ASCII), binary, and compressed binary. RV can read all three types. RV normally writes text files unless an **RVImageSource** is present in the session (because an image was sent to it from another process instead of a file). In that case it will write a compressed binary GTO to save space on disk.If you think you might want to generate binary files in addition to text files you can do so using the GTO API in C++ or python. However, the text version is simple enough to write using only regular I/O APIs in any language. We recommend you write out .rv session files from RV and look at them in an editor to generate templates of the portions that are important to you. You can copy and paste parts of session files into source code as strings or even shell scripts as templates with variable substitution.

CHAPTER

FORTYTWO

CHAPTER 7 - USING QT IN MU

You can browse the Qt and other Mu modules with the documentation browser. RV wraps the Qt API. Using Qt in Mu is similar to using it in C++. Each Qt class is presented as a Mu class which you can either use directly or inherit from if need be. However, there are some major differences that need to be observed:

Note: Python code can assume that default parameters values will be supplied if not specified.

- Not all Qt classes are wrapped in Mu. It's a good idea to look in the documentation browser to see if a class is available yet.
- Property names in C++ do not always match those in Mu. Mu collects Qt properties at runtime in order to provide limited supported for unknown classes. So the set and get functions for the properties are generated at that time. Usually these names match the C++ names, but sometimes there are differences. In general, the Mu function to get a property called **foo** will be called foo(). The Mu function to set the foo property will be called setFoo(). (A good example of this is the QWidget property **visible** . In C++ the get function is isVisible() whereas the Mu function is called visible().)
- Templated classes in Qt are not available in Mu. Usually these are handled by dynamic array types or something analogous to the Qt class. In the case of template member functions (like QWidget::findChild<>) there may be an equivalent Mu version that operates slightly differently (like the Mu version QWidget.findChild).
- The QString class is not wrapped (yet). Instead, the native Mu string can be used anywhere a function takes a QString.
- You cannot control widget destruction. If you loose a reference to a QObject it will eventually be finalized (destroyed), but at an unknown time.
- Some classes cannot be inherited from. You can inherit from any QObject, QPainter, or QLayoutItem derived class except QWebFrame and QNetworkReply.
- The signal slot mechanism is slightly different in Mu than C++. It is currently not possible to make a new Qt signal, and slots do not need to be declared in a special way (but they do need to have the correct signatures to be connected). In addition, you are not required to create a QObject class to receive a signal in Mu. You can also connect a signal directly to a regular function if desired (as opposed to class member functions in C++).
- Threading is not yet available. The QThread class cannot be used in Mu yet.
- Abstract Qt classes can be instantiated. However, you can't really do anything with them.
- Protected member functions are public.

42.1 7.1 Signals and Slots

Possibly the biggest difference between the Mu and C++ Qt API is how signals and slots are handled. This discussion will assume knowledge of the C++ mechanism. See the Qt documentation if you don't know what signals and slots are.Jumping right in, here is an example hello world MuQt program. This can be run from the mu-interp binary:

```
 use qt;

\: clicked (void; bool checked)
{
    print("OK BYE\n");
    QCoreApplication.exit(0);
}

\: main ()
{
    let app    = QApplication(string[] {"hello.mu"}),
        window = QWidget(nil, Qt.Window),
        button = QPushButton("MuQt: HELLO WORLD!", window);

    connect(button, QPushButton.clicked, clicked);

    window.setSize(QSize(200, 50));
    window.show();
    window.raise();
    QApplication.exec();
}

main();
```

The main thing to notice in this example is the connect() function. A similar C++ version of this would look like this:

```
connect(button, SIGNAL(clicked(bool)), SLOT(myclickslot(bool)));
```

where myclickslot would be a slot function declared in a class. In Mu it's not necessary to create a class to receive a signal. In addition the SIGNAL and SLOT syntax is also unnecessary. However, it is necessary to exactly specify which signal is being referred to by passing its Mu function object directly. In this case QPushButton.clicked. The signal must be a function on the class of the first argument of connect().In Mu, any function which matches the signal's signature can be used to receive the signal. The downside of this is that some functions like sender() are not available in Mu. However this is easily overcome with partial application. In the above case, if we need to know who sent the signal in our clicked function, we can change its signature to accept the sender and partially apply it in the connect call like so:

```
\: clicked (void; bool checked, QPushButton sender)
{
    // do something with sender
}

\: main ()
{
    ...
```

(continues on next page)

(continued from previous page)

```
    connect(button, QPushButton.clicked, clicked(,button));
}
```

And of course additional information can be passed into the clicked function by applying more arguments.It's also possible to connect a signal to a class method in Mu if the method signature matches. Partial application can be used in that case as well. This is frequently the case when writing a mode which uses Qt interface.

42.2 7.2 Inheriting from Qt Classes

It's possible to inherit directly from the Qt classes in Mu and override methods. Virtual functions in the C++ version of Qt are translated as class methods in Mu. Non-virtual functions are regular functions in the scope of the class. In practice this means that the Mu Qt class usage is very similar to the C++ usage.The following example shows how to create a new widget type that implements a drop target. Drag and drop is one aspect of Qt that requires inheritance (in C++ and Mu):

```
use qt;

class: MyWidget : QWidget
{
    method: MyWidget (MyWidget; QObject parent, int windowFlags)
    {
        // REQUIRED: call base constructor to build Qt native object
        QWidget.QWidget(this, parent, windowFlags);
        setAcceptDrops(true);
    }

    method: dragEnterEvent (void; QDragEnterEvent event)
    {
        print("drop enter\n");
        event.acceptProposedAction();
    }

    method: dropEvent (void; QDropEvent event)
    {
        print("drop\n");
        let mimeData = event.mimeData(),
            formats = mimeData.formats();

        print("--formats--\n");
        for_each (f; formats) print("%s\n" % f);

        if (mimeData.hasUrls())
        {
            print("--urls--\n");
            for_each (u; event.mimeData().urls())
                print("%s\n" % u.toString(QUrl.None));
        }

        if (mimeData.hasText())
        {
```

(continues on next page)

(continued from previous page)

```
            print("--text--\n");
            print("%s\n" % mimeData.text());
        }

        event.acceptProposedAction();
    }
}
```

Things to note in this example: the names of the drag and drop methods matter. These are same names as used in C++. If you browser the documentation of a Qt class in Mu these will be the class methods. Only class methods can be overridden.

CHAPTER

FORTYTHREE

CHAPTER 8 - MODES AND WIDGETS

The user interface layer can augment the display and event handling in a number of different ways. For display, at the lowest level it's possible to intercept the render event in which case you override all drawing. Similarily for event handling you can bind functions in the global event table possibly overwriting existing bindings and thus replace their functions.At a higher level, both display and event handling can be done via Modes and Widgets. A Mode is a class which manages an event table independent of the global event table and a collection of functions which are bound in that table. In addition the mode can have a render function which is automatically called at the right time to augment existing rendering instead of replacing it. The UI has code which manages modes so that they may be loaded externally only when needed and automatically turned on and off.Modes are further classified as being minor or major. The only difference between them is that a major mode will always get precedence over any minor mode when processing events and there can be only a single major mode active at a time. There can be many minor modes active at once. Most extensions are created by creating a minor mode. RV currently has a single basic major mode.

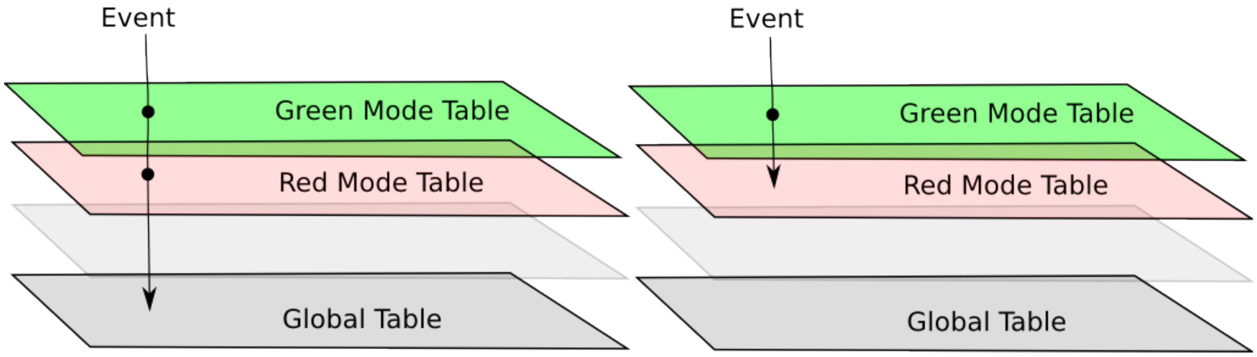


Figure 8.1:Event Propagation. Red and Green modes process the event. On the left the Red mode rejects the event allowing it to continue. On the right Red mode does not reject the event stopping the propagation.By using a mode to implement a new feature or replace or augment an existing feature in RV you can keep your extensions separate from the portion of the UI that ships with RV. In other words, you never need to touch the shipped code and your code will remain isolated.A further refinement of a mode is a widget. Widgets are minor modes which operate in a constrained region of the screen. When the pointer is in the region, the widget will receive events. When the pointer is outside the region it will not. Like a regular mode, a widget has a render function which can draw anywhere on the screen, but usually is constrainted to its input region. For example, the image info box is a widget as is the color inspector.Multiple modes and widgets may be active at the same time. At this time Widgets can only be programmed using Mu.

43.1 8.1 Outline of a Mode

In order to create a new mode you need to create a module for it and derive your mode class from the MinorMode class in the rvtypes module. The basic outline which we'll put in a file called new_mode.mu looks like this:

```
 use rvtypes;

module: new_mode {

class: NewMode : MinorMode
{
    method: NewMode (NewMode;)
    {
        init ("new-mode",
              [ global bindings ... ],
              [ local bindings ... ],
              Menu(...) );
    }
}

\: createMode (Mode;)
{
    return NewMode();
}

} // end of new_mode module
```

The function createMode() is used by the mode manager to create your mode without knowing anything about it. It should be declared in the scope of the module (not your class) and simply create your mode object and initialize it if that's necessary.When creating a mode it's necessary to call the init() function from within your constructor method. This function takes at least three arguments and as many as six. Chapter *10* goes into detail about the structure in more detail. It's declared like this in rvtypes.mu:

```
method: init (void;
              string name,
              BindingList globalBindings,
              BindingList overrideBindings,
              Menu menu = nil,
              string sortKey = nil,
              int ordering = 0)
```

The name of the mode is meant to be human readable.The "bindings" arguments supply event bindings for this mode. The bindings are only active when the mode is active and take precedence over any "global" bindings (bindings not associated with any mode). In your event function you can call the "reject" method on an event which will cause rv to pass it on to bindings "underneath" yours. This technique allows you to augment an existing binding instead of replacing it. The separation of the bindings into overrideBindings and globalBindings is due to backwards compatibility requirements, and is no longer meaningful.The menu argument allows you to pass in a menu structure which is merged into the main menu bar. This makes it possible to add new menus and menu items to the existing menus.Finally the sortKey and ordering arguments allow fine control over the order in which event bindings are applied when multiple modes are active. First the ordering value is checked (default is 0 for all modes), then the sortKey (default is the mode name).Again, see chapter *10* for more detailed information.

43.2 8.2 Outline of a Widget

A Widget looks just like a MinorMode declaration except you will derive from Widget instead of MinorMode and the base class init() function is simpler. In addition, you'll need to have a render() method (which is optional for regular modes).

```
 use rvtypes;

module: new_widget {

class: NewWidget : Widget
{
    method: NewWidget (NewWidget;)
    {
        init ("new-widget",
              [ local bindings ... ] );
    }

    method: render (void; Event event)
    {
        ...
        updateBounds(min_point, max_point);
        ...
    }
}

\: createMode (Mode;)
{
    return NewWidget();
}

} // end of new_widget module
```

In the outline above, the function updateBounds() is called in the render() method. updateBounds() informs the UI about the bounding box of your widget. This function must be called by the widget at some point. If your widget can be interactively or procedurally moved, you will probably want to may want to call it in your render() function as shown (it does not hurt to call it often). The min_point and max_point arguments are Vec2 types.

CHAPTER

FORTYFOUR

CHAPTER 9 - PACKAGE SYSTEM

Use the package system described below to expand RV.

44.1 9.1 rvpkg Command Line Tool

The rvpkg command line tool makes it possible to manage packages from the shell. If you use rvpkg you do not need to use RV's preferences UI to install/uninstall add/remove packages from the file system. We recommend using this tool instead of manually editing files to prevent the necessity of keeping abreast of how all the state is stored in new versions.The rvpkg tool can perform a superset of the functions available in RV's packages preference user interface.

-include directory	include directory as if part of RV_SUPPORT_PATH
-env	show RV_SUPPORT_PATH include app areas
-only directory	use directory as sole content of RV_SUPPORT_PATH
-add directory	add packages to specified support directory
-remove	remove packages (by name, rvpkg name, or full path to rvpkg)
-install	install packages (by name, rvpkg name, or full path to rvpkg)
-uninstall	uninstall packages (by name, rvpkg name, or full path to rvpkg)
-optin	opt-in (load) now on behalf of all users, so it will be as if they opted in
-list	list installed packages
-info	detailed info about packages (by name, rvpkg name, or full path to rvpkg)
-force	Assume answer is 'y' to any confirmations – don't be interactive

Table 9.1: rvpkg Options

Note : many of the below commands, including install, uninstall, and remove will look for the designated packages in the paths in the RV_SUPPORT_PATH environment variable. If the package you want to operate on is not in a path listed there, that path can be added on the command line with the -include option.

44.1.1 9.1.1 Getting a List of Available Packages

```
shell> rvpkg -list
```

Lists all packages that are available in the RV_SUPPORT_PATH directories. Typical output from rvpkg looks like this:

```
I L - 1.7 "Annotation" /SupportPath/Packages/annotate-1.7.rvpkg
I L - 1.1 "Documentation Browser" /SupportPath/Packages/doc_browser-1.1.rvpkg
I - O 1.1 "Export Cuts" /SupportPath/Packages/export_cuts-1.1.rvpkg
I - O 1.3 "Missing Frame Bling" /SupportPath/Packages/missing_frame_bling-1.3.rvpkg
I - O 1.4 "OS Dependent Path Conversion" /SupportPath/Packages/os_dependent_path_
↪conversion_mode-1.4.rvpkg
I - O 1.1 "Nuke Integration" /SupportPath/Packages/rvnuke-1.1.rvpkg
I - O 1.2 "Sequence From File" /SupportPath/Packages/sequence_from_file-1.2.rvpkg
I L - 1.3 "Session Manager" /SupportPath/Packages/session_manager-1.3.rvpkg
I L - 2.2 "RV Color/Image Management" /SupportPath/Packages/source_setup-2.2.rvpkg
I L - 1.3 "Window Title" /SupportPath/Packages/window_title-1.3.rvpkg
```

The first three columns indicate installation status (I), load status (L), and whether or not the package is optional (O).If you want to include a support path directory that is not in RV_SUPPORT_PATH, you can include it like this:

```
shell> rvpkg -list -include /path/to/other/support/area
```

To limit the list to a single support area:

```
shell> rvpkg -list -only /path/to/area
```

The -include and -only arguments may be applied to other options as well.

44.1.2 9.1.2 Getting Information About the Environment

You can see the entire support path list with the command:

```
shell> rvpkg -env
```

This will show alternate version package areas constructed from the RV_SUPPORT_PATH environment variable to which packages maybe added, removed, installed and uninstalled. The list may differ based on the platform.

44.1.3 9.1.3 Getting Information About a Package

```
shell> rvpkg -info /path/to/file.rvpkg
```

This will result in output like:

```
Name: Window Title
Version: 1.3
Installed: YES
Loadable: YES
Directory:
Author: Tweak Software
Organization: Tweak Software
Contact: an actual email address
```

(continues on next page)

(continued from previous page)

```
URL: http://www.tweaksoftware.com
Requires:
RV-Version: 3.9.11
Hidden: YES
System: YES
Optional: NO
Writable: YES
Dir-Writable: YES
Modes: window_title
Files: window_title.mu
```

44.1.4 9.1.4 Adding a Package to a Support Area

```
shell> rvpkg -add /path/to/area /path/to/file1.rvpkg /path/to/file2.rvpkg
```

You can add multiple packages at the same time. Remember that adding a package makes it become available for installation, it does not install it.

44.1.5 9.1.5 Removing a Package from a Support Area

```
shell> rvpkg -remove /path/to/area/Packages/file1.rvpkg
```

Unlike adding, the package in this case is the one in the support area's Packages directory. You can remove multiple packages at the same time.If the package is installed rvpkg will interactively ask for confirmation to uninstall it first. You can override that by using -force as the first argument:

```
shell> rvpkg -force -remove /path/to/area/Packages/file1.rvpkg
```

44.1.6 9.1.6 Installing and Uninstalling Available Packages

```
 shell> rvpkg -install /path/to/area/Packages/file1.rvpkg
shell> rvpkg -uninstall /path/to/area/Packages/file1.rvpkg
```

If files are missing when uninstalling rvpkg may complain. This can happen if multiple versions where somehow installed into the same area.

44.1.7 9.1.7 Combining Add and Install for Automated Installation

If you're using rvpkg from an automated installation script you will want to use the -force option to prevent the need for interaction. rvpkg will assume the answer to any questions it might ask is "yes". This will probably be the most common usage:

```
shell> rvpkg -force -install -add /path/to/area /path/to/some/file1.rvpkg
```

Multiple packages can be specified with this command. All of the packages are installed into /path/to/area.To force uninstall followed by removal:

```
shell> rvpkg -force -remove /path/to/area/Packages/file1.rvpkg
```

The -uninstall option is unnecessary in this case.

44.1.8 9.1.8 Overrideing Default Optional Package Load Behavior

If you want optional packages to be loaded by default for all users, you can do the following:

```
shell> rvpkg -optin /path/to/area/Packages/file1.rvpkg
```

In this case, rvkpg will rewrite the rvload2 file associated with the support area to indicate the package is no longer optional. The user can still unload the package if they want, but it will be loaded by default after running the command.

44.2 9.2 Package File Contents

A package file is zip file with at least one special file called PACKAGE along with .mu, .so, .dylib, and support files (plain text, images, icons, etc) which implement the actual package.Creating a package requires the zip binary. The zip binary is usually part of the default install on each of the OSes that RV runs on.The contents of the package should NOT be put in a parent directory before being zipped up. The PACKAGE manifest as well as any other files should be at the root level of the zip file.When a package is installed, RV will place all of its contents into subdirectories in one of the RV_SUPPORT_PATH locations. If the RV_SUPPORT_PATH is not defined in the environment, it is assumed to have the value of RV_HOME/plugins followed by the home directory support area (which varies with each OS: see the user manual for more info). Files contained in one zip file will all be under the same support path directory; they will not be installed distributed over more than one support path location.The install locations of files in the zip file is described in a filed called PACKAGE which must be present in the zip file. The minimum package file contains two files: PACKAGE and one other file that will be installed. A package zip file must reside in the subdirectory called Packages in one of the support path locations in order to be installed. When the user adds a package in the RV package manager, this is where the file is copied to.

44.3 9.3 PACKAGE Format

The PACKAGE file is a YAML file providing information about how the package is used and installed as well as user documentation. Every package must have a PACKAGE file with an accurate description of its contents.The top level of the file may contain the following fields:

Field	Value Type	Require	Description
package	string	•	The name of the package in human readable form
author	string		The name of the author/creator of the package
organization	string		The name of the organization (company) the author created the package for
contact	email address		The email contact of the author/support person
version	version number	•	The package version
url	URL		Web location for the package where updates, additional documentation resides
rv	version number	•	The minimum version of commercial RV which this package is compatible with
openrv	version number	•	The minimum version of Open RV which this package is compatible with
requires	zip file name list		Any other packages (as zip file names) which are required in order to install/load this package
icon	PNG file name		The name of an file with an icon for this package
imageio	file list		List of files in package which implement Image I/O
movie	file list		List of files in package which implement Movie I/O
hidden	boolean		Either “true” or “false” indicating whether package should be visible by default in the package manager
system	boolean		Either “true” or “false” indicating whether the package was pre-installed with RV and cannot be removed/uninstalled
optional	boolean		Either “true” or “false” indicating whether the package should appear loaded by default. If true the package is not loaded by default after it is installed. Typically this is used only for packages that are pre-installed. (Added in 3.10.9)
modes	YAML list		List of modes implemented in the package
files	YAML list		List non-mode file handling information
description	HTML 1.0 string	•	HTML documentation of the package for user viewing in the package manager

Table 9.2:Top level fields of PACKAGE file.Each element of the modes list describes one Mu module which is implemented as either a .mu file or a .so file. Files implementing modes are assumed to be Mu module files and will be placed in the Mu subdirectory of the support path location. The other fields are used to optionally create a menu item and/or a short cut key either of which will toggle the mode on/off. The load field indicates when the mode should be loaded: if the value is "delay" the mode will be loaded the first time it is activated, if the value is "immediate" the mode will be loaded on start up.

Field	Value Type	Required	Description
file	string	•	The name of the file which implements the mode
menu	string		If defined, the string which will appear in a menu item to indicate the status (on/off) of the mode
shortcut	string		If defined and menu is defined the shortcut for the menu item
event	string		Optional event name used to toggle mode on/off
load	string	•	Either immediate or delay indicating when the mode should be loaded
icon	PNG image file		Icon representing the mode
requires	mode file name list		Names of other mode files required to be active for this mode to be active

Table 9.3:Mode Fields

As an example, the package window_title-1.0.rvpkg has a relatively simple PACKAGE file shown here:

```
package: Window Title
author: Tweak Software
organization: Tweak Software
contact: some email address of the usual form
version: 1.0
url: http://www.tweaksoftware.com
rv: 3.6
requires: ''

modes:
  - file: window_title
    load: immediate

description: |

  <p> This package sets the window title to something that indicates the
  currently viewed media.
  </p>

  <h2>How It Works</h2>

  <p> The events play-start, play-stop, and frame-changed, are bound to
  functions which call setWindowTitle(). </p>
```

When the package zip file contains additional support files (which are not specified as modes) the package manager will try to install them in locations according to the file type. However, you can also directly specify where the additional files go relative to the support path root directory.

Field	Value Type	Required	Description
file	string	•	The name of the file in the package zip file
location	string	•	Location to install file in relative to the support path root. This can contain the variable $PACKAGE to specify special package directories. E.g. SupportFiles/$PACKAGE is the support directory for the package.

Table 9.4:File FieldsFor example if you package contains icon files for user interface, they can be forced into the support files area of the package like this:

```
files:
 - file: myicon.tif
   location: SupportFiles/$PACKAGE
```

44.4 9.4 Package Management Configuration Files

There are two files which the package manager creates and uses: rvload2 (previous releases had a file called rvload) in the Mu subdirectory and rvinstall in the Packages subdirectory. rvload2 is used on start up to load package modes and create stubs in menus or events for toggling the modes on/off if they are lazy loaded. rvinstall lists the currently known package zip files with a possible an asterisk in front of each file that is installed. The rvinstall file in used only by the package manager in the preferences to keep track of which packages are which.The rvload2 file has a one line entry for each mode that it knows about. This file is automatically generated by the package manager when the user installs a package with modes in it. The first line of the file indicates the version number of the rvload2 file itself (so we can change it in the future) followed by the one line descriptions.For example, this is the contents of rvload2 after installing the window title package:

```
3
window_title,window_title.zip,nil,nil,nil,true,true,false
```

The fields are:

1. The mode name (as it appears in a require statement in Mu)
2. The name of the package zip file the mode originally comes from
3. An optional menu item name
4. An optional menu shortcut/accelerator if the menu item exists
5. An optional event to bind mode toggling to
6. A boolean indicating whether the mode should be loaded immediately or not
7. A boolean indicating whether the mode should be activated immediately
8. A boolean indicating whether the mode is optional so it should not be loaded by default unless the user opts-in.

Each field is separated by a comma and there should be no extra whitespace on the line. The rvinstall file is much simpler: it contains a single zip file name on each line and an asterisk next to any file which is current known to be installed. For example:

```
crop.zip
layer_select.zip
metadata_info.zip
sequence_from_file.zip
*window_title.zip
```

In this case, five modes would appear in the package manager UI, but only the window title package is actually installed. The zip files should exist in the same directory that rvinstall lives in.

44.5 9.5 Developing a New Package

In order to start a new package there is a chicken and egg problem which needs to be overcome: the package system wants to have a package file to install.The best way to start is to create a source directory somewhere (like your source code repository) where you can build the zip file form its contents. Create a file called PACKAGE in that directory by copying and pasting from either this manual (listing *9.3*) or from another package you know works and edit the file to reflect what you will be doing (i.e. give it a name, etc).If you are writing a Mu module implementing a mode or widget (which is also a mode) then create the .mu file in that directory also.You can at that point use zip to create the package like so:

```
shell> zip new_package-0.0.rvpkg PACKAGE the_new_mode.mu
```

This will create the new_package-0.0.rvpkg file. At this point you're ready to install your package that doesn't do anything. Open RV's preferences and in the package manager UI add the zip file and install it (preferably in your home directory so it's visible only to you while you implement it).Once you've done this, the rvload2 and rvinstall files will have been either created or updated automatically. You can then start hacking on the installed version of your Mu file (not the one in the directory you created the zip file in). Once you have it working the way you want copy it back to your source directory and create the final zip file for distribution and delete the one that was added by RV into the Packages directory.

44.5.1 9.5.2 Using the Mode Manager While Developing

It's possible to delay making an actual package file when starting development on individual modes. You can force RV to load your mode (assuming it's in the MU_MODULE_PATH someplace) like so:

```
shell> rv -flags ModeManagerLoad=my_new_mode
```

where my_new_mode is the name of the .mu file with the mode in it (without the extension).You can get verbose information on what's being loaded and why (or why not by setting the verbose flag):

```
shell> rv -flags ModeManagerVerbose
```

The flags can be combined on the command line.

```
shell> rv -flags ModeManagerVerbose ModeManagerLoad=my_new_mode
```

If your package is installed already and you want to force it to be loaded (this overrides the user preferences) then:

```
shell> rv -flags ModeManagerPreload=my_already_installed_mode
```

similarly, if you want to force a mode not to be loaded:

```
shell> rv -flags ModeManagerReject=my_already_installed_mode
```

44.5.2 9.5.3 Using -debug mu

Normally, RV will compile Mu files to conserve space in memory. Unfortunately, that means loosing a lot of information like source locations when exceptions are thrown. You can tell RV to allow debugging information by adding -debug mu to the end of the RV command line. This will consume more memory but report source file information when displaying a stack trace.

44.5.3 9.5.4 The Mu API Documentation Browser

The Mu modules are documented dynamically by the documentation browser. This is available under RV's help menu "Mu API Documentation Browser".

44.6 9.6 Loading Versus Installing and User Override

The package manager allows each user to individually install and uninstall packages in support directories that they have permission in. For directories that the user does not have permission in the package manager maintains a separate list of packages which can be excluded by the user.For example, there may be a package installed facility wide owned by an administrator. The support directory with facility wide packages only allows read permission for normal users. Packages that were installed and loaded by the administrator will be automatically loaded by all users.In order to allow a user to override the loading of system packages, the package manager keeps a list of packages not to load. This is kept in the user's preferences file (see user manual for location details). In the package manager UI the "load" column indicates the user status for loading each package in his/her path.

44.6.1 9.6.1 Optional Packages

The load status of optional packages are also kept in the user's preferences, however these packages use a different preference variable to determine whether or not they should be loaded. By default optional packages are not loaded when installed. A package is made optional by setting the ``optional" value in the PACKAGE file to true.

CHAPTER

FORTYFIVE

CHAPTER 10 - A SIMPLE PACKAGE

This first example will show how to create a package that defines some key bindings and creates a custom personal menu. You will not need to edit a .rvrc.mu file to do this as in previous versions.We'll be creating a package intended to keep all our personal customizations. To start with we'll need to make a Mu module that implements a new mode. At first won't do anything at all: just load at start up. Put the following in to a file called mystuff.mu.

```
use rvtypes;
use extra_commands;
use commands;

module: mystuff {

class: MyStuffMode : MinorMode
{
    method: MyStuffMode (MyStuffMode;)
    {
        init("mystuff-mode",
             nil,
             nil,
             nil);
    }
}

\: createMode (Mode;)
{
    return MyStuffMode();
}

} // end module
```

Now we need to create a PACKAGE file in the same directory before we can create the package zip file. It should look like this:

```
package: My Stuff
author: M. VFX Artiste
version: 1.0
rv: 3.6
requires: ''

modes:
  - file: mystuff
```

(continues on next page)

(continued from previous page)

```
    load: immediate

description: |
  <p>M. VFX Artiste's Personal RV Customizations</p>
```

Assuming both files are in the same directory, we create the zip file using this command from the shell:

```
shell> zip mystuff-1.0.rvpkg PACKAGE mystuff.mu
```

The file mystuff-1.0.rvpkg should have been created. Now start RV, open the preferences package pane and add the mystuff-1.0.rvpkg package. You should now be able to install it. Make sure the package is both installed and loaded in your home directory's RV support directory so it's private to you.At this point, we'll edit the installed Mu file directly so we can see results faster. When we have something we like, we'll copy it back to the original mystuff.mu and make the rvpkg file again with the new code. Be careful not to uninstall the mystuff package while we're working on it or our changes will be lost. Alternately, for the more paranoid (and wiser), we could edit the file elsewhere and simply copy it onto the installed file.To start with let's add two functions on the ``<" and ``>" keys to speed up and slow down the playback by increasing and decreasing the FPS. There are two main this we need to do: add two method to the class which implement speeding up and slowing down, and bind those functions to the keys.First let's add the new methods after the class constructor MyStuffMode() along with two global bindings to the ``<" and ``>" keys. The class definition should now look like this:

```
 ...

class: MyStuffMode : MinorMode
{
    method: MyStuffMode (MyStuffMode;)
    {
        init("mystuff-mode",
             [("key-down-->", faster, "speed up fps"),
               ("key-down--<", slower, "slow down fps")],
             nil,
             nil);
    }

method: faster (void; Event event)
    {
        setFPS(fps() \* 1.5);
        displayFeedback("%g fps" % fps());
    }

method: slower (void; Event event)
    {
        setFPS(fps() \* 1.0/1.5);
        displayFeedback("%g fps" % fps());
    }
}
```

The bindings are created by passing a list of tuples to the init function. Each tuple contains three elements: the event name to bind to, the function to call when it is activated, and a single line description of what it does. In Mu a tuple is formed by putting parenthesis around comma separated elements. A list is formed by enclosing its elements in square brackets. So a list of tuples will have the form:

```
[ (...), (...), ... ]
```

Where the ``…” means ``and so on”. The first tuple in our list of bindings is:

```
(key-down-->, faster, speed up fps)
```

So the event in this case is key-down–> which means the point at which the > key is pressed. The symbol faster is referring to the method we declared above. So faster will be called whenever the key is pressed. Similarily we bind slower (from above as well) to key-down–<.

```
("key-down--<", slower, "slow down fps")
```

And to put them in a list requires enclose the two of them in square brackets:

```
[("key-down-->", faster, "speed up fps"),
("key-down--<", slower, "slow down fps")]
```

To add more bindings you create more methods to bind and add additional tuples to the list.The python version of above looks like this:

```
from rv.rvtypes import *
from rv.commands import *
from rv.extra_commands import *

class PyMyStuffMode(MinorMode):

   def __init__(self):
        MinorMode.__init__(self)
        self.init("py-mystuff-mode",
                  [ ("key-down-->", self.faster, "speed up fps"),
                    ("key-down--<", self.slower, "slow down fps") ],
                  None,
                  None)

    def faster(self, event):
        setFPS(fps() * 1.5)
        displayFeedback("%g fps" % fps(), 2.0);

    def slower(self, event):
        setFPS(fps() * 1.0/1.5)
        displayFeedback("%g fps" % fps(), 2.0);


def createMode():
    return PyMyStuffMode()
```

45.1 10.1 How Menus Work

Adding a menu is fairly straightforward if you understand how to create a MenuItem. There are different types of MenuItems: items that you can select in the menu and cause something to happen, or items that are themselves menus (sub-menu). The first type is constructed using this constructor (shown here in prototype form) for Mu:

```
MenuItem(string       label,
        (void;Event) actionHook,
        string       key,
        (int;)       stateHook);
```

or in Python this is specified as a tuple:

```
("label", actionHook, "key", stateHook)
```

The actionHook and stateHook arguments need some explanation. The other two (the label and key) are easier: the label is the text that appears in the menu item and the key is a hot key for the menu item.The actionHook is the purpose of the menu item–it is a function or method which will be called when the menu item is activated. This is just like the method we used with bind() — it takes an Event object. If actionHook is nil, than the menu item won't do anything when the user selects it.The stateHook provides a way to check whether the menu item should be enabled (or greyed out)–it is a function or method that returns an int. In fact, it is really returning one of the following symbolic constants: NeutralMenuState, UncheckMenuState, CheckedMenuState, MixedStateMenuState, or DisabledMenuState. If the value of stateHook is nil, the menu item is assumed to always be enabled, but not checked or in any other state.A sub-menu MenuItem can be create using this constructor in Mu:

```
MenuItem(string     label,
        MenuItem[] subMenu);
```

or a tuple of two elements in Python:

```
("label", subMenu)
```

The subMenu is an array of MenuItems in Mu or a list of menu item tuples in Python. Usually we'll be defining a whole menu — which is an array of MenuItems. So we can use the array initialization syntax to do something like this:

```
let myMenu = MenuItem {"My Menu", Menu {
   {"Menu Item", menuItemFunc, nil, menuItemState},
   {"Other Menu Item", menuItemFunc2, nil, menuItemState2}
}}
```

Finally you can create a sub-menu by nesting more MenuItem constructors in the subMenu.

```
MenuItem myMenu = {"My Menu", Menu {
       {"Menu Item", menuItemFunc, nil, menuItemState},
       {"Other Menu Item", menuItemFunc2, nil, menuItemState2},
       {"Sub-Menu", Menu {
            {"First Sub-Menu Item", submenuItemFunc1, nil, submenu1State}
       }}
   }};
```

in Python this looks like:

```
("My Menu", [
 ("Menu Item", menuItemFunc, None, menuItemState),
 ("Other Menu Item", menuItemFunc2, None, menuItemState2)])
```

You'll see this on a bigger scale in the rvui module where most the menu bar is declared in one large constructor call.

45.2 10.2 A Menu in MyStuffMode

Now back to our mode. Let's say we want to put our faster and slower functions on menu items in the menu bar. The fourth argument to the init() function in our constructor takes a menu representing the menu bar. You only define menus which you want to either modify or create. The contents of our main menu will be merged into the menu bar.By merge into we mean that the menus with the same name will share their contents. So for example if we add the File menu in our mode, RV will not create a second File menu on the menu bar; it will add the contents of our File menu to the existing one. On the other hand if we call our menu MyStuff RV will create a brand new menu for us (since presumably MyStuff doesn't already exist). This algorithm is applied recursively so sub-menus with the same name will also be merged, and so on.So let's add a new menu called MyStuff with two items in it to control the FPS. In this example, we're only showing the actual init() call from mystuff.mu:

```
init("mystuff-mode",
    [ ("key-down-->", faster, "speed up fps"),
      ("key-down--<", slower, "slow down fps") ],
    nil,
    Menu {
        {"MyStuff", Menu {
                {"Increase FPS", faster, nil},
                {"Decrease FPS", slower, nil}
            }
        }
    });
```

Normally RV will place the new menu (called ``MyStuff") just before the Windows menu.If we wanted to use menu accelerators instead of (or in addition to) the regular event bindings we add those in the menu item constructor. For example, if we wanted to also use the keys - and = for slower and faster we could do this:

```
init("mystuff-mode",
    [ ("key-down-->", faster, "speed up fps"),
      ("key-down--<", slower, "slow down fps") ],
    nil,
    Menu {
        {"MyStuff", Menu {
                {"Increase FPS", faster, "="},
                {"Decrease FPS", slower, "-"}
            }
        }
    });
```

The advantage of using the event bindings instead of the accelerator keys is that they can be overridden and mapped and unmapped by other modes and ``chained" together. Of course we could also use > and < for the menu accelerator keys as well (or instead of using the event bindings).The Python version of the script might look like this:

```
from rv.rvtypes import *
from rv.commands import *
```

(continues on next page)

(continued from previous page)

```
from rv.extra_commands import *

class PyMyStuffMode(MinorMode):

   def __init__(self):
        MinorMode.__init__(self)
        self.init("py-mystuff-mode",
                  [ ("key-down-->", self.faster, "speed up fps"),
                    ("key-down--<", self.slower, "slow down fps") ],
                  None,
                  [ ("MyStuff",
                     [ ("Increase FPS", self.faster, "=", None),
                       ("Decrease FPS", self.slower, "-", None)] )] )

    def faster(self, event):
        setFPS(fps() * 1.5)
        displayFeedback("%g fps" % fps(), 2.0);

    def slower(self, event):
        setFPS(fps() * 1.0/1.5)
        displayFeedback("%g fps" % fps(), 2.0);


def createMode():
    return PyMyStuffMode()
```

45.3 10.3 Finishing up

Finally, we'll create the final rvpkg package by copying mystuff.mu back to our temporary directory with the PACKAGES file where we originally made the rvpkg file.Next start RV and uninstall and remove the mystuff package so it no longer appears in the package manager UI. Once you've done this recreate the rvpkg file from scratch with the new mystuff.mu file and the PACKAGES file:

```
shell> zip mystuff-1.0.rvpkg PACKAGES mystuff.mu
```

or if you're using python:

```
shell> zip mystuff-1.0.rvpkg PACKAGES mystuff.py
```

You can now add the latest mysuff-1.0.rvpkg file back to RV and use it. In the future add personal customizations directly to this package and you'll always have a single file you can install to customize RV.

CHAPTER

FORTYSIX

CHAPTER 11 - THE CUSTOM MATTE PACKAGE

Now that we've tried the simple stuff, let's do something useful. RV has a number of settings for viewing mattes. These are basically regions of the frame that are darkened or completely blackened to simulate what an audience will see when the movie is projected. The size and shape of the matte is an artistic decision and sometimes a unique matte will be required.

You can find various common mattes already built into RV under the View menu. In this example we'll create a Python package that reads a file when RV starts to get a list of matte geometry and names. We'll make a custom menu out of these which will set some state in the UI.To start with, we'll assume that the path to the file containing the mattes is located in an environment variable called RV_CUSTOM_MATTE_DEFINITIONS. We'll get the value of that variable, open and parse the file, and create a data struct holding all of the information about the mattes. If it is not defined we will provide a way for the user to locate the file through an open-file-dialog and then parse the file.

46.1 11.1 Creating the Package

Use the same method described in Chapter *10* to begin working on the package. If you haven't read that chapter please do so first. A completed version of the package created in this chapter is included in the RV distribution. So using that as reference is a good idea.

46.2 11.2 The Custom Matte File

The file will be a very simple comma separated value (CSV) file. Each line starts with the name of the custom matte (shown in the menu) followed by four floating point values and then a text field description which will be displayed when that matte is activated. So each line will look something like this:

```
matte menu name, aspect ratio, fraction of image visible, center point of matte in X,␣
↪center point of matte in Y, descriptive text
```

46.3 11.3 Parsing the Matte File

Before we actually parse the file, we should decide what we want when we're done. In this case we're going to make our own data structure to hold the information in each line of the file. We'll hold all of the information we collect in a Python dictionary with the following keys:

```
"name", "ratio", "heightVisible", "centerX", "centerY", and "text"
```

Next we'll write a method for our mode that does the parsing and updates our internal mattes dictionary.

Note: If you are unfamiliar with object oriented programing you can substitute the word function for method. This manual will sometimes refer to a method as a function. It will never refer to a non-method function as a method.

```
    def updateMattesFromFile(self, filename):
        # Make sure the definition file exists
        if (not os.path.exists(filename)):
            raise KnownError("ERROR: Custom Mattes Mode: Non-existent mattes" +
                " definition file: '%s'" % filename)

        # Walk through the lines of the definition file collecting matte
        # parameters
        order = []
        mattes = {}
        for line in open(filename).readlines():
            tokens = line.strip("\n").split(",")
            if (len(tokens) == 6):
                order.append(tokens[0])
                mattes[tokens[0]] = {
                    "name" : tokens[0], "ratio" : tokens[1],
                    "heightVisible" : tokens[2], "centerX" : tokens[3],
                    "centerY" : tokens[4], "text" : tokens[5]}

        # Make sure we got some valid mattes
        if (len(order) == 0):
            self._order = []
            self._mattes = {}
            raise KnownError("ERROR: Custom Mattes Mode: Empty mattes" +
                " definition file: '%s'" % filename)

        self._order = order
        self._mattes = mattes
```

There are a number of things to note in this function. First of all, to keep track of the order in which we read the definitions from the mattes file you will see that stored in the "_order" Python list. The "_mattes" dictionary's keys are the same as the "_order" list, but since dictionaries are not ordered we use the list to remember the order.We check to see if the file actually exists and if not simply raise a KnownError Exception. So the caller of this function will have to be ready to except a KnownError if the matte definition file cannot be found or if it is empty. The KnowError Exception is simply our own Exception class. Having our own Exception class allows us to raise and except Exceptions that we know about while letting others we don't expect to still reach the user. Here is the definition of our KnownError Exception class.

```
class KnownError(Exception): pass
```

We use the built-in Python readlines() method to go through the mattes file contents one line at a time. Each time through the loop, the next line is split over commas since that's how defined the fields of each line.If there are not exactly 6 tokens after splitting the line, that means the line is corrupt and we ignore it. Otherwise, we add a new dictionary to our "_mattes" dictionary of matte definition dictionaries.If we cannot find the path defined in the environment variable then we leave it blank:

```
         try:
            definition = os.environ["RV_CUSTOM_MATTE_DEFINITIONS"]
        except KeyError:
            definition = ""
```

At this point the custom_mattes.py file looks like this:

```
from rv import commands, rvtypes
import os

class KnownError(Exception): pass

class CustomMatteMinorMode(rvtypes.MinorMode):

    def __init__(self):
        rvtypes.MinorMode.__init__(self)
        self._order = []
        self._mattes = {}
        self._currentMatte = ""
        self.init("custom-mattes-mode", None, None, None)

        try:
            definition = os.environ["RV_CUSTOM_MATTE_DEFINITIONS"]
        except KeyError:
            definition = ""
        try:
            self.updateMattesFromFile(definition)
        except KnownError,inst:
            print(str(inst))

    def updateMattesFromFile(self, filename):

        # Make sure the definition file exists
        if (not os.path.exists(filename)):
            raise KnownError("ERROR: Custom Mattes Mode: Non-existent mattes" +
                " definition file: '%s'" % filename)

        # Walk through the lines of the definition file collecting matte
        # parameters
        order = []
        mattes = {}
        for line in open(filename).readlines():
            tokens = line.strip("\n").split(",")
            if (len(tokens) == 6):
                order.append(tokens[0])
                mattes[tokens[0]] = {
                    "name" : tokens[0], "ratio" : tokens[1],
                    "heightVisible" : tokens[2], "centerX" : tokens[3],
                    "centerY" : tokens[4], "text" : tokens[5]}

        # Make sure we got some valid mattes
        if (len(order) == 0):
            self._order = []
            self._mattes = {}
            raise KnownError("ERROR: Custom Mattes Mode: Empty mattes" +
                " definition file: '%s'" % filename)

        self._order = order
```

(continues on next page)

(continued from previous page)

```
        self._mattes = mattes

def createMode():
    return CustomMatteMinorMode()
```

46.4 11.4 Adding Bindings and Menus

The mode constructor needs to do three things: call the file parsing function, do something sensible if the matte file parsing fails, and build a menu with the items found in the matte file as well as add bindings to the menu items.We have already gone over the parsing. Once parsing is done we either have a good list of mattes or an empty one, but either way we move on to setting up the menus. Here is the method that will build the menus and bindings.

```
     def setMenuAndBindings(self):

        # Walk through all of the mattes adding a menu entry as well as a
        # hotkey binding for alt + index number
        # NOTE: The bindings will only matter for the first 9 mattes since you
        # can't really press "alt-10".
        matteItems = []
        bindings = []
        if (len(self._order) > 0):
            matteItems.append(("No Matte", self.selectMatte(""), "alt `",
                self.currentMatteState("")))
            bindings.append(("key-down--alt--`", ""))

            for i,m in enumerate(self._order):
                matteItems.append((m, self.selectMatte(m),
                    "alt %d" % (i+1), self.currentMatteState(m)))
                bindings.append(("key-down--alt--%d" % (i+1), m))
        else:
            def nada():
                return commands.DisabledMenuState
            matteItems = [("RV_CUSTOM_MATTE_DEFINITIONS UNDEFINED",
                None, None, nada)]

        # Always add the option to choose a new definition file
        matteItems += [("_", None)]
        matteItems += [("Choose Definition File...", self.selectMattesFile,
            None, None)]

        # Clear the menu then add the new entries
        matteMenu = [("View", [("_", None), ("Custom Mattes", None)])]
        commands.defineModeMenu("custom-mattes-mode", matteMenu)
        matteMenu = [("View", [("_", None), ("Custom Mattes", matteItems)])]
        commands.defineModeMenu("custom-mattes-mode", matteMenu)

        # Create hotkeys for each matte
        for b in bindings:
            (event, matte) = b
            commands.bind("custom-mattes-mode", "global", event,
```

(continues on next page)

(continued from previous page)

```
                self.selectMatte(matte), "")
```

You can see that creating the menus and bindings walks through the contents of our "_mattes" dictionary in the order dictated by "_order". If there are no valid mattes found then we add the alert in the menu to the user that the environment variable was not defined. You can also see from the example above that each menu entry is set to trigger a call to selectMatte for the associated matte definition. This is a neat technique where we use a factory method to create our event handling method for each valid matte we found. Here is the content of that:

```
    def selectMatte(self, matte):

       # Create a method that is specific to each matte for setting the
       # relevant session node properties to display the matte
       def select(event):
           self._currentMatte = matte
           if (matte == ""):
               commands.setIntProperty("#Session.matte.show", [0], True)
               extra_commands.displayFeedback("Disabling mattes", 2.0)
           else:
               m = self._mattes[matte]
               commands.setFloatProperty("#Session.matte.aspect",
                   [float(m["ratio"])], True)
               commands.setFloatProperty("#Session.matte.heightVisible",
                   [float(m["heightVisible"])], True)
               commands.setFloatProperty("#Session.matte.centerPoint",
                   [float(m["centerX"]), float(m["centerY"])], True)
               commands.setIntProperty("#Session.matte.show", [1], True)
               extra_commands.displayFeedback(
                   "Using '%s' matte" % matte, 2.0)
       return select
```

Notice that we didn't say which matte to set it to. The function just sets the value to whatever its argument is. Since this function is going to be called when the menu item is selected it needs to be an event function (a function which takes an Event as an argument and returns nothing). In the case where we want no matte drawn, we'll pass in the empty string ("").The menu state function (which will put a check mark next to the current matte) has a similar problem. In this case we'll use a mechanism with similar results. We'll create a method which returns a function given a matte. The returned function will be our menu state function. This sounds complicated, but it's simple in use:The thing to note here is that the parameter m passed into currentMatteState() is being used inside the function that it returns. The m inside the matteState() function is known as a free variable. The value of this variable at the time that currentMatteState() is called becomes wrapped up with the returned function. One way to think about this is that each time you call currentMatteState() with a new value for m, it will return a different copy of matteState() function where the internal m is replaced the value of currentMatteState()'s m.

```
    def currentMatteState(self, m):
       def matteState():
           if (m != "" and self._currentMatte == m):
               return commands.CheckedMenuState
           return commands.UncheckedMenuState
       return matteState
```

Selecting mattes is not the only menu option we added in setMenuAndBindings(). We also added an option to select the matte definition file (or change the selected one) if none was found before. Here is the contents of the selectMatteFile() method:

```
    def selectMattesFile(self, event):
        definition = commands.openFileDialog(True, False, False, None, None)[0]
        try:
            self.updateMattesFromFile(definition)
        except KnownError,inst:
            print(str(inst))
        self.setMenuAndBindings()
```

Notice here that we basically repeat what we did before when parsing the mattes definition file from the environment. We update our internal mattes structures and the setup the menus and bindings.It is also important to clear out any existing bindings when we load a new mattes file. Therefore we should modify our parsing function do this for us like so:

```
    def updateMattesFromFile(self, filename):

        # Make sure the definition file exists
        if (not os.path.exists(filename)):
            raise KnownError("ERROR: Custom Mattes Mode: Non-existent mattes" +
                " definition file: '%s'" % filename)

        # Clear existing key bindings
        for i in range(len(self._order)):
            commands.unbind(
                "custom-mattes-mode", "global", "key-down--alt--%d" % (i+1))

      ... THE REST IS AS BEFORE ...
```

So the full mode constructor function now looks like this:

```
class CustomMatteMinorMode(rvtypes.MinorMode):

    def __init__(self):
        rvtypes.MinorMode.__init__(self)
        self._order = []
        self._mattes = {}
        self._currentMatte = ""
        self.init("custom-mattes-mode", None, None, None)

        try:
            definition = os.environ["RV_CUSTOM_MATTE_DEFINITIONS"]
        except KeyError:
            definition = ""
        try:
            self.updateMattesFromFile(definition)
        except KnownError,inst:
            print(str(inst))
```

46.5 11.5 Handling Settings

Wouldn't it be nice to have our package remember what our last matte setting was and where the last definition file was? Lets see how to add settings. First thing is first. We need to write our settings in order to read them back later. Lets start by writing out the location of our mattes definition file when we parse a new one. Here is an updated version of updateMattesFromFile():

```
    def updateMattesFromFile(self, filename):

        # Make sure the definition file exists
        if (not os.path.exists(filename)):
            raise KnownError("ERROR: Custom Mattes Mode: Non-existent mattes" +
                " definition file: '%s'" % filename)

        # Clear existing key bindings
        for i in range(len(self._order)):
            commands.unbind(
                "custom-mattes-mode", "global", "key-down--alt--%d" % (i+1))

        # Walk through the lines of the definition file collecting matte
        # parameters
        order = []
        mattes = {}
        for line in open(filename).readlines():
            tokens = line.strip("\n").split(",")
            if (len(tokens) == 6):
                order.append(tokens[0])
                mattes[tokens[0]] = {
                    "name" : tokens[0], "ratio" : tokens[1],
                    "heightVisible" : tokens[2], "centerX" : tokens[3],
                    "centerY" : tokens[4], "text" : tokens[5]}

        # Make sure we got some valid mattes
        if (len(order) == 0):
            self._order = []
            self._mattes = {}
            raise KnownError("ERROR: Custom Mattes Mode: Empty mattes" +
                " definition file: '%s'" % filename)

        # Save the definition path and assign the mattes
        commands.writeSettings(
            "CUSTOM_MATTES", "customMattesDefinition", filename)
        self._order = order
        self._mattes = mattes
```

See how at the bottom of the function we are now writting the definition file to the CUSTOM_MATTES settings. Now lets also update the selectMatte() method to remember what matte we selected.

```
    def selectMatte(self, matte):

        # Create a method that is specific to each matte for setting the
        # relevant session node properties to display the matte
        def select(event):
```

(continues on next page)

(continued from previous page)

```
            self._currentMatte = matte
            if (matte == ""):
                commands.setIntProperty("#Session.matte.show", [0], True)
                extra_commands.displayFeedback("Disabling mattes", 2.0)
            else:
                m = self._mattes[matte]
                commands.setFloatProperty("#Session.matte.aspect",
                    [float(m["ratio"])], True)
                commands.setFloatProperty("#Session.matte.heightVisible",
                    [float(m["heightVisible"])], True)
                commands.setFloatProperty("#Session.matte.centerPoint",
                    [float(m["centerX"]), float(m["centerY"])], True)
                commands.setIntProperty("#Session.matte.show", [1], True)
                extra_commands.displayFeedback(
                    "Using '%s' matte" % matte, 2.0)
            commands.writeSettings("CUSTOM_MATTES", "customMatteName", matte)
        return select
```

Here notice the second to last line. We save the matte that was just selected. Lastly lets see what we have to do to make use of these when we initialize our mode. Here is the final version of the constructor:

```
class CustomMatteMinorMode(rvtypes.MinorMode):

    def __init__(self):
        rvtypes.MinorMode.__init__(self)
        self._order = []
        self._mattes = {}
        self._currentMatte = ""
        self.init("custom-mattes-mode", None, None, None)

        try:
            definition = os.environ["RV_CUSTOM_MATTE_DEFINITIONS"]
        except KeyError:
            definition = str(commands.readSettings(
                "CUSTOM_MATTES", "customMattesDefinition", ""))
        try:
            self.updateMattesFromFile(definition)
        except KnownError,inst:
            print(str(inst))
        self.setMenuAndBindings()

        lastMatte = str(commands.readSettings(
            "CUSTOM_MATTES", "customMatteName", ""))
        for matte in self._order:
            if matte == lastMatte:
                self.selectMatte(matte)(None)
```

Here we grab the last known location of the mattes definition file if we did not find one in the environment. We also attempt to look up the last matte that was used and if we can find it among the mattes we parsed then we enable that selection.

46.6 11.6 The Finished custom_mattes.py File

```
from rv import commands, rvtypes, extra_commands
import os

class KnownError(Exception): pass

class CustomMatteMinorMode(rvtypes.MinorMode):

    def __init__(self):
        rvtypes.MinorMode.__init__(self)
        self._order = []
        self._mattes = {}
        self._currentMatte = ""
        self.init("custom-mattes-mode", None, None, None)

        try:
            definition = os.environ["RV_CUSTOM_MATTE_DEFINITIONS"]
        except KeyError:
            definition = str(commands.readSettings(
                "CUSTOM_MATTES", "customMattesDefinition", ""))
        try:
            self.updateMattesFromFile(definition)
        except KnownError,inst:
            print(str(inst))
        self.setMenuAndBindings()

        lastMatte = str(commands.readSettings(
            "CUSTOM_MATTES", "customMatteName", ""))
        for matte in self._order:
            if matte == lastMatte:
                self.selectMatte(matte)(None)

    def currentMatteState(self, m):
        def matteState():
            if (m != "" and self._currentMatte == m):
                return commands.CheckedMenuState
            return commands.UncheckedMenuState
        return matteState

    def selectMatte(self, matte):

        # Create a method that is specific to each matte for setting the
        # relevant session node properties to display the matte
        def select(event):
            self._currentMatte = matte
            if (matte == ""):
                commands.setIntProperty("#Session.matte.show", [0], True)
                extra_commands.displayFeedback("Disabling mattes", 2.0)
            else:
                m = self._mattes[matte]
                commands.setFloatProperty("#Session.matte.aspect",
```

(continues on next page)

(continued from previous page)

```
                [float(m["ratio"])], True)
            commands.setFloatProperty("#Session.matte.heightVisible",
                [float(m["heightVisible"])], True)
            commands.setFloatProperty("#Session.matte.centerPoint",
                [float(m["centerX"]), float(m["centerY"])], True)
            commands.setIntProperty("#Session.matte.show", [1], True)
            extra_commands.displayFeedback(
                "Using '%s' matte" % matte, 2.0)
        commands.writeSettings("CUSTOM_MATTES", "customMatteName", matte)
    return select

def selectMattesFile(self, event):
    definition = commands.openFileDialog(True, False, False, None, None)[0]
    try:
        self.updateMattesFromFile(definition)
    except KnownError,inst:
        print(str(inst))
    self.setMenuAndBindings()

def setMenuAndBindings(self):

    # Walk through all of the mattes adding a menu entry as well as a
    # hotkey binding for alt + index number
    # NOTE: The bindings will only matter for the first 9 mattes since you
    # can't really press "alt-10".
    matteItems = []
    bindings = []
    if (len(self._order) > 0):
        matteItems.append(("No Matte", self.selectMatte(""), "alt `",
            self.currentMatteState("")))
        bindings.append(("key-down--alt--`", ""))

        for i,m in enumerate(self._order):
            matteItems.append((m, self.selectMatte(m),
                "alt %d" % (i+1), self.currentMatteState(m)))
            bindings.append(("key-down--alt--%d" % (i+1), m))
    else:
        def nada():
            return commands.DisabledMenuState
        matteItems = [("RV_CUSTOM_MATTE_DEFINITIONS UNDEFINED",
            None, None, nada)]

    # Always add the option to choose a new definition file
    matteItems += [("_", None)]
    matteItems += [("Choose Definition File...", self.selectMattesFile,
        None, None)]

    # Clear the menu then add the new entries
    matteMenu = [("View", [("_", None), ("Custom Mattes", None)])]
    commands.defineModeMenu("custom-mattes-mode", matteMenu)
    matteMenu = [("View", [("_", None), ("Custom Mattes", matteItems)])]
    commands.defineModeMenu("custom-mattes-mode", matteMenu)
```

(continues on next page)

(continued from previous page)

```
        # Create hotkeys for each matte
        for b in bindings:
            (event, matte) = b
            commands.bind("custom-mattes-mode", "global", event,
                self.selectMatte(matte), "")

    def updateMattesFromFile(self, filename):

        # Make sure the definition file exists
        if (not os.path.exists(filename)):
            raise KnownError("ERROR: Custom Mattes Mode: Non-existent mattes" +
                " definition file: '%s'" % filename)

        # Clear existing key bindings
        for i in range(len(self._order)):
            commands.unbind(
                "custom-mattes-mode", "global", "key-down--alt--%d" % (i+1))

        # Walk through the lines of the definition file collecting matte
        # parameters
        order = []
        mattes = {}
        for line in open(filename).readlines():
            tokens = line.strip("\n").split(",")
            if (len(tokens) == 6):
                order.append(tokens[0])
                mattes[tokens[0]] = {
                    "name" : tokens[0], "ratio" : tokens[1],
                    "heightVisible" : tokens[2], "centerX" : tokens[3],
                    "centerY" : tokens[4], "text" : tokens[5]}

        # Make sure we got some valid mattes
        if (len(order) == 0):
            self._order = []
            self._mattes = {}
            raise KnownError("ERROR: Custom Mattes Mode: Empty mattes" +
                " definition file: '%s'" % filename)

        # Save the definition path and assign the mattes
        commands.writeSettings(
            "CUSTOM_MATTES", "customMattesDefinition", filename)
        self._order = order
        self._mattes = mattes

def createMode():
    return CustomMatteMinorMode()
```

CHAPTER

FORTYSEVEN

CHAPTER 12 - AUTOMATED COLOR AND VIEWING MANAGEMENT

Color management in RV can be broken into three separate issues:

- Determination of the input color space
- Deciding whether the input color space should be converted to the linear working space and with what transform
- Displaying the working color space on a particular device possibly in a manner which simulates another device (e.g, film look on an LCD monitor).

Each of the above corresponds to a set of features in RV which can be automated:

- Examining particular image attributes it's often possible to determine color space. Some images may use a naming convention or may be located in a particular place on a file system which indicates its color space. It's even possible that a separate file or program needs to be executed to get the actual color space.
- Input spaces can be transformed to working spaces using the built in transforms like sRGB, gamma, or Cineon log space to linear space. If RV does not have a built-in transform for the color space, a file LUT (one per input source) may be used to interpolate independent channel functions using a channel LUT or a general function of R G and B channels using a 3D LUT. Values from a CDL can also be used to bring the input into the working color space.
- Ideally RV will have a fixed set of display transform which map a linear space to a display. This makes it possible to load multiple sets of images with differing color spaces, transform them to a common linear working space, and display them using a global display transform. RV has built-in transform for sRGB and gamma and can also use a channel or 3D LUT if a custom function is needed.

In addition to the color issues there are a few others which might need to be detected and/or corrected:

- Unrecorded or incorrect pixel aspect ratios (e.g., DPX files with inaccurate headers)
- Special mattes which should be used with particular images
- Incorrect frame numbers
- Incorrect fps
- Specific production information which is not located in the image (e.g., shot information, tracking information)

RV lets you customize all of the above for your facility and workflow by hooking into the user interface code. The most important method of doing so is using special events generated by RV internally and setting internal state at that time.

47.1 12.1 The source-group-complete Event

The source-group-complete event is generated whenever media is added to a session; this includes when a Source is created, or when the set of media held be a Source is modified. By binding a function to this event, it's possible to configure any color space or other image dependant aspects of RV at the time the file is added. This can save a considerable amount of time and headache when a large number of people are using RV in differing circumstances.See the sections below for information about creating a package which binds source-group-complete to do color management.

47.2 12.2 The default source-group-complete behavior

RV binds its own color management function located in the source_setup.py file called sourceSetup(). It's a good idea to override or augment this package for use in production environments. For example, you may want to have certain default color behavior for technical directors using movie files which differs from how a coordinator might view them (the coordinator may be looking at movies in sRGB space instead of with a film simulation for example).RV's default color management tries to use good defaults for incoming file formats. Here's the complete behavior shown as a set of heuristics applied in order:

1. If the incoming image is a TIFF file and it has no color space attribute assume it's linear
2. If the image is JPEG or a quicktime movie file (.mov) and there is no color space attribute assume it's in sRGB space
3. If there is an embedded ICC profile and that profile is for sRGB space use RV's internal sRGB space transform instead (because RV does not yet handle embedded ICC profiles)
4. If the image is TIFF and it was created by ifftoany, assume the pixel aspect ratio is incorrect and fix it
5. If the image is JPEG, has no pixel aspect ratio attribute and no density attribute and looks like it comes from Maya, fix the pixel aspect ratio
6. Use the proper built-in conversion for the color space indicated in the color space attribute of the image
7. Use the sRGB display transform if any color space was successfully determined for the input image(s)

From the user's point of view, the following situations will occur:

- A DPX or Cineon is loaded which is determined to be in Log space — turn on the built in log to linear converter
- A JPEG or Quicktime movie file is determined to be in sRGB space or if no space is specified assumed to be in sRGB space — apply the built-in sRGB to linear converter
- An EXR is loaded — assume it's linear
- A TIFF file with no color space indication is assumed to be linear, if it does have a color space use that.
- A PNG file with no color space is assumed linear, otherwise use the color space attribute in the file
- Any file with a pixel aspect ratio attribute will be assumed to be correct (unless it's determined to have come from Maya)
- The monitor's ``gamma" will be accounted for automatically (because RV assumes the monitor is an sRGB device)

In addition, the default color management implements two varieties of user-level control, as examples of what you can do from the scripting level.First, environment variables with a standard format can be used to control the linearization process for a given file type. An environment variable of the form "RV_OVERRIDE_TRANSFER_" will set the linearization transform for the specified file type (and this will override the default rules described above). For example, if the environment variable "RV_OVERRIDE_TRANSFER_TIF" is set to "sRGB" then all files with extension "tif" or "TIF" will be linearized with the sRGB transform. If you want, you can also specify the bit depth. So you could set

RV_OVERRIDE_TRANSFER_TIF_8 to sRGB and RV_OVERRIDE_TRANSFER_TIF_32 to Linear. The transform function name must be one of the following standard transforms. (The number following “Gamma” is arbitrary.)

Linear
sRGB
Cineon Log
Viper Log
Rec709
Gamma *f*

Table 12.1:Standard Linearization Transforms.

Second, any of the above-described environment variable names and standard transform names can appear on the command line following the “-flags” option.

47.3 12.3 Breakdown of sourceSetup() in the source_setup Package

The source_setup system package defines the default sourceSetup() function. This is where RV’s default color management comes from. The function starts by parsing the event contents (which contains the name of the file, the type of source node, and the source node name) as well as setting up the regular expressions used later in the function:

Note: RV Python to implement the source_setup package. The actual sourceSetup() function in source_setup.py may differ from what is described here.

```
        args         = event.contents().split(";;")
        group        = args[0]
        fileSource   = groupMemberOfType(group, "RVFileSource")
        imageSource  = groupMemberOfType(group, "RVImageSource")
        source       = fileSource if imageSource == None else imageSource
        linPipeNode  = groupMemberOfType(group, "RVLinearizePipelineGroup")
        linNode      = groupMemberOfType(linPipeNode, "RVLinearize")
        lensNode     = groupMemberOfType(linPipeNode, "RVLensWarp")
        fmtNode      = groupMemberOfType(group, "RVFormat")
        tformNode    = groupMemberOfType(group, "RVTransform2D")
        lookPipeNode = groupMemberOfType(group, "RVLookPipelineGroup")
        lookNode     = groupMemberOfType(lookPipeNode, "RVLookLUT")
        typeName     = commands.nodeType(source)
        fileNames    = commands.getStringProperty("%s.media.movie" % source, 0, 1000)
        fileName     = fileNames[0]
        ext          = fileName.split('.')[-1].upper()
        igPrim       = self.checkIgnorePrimaries(ext)
        mInfo        = commands.sourceMediaInfo(source, None)
```

The event.contents() function returns a string which might look something like this:

```
sourceGroup000000;;new
```

The split() function is used to create a dynamic array of strings to extract the source group’s name. The nodes associated with the source group are then located and the media names are taken from the source node. The source node is either an RVImageSource which stores its image data directly in the session or an RVFileSource which references media on the filesystem. Both of these node types have a media component which contains the actual media names (usually a single file in the case of an RVFileSource node).

There are three pipeline group nodes in each source group node and one pipeline group in the display group. For the default source_setup the linearize pipeline group is need to get the default RVLinearize node it contains.The next section of the function iterates over the image attributes and caches the ones we're interested in. The most important of these is the Colorspace attribute which is set by the file readers when the image color space is known.

```
        srcAttrs = commands.sourceAttributes(source, fileName)
        attrDict = dict(zip([i[0] for i in srcAttrs],[j[1] for j in srcAttrs]))
        attrMap = {
            "ColorSpace/ICC/Description" : "ICCProfileDesc",
            "ColorSpace" : "ColorSpace",
            "ColorSpace/Transfer" : "TransferFunction",
            "ColorSpace/Primaries" : "ColorSpacePrimaries",
            "DPX-0/Transfer" : "DPX0Transfer",
            "ColorSpace/Conversion" : "ConversionMatrix",
            "JPEG/PixelAspect" : "JPEGPixelAspect",
            "PixelAspectRatio" : "PixelAspectRatio",
            "JPEG/Density" : "JPEGDensity",
            "TIFF/ImageDescription" : "TIFFImageDescription",
            "DPX/Creator" : "DPXCreator",
            "EXIF/Orientation" : "EXIFOrientation",
            "EXIF/Gamma" : "EXIFGamma"}
        for key in attrMap.keys():
            try:
                exec('%s = "%s"' % (attrMap[key],attrDict[key]))
            except KeyError:
                pass
```

The function sourceAttributes() returns the image attributes for a given file in a source. In this case we're passing in the source and file which caused the event. The return value of the function is a dynamic array of tuples of type (string,string) where the first element is the name of the attribute and the second is a string representation of the value. Each iteration through the loop, the next tuple is used to assign the attribute value to the a variable with name of the attribute.The variables ICCProfileName, Colorspace, JPEGPixelAspect, etc, are all variable of type string which are defined earlier in the function.Before getting to the meat of the function, there are two helper functions declared: setPixelAspect() and setFileColorSpace().The next major section of the function matches the file name against the regular expressions that were declared at the beginning and against the values of some of the attributes that were cached.

```
        #
        #  Rules based on the extension
        #

        if (ext == 'DPX'):
            if (DPXCreator == "AppleComputers.libcineon" or
                DPXCreator == "AUTODESK"):
                #
                #  Final Cut's "Color" and Maya write out bogus DPX
                #  header info with the aspect ratio fields set
                #  improperly (to 0s usually). Properly undefined DPX
                #  headers do not have the value 0.
                #
```

(continues on next page)

(continued from previous page)

```
            if (int(PixelAspectRatio) == 0):
                self.setPixelAspect(lensNode, 1.0)
        elif (DPXCreator == "Nuke" and
                (ColorSpace == ""   or ColorSpace == "Other (0)") and
                (DPX0Transfer == "" or DPX0Transfer == "Other (0)")):
            #
            #  Nuke produces identical (uninformative) dpx headers for
            #  both Linear and Cineon files.  But we expect Cineon to be
            #  much more common, so go with that.
            #

            TransferFunction = "Cineon Log"
    elif (ext == 'TIF' and TransferFunction == ""):
        #
        #  Assume 8bit tif files are sRGB if there's no other indication;
        #  fall back to linear.
        #

        if (mInfo['bitsPerChannel'] == 8):
            TransferFunction = "sRGB"
        else:
            TransferFunction = "Linear"

    elif (ext in ['JPEG','JPG','MOV','AVI','MP4'] and TransferFunction == ""):
        #
        #  Assume jpeg/mov is in sRGB space if none is specified
        #

        TransferFunction = "sRGB"
    elif (ext in ['J2C','J2K','JPT','JP2'] and ColorSpacePrimaries == "UNSPECIFIED"):
        #
        #  If we're assuming XYZ primaries, but ignoring primaries just set
        #  transfer to sRGB.
        #

        if (igPrim):
            TransferFunction = "sRGB";

    if (igPrim):
        commands.setIntProperty(linNode + ".color.ignoreChromaticities", [1], True)

    if (ICCProfileDesc != ""):
        #
        #  Hack -- if you see sRGB in a color profile name just use the
        #  built-in sRGB conversion.
        #

        if ("sRGB" in ICCProfileDesc):
            TransferFunction = "sRGB"
        else:
            TransferFunction = ""
```

(continues on next page)

(continued from previous page)

```
        if (TIFFImageDescription == "Image converted using ifftoany"):
            #
            #  Get around maya bugs
            #

            print("WARNING: Assuming %s was created by Maya with a bad pixel aspect␣
↪ratio\n" % fileName)
            self.setPixelAspect(lensNode, 1.0)

        if (JPEGPixelAspect != "" and JPEGDensity != ""):
            info     = commands.sourceMediaInfo(source, fileName)
            attrPA   = float(JPEGPixelAspect)
            imagePA  = float(info['width']) / float(info['height'])
            testDiff = attrPA - 1.0 / imagePA

            if ((testDiff < 0.0001) and (testDiff > -0.0001)):
                #
                #  Maya JPEG -- fix pixel aspect
                #

                print("WARNING: Assuming %s was created by Maya with a bad pixel aspect␣
↪ratio\n" % fileName)
                self.setPixelAspect(lensNode, 1.0)

        if (EXIFOrientation != ""):
            #
            #  Some of these tags are beyond the internal image
            #  orientation choices so we need to possibly rotate, etc
            #

            if not self.definedInSessionFile(tformNode):
                rprop = tformNode + ".transform.rotate"
                if (EXIFOrientation == "right - top"):
                    commands.setFloatProperty(rprop, [90.0], True)
                elif (EXIFOrientation == "right - bottom"):
                    commands.setFloatProperty(rprop, [-90.0], True)
                elif (EXIFOrientation == "left - top"):
                    commands.setFloatProperty(rprop, [90.0], True)
                elif (EXIFOrientation == "left - bottom"):
                    commands.setFloatProperty(rprop, [-90.0], True)
```

At this point in the function the color space of the input image will be known or assumed to be linear. Finally, we try to set the color space (which will result in the image pixels being converted to the linear working space). If this succeeds, use sRGB display as the default.

```
        if (not noColorChanges):
        #
        #  Assume (in the absence of info to the contrary) any 8bit file will be in sRGB␣
↪space.
        #
            if (TransferFunction == "" and mInfo['bitsPerChannel'] == 8):
                TransferFunction = "sRGB"
```

(continues on next page)

(continued from previous page)

```
        #
        #  Allow user to override with environment variables
        #
        TransferFunction = self.checkEnvVar(ext, mInfo['bitsPerChannel'],␣
↪TransferFunction)

        if (self.setFileColorSpace(linNode, TransferFunction, ColorSpace)):

            #
            #  The default display correction is sRGB if the
            #  pixels can be converted to (or are already in)
            #  linear space
            #
            #  For gamma instead do this:
            #
            #      setFloatProperty("#RVDisplayColor.color.gamma", float[] {2.2}, true);
            #
            #  For a linear -> screen LUT do this:
            #
            #      readLUT(lutfile, "#RVDisplayColor", true);
            #
            #  If this is not the first source, assume that user or source_seetup
            #  has already set the desired display transform

            if len(commands.sources()) == 1:
                self.setDisplayFromProfile()
```

47.4 12.4 Setting up 3D and Channel LUTs

The default source-group-complete event function does not set up any non-built-in transforms. When you need to automatically apply a LUT, as a file, look, or a display LUT, you need to do the following:

```
readLUT(file, nodeName, True)
```

The nodeName will be ``#RVDisplayColor” (to refer to it by type) for the display LUT. For a file or look LUT, you use the associated node name for the color node — in the default sourceSetup() function this would be the linNode variable. The file parameter to readLUT() will be the name of the LUT file on disk and can be any of the LUT types that RV reads.

47.5 12.5 Setting CDL Values From File

As with using LUT files to fill in where built-in transforms do not cover your needs, you can read in CDL property values from a file. Use the following to read values from a CDL file on disk:

```
readCDL(file, nodeName, True)
```

When using readCDL the “nodeName” should be that of the targeted RVColor or RVLookLUT node to which you are applying the CDL values read from “file”. In the default RV graph you will find CDL properties to set in the RVColor and RVLookLUT nodes for each source, but there are none out-of-the-box in the display pipeline. However, you can add RVColor or RVLookLUT nodes to any pipeline you need CDL control that does not have them by default.You can also add RVCDL nodes where you want CDL control, but these nodes do not require the use of readCDL. With RVCDL nodes you only need to set the node's node.file property and it will automatically load and parse the file from the path provided. Errors will be thrown if the file provided is invalid.

47.6 12.6 Building a Package For Color Management

The recommend way to handle all event bindings is via a python package. To customize color management you can either create a new package from scratch as described here, or copy, rename, and hack the existing source_setup package.The use of source-group-complete is no different from any other event. By creating a package you can override the existing behavior or modify it. It also makes it possible to have layers of color management packages which (assuming they don't contradict each other) can collectively create a desired behavior.

```
from rv import rvtypes, commands, extra_commands
import os, re

class CustomColorManagementMode(rvtypes.MinorMode):

    def sourceSetup (self, event, noColorChanges=False):

        // do work on the new source here
        event.reject()

    def __init__(self):
        rvtypes.MinorMode.__init__(self)
        self.init("Source Setup",
                  None,
                  None,
                  [ ("source-group-complete", self.sourceSetup, "Color and Geometry␣
↪Management") ],
                  "source_setup",
                  20)

def createMode():
    return CustomColorManagementMode()
```

Note that we use the sortKey "source_setup" and the sortOrder "20". This will ensure that our additional sourceSetup runs after the default **color** management.The included optional package "ocio_source_setup" is a good example of a package that does additional source setup.

CHAPTER

FORTYEIGHT

CHAPTER 13 - NETWORK COMMUNICATION

RV can communicate with multiple external programs via its network protocol. The mechanism is designed to function like a "chat" client. Once a connection is established, messages can be sent and received including arbitrary binary data.There are a number of applications which this enables:

- **Controlling RV remotely.** E.g., a program which takes input from a dial and button board or a mobile device and converts the input into commands to start/stop playback or scrubbing in RV.
- **Synchronizing RV sessions across a network** . This is how RV's sync mode is implemented: each RV serves as a controller for the other.
- **Monitoring a Running RV** . For VFX theater dailies the RV session driving the dailies could be monitored by an external program. This program could then indicate to others in the facility when their shots are coming up.
- **A Display Driver for a Renderer** . Renders like Pixar's RenderMan have a plug-in called a display driver which is normally used to write out rendered frames as files. Frequently this type of plug-in is also used to send pixels to an external frame buffer (like RV) to monitor the renderer's progress in real time. It's possible to write a display driver that talks to RV using the network protocol and send it pixels as they are rendered. A more advanced version might receive feedback from RV (e.g. a selected rectangle on the image) in order to recommend areas the renderer should render sooner.

Any number of network connections can be established simultaneously, so for example it's possible to have a synchronized RV session with a remote RV and drive it with an external hardware device at the same time.

48.1 13.1 Example Code

There are two working examples that come with RV: the rvshell program and pyNetwork.py python example. The rvshell program uses a C++ library included with the distribution called TwkQtChat which you can use to make interfacing easier — especially if your program will use Qt. We highly recommend using this library since this is code which RV uses internally so it will always be up-to-date. The library is only dependent on the QtCore and QtNetwork modules.The pyNetwork example implements the network protocol using only python native code. You can use it directly in python programs.

48.1.1 13.1.1 Using rvshell

To use rvshell, start RV from the command line with the network started and a default port of 45000 (to make sure it doesn't interfere with existing RV sessions):

```
shell> rv -network -networkPort 45000
```

Next start the rvshell program program from a different shell:

```
shell> rvshell user localhost 45000
```

Assuming all went well, this will start rvshell connected to the running RV. There are three things you can experiment with using rvhell: a very simple controller interface, a script editor to send portions of script or messages to RV manually, and a display driver simulator that sends stereo frames to RV.Start by loading a sequence of images or a quicktime movie into RV. In rvshell switch to the "Playback Control" tab. You should be able to play, stop, change frames and toggle full screen mode using the buttons on the interface. This example sends simple Mu commands to RV to control it. The feedback section of the interface shows the RETURN message send back from RV. This shows whatever result was obtained from the command.The "Raw Event" section of the interface lets you assemble event messages to send to RV manually. The default event message type is remote-eval which will cause the message data to be treated like a Mu script to execute. There is also a remote-pyeval event which does the same with Python (in which case you should type in Python code instead of Mu code). Messages sent this way to RV are translated into UI events. In order for the interface code to respond to the event something must have bound a function to the event type. By default RV can handle remote-eval and remote-pyeval events, but you can add new ones yourself.When RV receieves a remote-eval event it executes the code and looks for a return value. If a return value exists, it converts it to a string and sends it back. So using remote-eval it's possible to querry RV's current state. For example if you load an image into RV and then send it the command renderedImages() it will return a Mu struct as a string with information about the rendered image. Similarily, sending a remote-pyeval with the same command will return a Python dictionary as a string with the same information.The last tab "Pixels" can be used to emulate a display driver. Load a JPEG image into rvshell's viewer (don't try something over 2k — rvshell is using Qt's image reader). Set the number of tiles you want to send in X and Y, for example 10 in each. In RV clear the session. In rvshell hit the Send Image button. rvshell will create a new stereo image source in RV and send the image one tile at a time to it. The left eye will be the original image and the right eye will be its inverse. Try View → Stereo → Side by Side to see the results.

48.1.2 13.1.2 Using rvNetwork.py

document here

48.2 13.2 TwkQtChat Library

The TwkQtChat library is composed of three classes: Client, Connection, and Server.

sendMessage	Generic method to send a standard UTF-8 text message to a specific contact
sendData	Generic method to send a data message to a specific contact
broadcastMessage	Send a standard UTF-8 message to all contacts
sendEvent	Send an EVENT or RETURNEVENT message to a contact (calls sendMessage)
broadcastEvent	Send an EVENT or RETURNEVENT message to all contacts
connectTo	Initiate a connection to a specific contact
hasConnection	Query connection status to a contact
disconnectFrom	Force the shutdown of connection
waitForMessage	Block until a message is received from a specific contact
waitForSend	Block until a message is actually sent
signOff	Send a DISCONNECT message to a contact to shutdown gracefully
online	Returns true of the Server is running and listening on the port

Table 13.1:Important Client Member Functions

newMessage	A new message has been received on an existing connection
newData	A new data message has been received on an existing connection
newContact	A new contact (and associated connection) has been established
contactLeft	A previously established connection has been shutdown
requestConnection	A remote program is requesting a connection
connectionFailed	An attempted connection failed
contactError	An error occurred on an existing connection

Table 13.2:Client Signals

A single Client instance is required to represent your process and to manage the Connections and Server instances. The Connection and Server classes are derived from the Qt QTcpSocket and QTcpServer classes which do the lower level work. Once the Client instance exists you can get pointer to the Server and existing Connections to directly manipulate them or connect their signals to slots in other QObject derived classes if needed.The application should start by creating a Client instance with its contact name (usually a user name), application name, and port on which to create the server. The Client class uses standard Qt signals and slots to communicate with other code. It's not necessary to inherit from it.The most important functions on the Client class are list in table *13.1* .

48.3 13.3 The Protocol

There are two types of messages that RV can receive and send over its network socket: a standard message and a data message. Data messages can send arbitrary binary data while standard messages are used to send UTF-8 string data.The greeting is used only once on initial contact. The standard message is used in most cases. The data message is used primarily to send binary files or blocks of pixels to/from RV.

48.3.1 13.3.1 Standard Messages

RV recognizes these types of standard messages:

MESSAGE	The string payload is subdivided into multiple parts the first of which indicates the sub-type of the message. The rest of the message is interpreted according to its sub-type.
GREETING	Sent by RV to a synced RV when negotiating the initial contact.
NEW-GREETING	Sent by external controlling programs to RV during initial contact.
PINGPONG-CONTROL	Used to negotiate whether or not RV and the connected process should exchange PING and PONG messages on a regular basis.
PING	Query the state of the other end of the connection — i.e. check and see if the other process is still alive and functioning.
PONG	Returned when a PING message is received to indicate state.

Table 13.3:Message TypesWhen an application first connects to RV over its TCP port, a greeting message is exchanged. This consists of an UTF-8 byte string composed of:

The string “NEWGREETING”	1st word
The UTF-8 value 32 (space)	-
A UTF-8 integer composed of the characters [0-9] with the value **N** + **M** + 1 indicating the number of bytes remaining in the message	2nd word
The UTF-8 value 32 (space)	-
Contact name UTF-8 string (non-whitespace)	**N** bytes
The UTF-8 value 32 (space)	1 byte
Application name UTF-8 string (non-whitespace)	**M** bytes

Table 13.4:Greeting MessageIn response, the application should receive a NEWGREETING message back. At this point the application will be connected to RV.A standard message is a single UTF-8 string which has the form:

The string “MESSAGE”	1st word
The UTF-8 value 32 (space)	-
A UTF-8 integer composed of the characters [0-9] the value of which is **N** indicating the size of the remaining message	2nd word
The UTF-8 value 32 (space)	-
The message payload (remaining UTF-8 string)	**N** bytes

Table 13.5:Standard MessageWhen RV receives a standard message (MESSAGE type) it will assume the payload is a UTF-8 string and try to interpret it. The first word of the string is considered the sub-message type and is used to decide how to respond:

EVENT	Send the rest of the payload as a UI event (see below)
RETURNEVENT	Same as EVENT but will result in a response RETURN message
RETURN	The message is a response to a recently received RETURNEVENT message
DISCONNECT	The connection should be disconnected

Table 13.6:Sub-Message TypesThe EVENT and RETURNEVENT messages are the most common. When RV receives an EVENT or RETURNEVENT message it will translate it into a user interface event. The additional part of the string (after EVENT or RETURNEVENT) is composed of:

EVENT or RETURNEVENT	UTF-8 string identifying the message as an EVENT or RETURNEVENT message.
space character	-
non-whitespace-event-name	The event that will be sent to the UI as a string event (e.g. remote-eval). This can be obtained from the event by calling event.name()in Mu or Python
space character	-
non-whitespace-target-name	Present for backwards compatibility only. We recommend you use a single “*” character to fill this slot.
space character	-
UTF-8 string	The string event contents. Retrievable with event.contents() in Mu or Python.

Table 13.7:EVENT MessagesFor example the full contents of an EVENT message might look like:

```
MESSAGE 34 EVENT my-event-name red green blue
```

The first word indicates a standard message. The next word (34) indicates the length of the rest of the data. EVENT is the message sub-type which further specifies that the next word (my-event-name) is the event to send to the UI with the rest of the string (red green blue) as the event contents.If a UI function that receives the event sets the return value and the message was a RETURNEVENT, then a RETURN will be sent back. A RETURN will have a single string that is the return value. An EVENT message will not result in a RETURN message.

RETURN	UTF-8 string identifying the message as an RETURN message.
space character	-
UTF-8 string	The string event returnContents(). This is the value set by setReturnContents() on the event object in Mu or Python.

Table 13.8:RETURN MessageGenerally, when a RETURNEVENT is sent to your application, a RETURN should be sent back because the other side may be blocked waiting. It's ok to send an empty RETURN. Normally, RV will not send EVENT or RETURNEVENT messages to other non-RV applications. However, it's possible that this could happen while connected to an RV that is also engaged in a sync session with another RV.Finally a DISCONNECT message comes with no additional data and signals that the connection should be closed.

Ping and Pong Messages

There are three lower level messages used to keep the status of the connection up to date. This scheme relies on each side of the connection returning a PONG message if it ever receives a PING message whenever ping pong messages are active.Whether or not it's active is controlled by sending the PINGPONGCONTROL message: when received, if the payload is the UTF-8 value "1" then PING messages should be expected and responded to. If the value is "0" then responding to a PING message is not mandatory.For some applications especially those that require a lot of computation (e.g. a display driver for a renderer) it can be a good to shut down the ping pong notification. When off, both sides of the connection should assume the other side is busy but not dead in the absence of network activity.

Message	Description	Full message value
PINGPONGCONTROL	A payload value of "1" indicates that PING and PONG messages should be used	PINGPONGCONTROL 1 (1 or 0)
PING	The payload is always the character "p". Should result in a PONG response	PING 1 p
PONG	The payload is always "p". Should be sent in response to a PING message	PONG 1 p

Table 13.9:PING and PONG Messages

48.3.2 13.3.2 Data Messages

The data messages come it two types: PIXELTILE and DATAEVENT. These take the form:

PIXELTILE(parameters) -or- DATAEVENT(parameters)	1st word
space character	-
A UTF-8 integer composed of the characters [0-9] the value of which is **N** indicating the size of the remaining message	2nd word
space character	-
Data of size **N**	**N** bytes

Table 13.10:PIXELTILE and DATAEVENTThe PIXELTILE message is used to send a block of pixels to or from RV. When received by RV the PIXELTILE message is translated into a pixel-block event (unless another event name is specified) which is sent to the user interface. This message takes a number of parameters which should have no whitespace characters and separated by commas (“,”):

w	Width of data in pixels.
h	Height of the data in pixels. (If the height of the block of pixels is 1 and the width is the width of the image, the block is equivalent to a scanline.)
x	The horizontal offset of the pixel block relative to the image origin
y	The vertical offset of the pixel block relative to the image origin
f	The frame number
event-name	Alternate event name (instead of pixel-block). RV will only recognize pixel-block event by default. You can bind to other events however.
media	The name of the media associated with data.
layer	The name of the layer associated with the meda. This is analogous to an EXR layer
view	The name of the view associated with the media. This is analogous to an EXR view

Table 13.11:PIXELTILE MessageFor example, the PIXELTILE header to the data message might appear as:

```
PIXELTILE(media=out.9.exr,layer=diffuse,view=left,w=16,h=16,x=160,y=240,f=9)
```

Which would be parsed and used to fill fields in the Event type. This data becomes available to Mu and Python functions binding to the event. By default the Event object is sent to the insertCreatePixelBlock() function which fins the image source associated with the meda and inserts the data into the correct layer and view of the image. Each of the keywords in the PIXELTILE header is optional.The DATAEVENT message is similar to the PIXELTILE but is intended to be implemented by the user. The message header takes at least three parameters which are ordered (no keywords like PIXELTILE). RV will use only the first three parameters:

event-name	RV will send a raw data event with this name
target	Required but not currently used
content type string	An arbitrary string indicating the type of the content. This is available to the UI from the Event.contentType() function.

Table 13.12:DATAEVENT MessageFor example, the DATAEVENT header might appear as:

```
DATAEVENT(my-data-event,unused,special-data)
```

Which would be sent to the user interface as a my-data-event with the content type “special-data”. The content type is retrievable with Event.contentType(). The data payload is available via Event.dataContents() method.

CHAPTER

FORTYNINE

CHAPTER 14 - WEBKIT JAVASCRIPT INTEGRATION

RV can communicate with JavaScript running in a Qt WebEngine widget. This makes it possible to serve custom RV-aware web pages which can interact with a running RV. JavaScript running in the web page can execute arbitrary Mu script strings as well as receive events from RV.You can experiment with this using the example webview package included with RV.

49.1 14.1 Executing Mu or Python from JavaScript

RV exports a JavaScript object called rvsession to the Javascript runtime environment. Two of the functions in that namespace are evaluate() and pyevaluate(). By calling evaluate() or pyevaluate() or pyexec() you can execute arbitrary Mu or Python code in the running RV to control it. If the executed code returns a value, the value will be converted to a string and returned by the (py)evaluate() functions. Note that pyevaluate() triggers a python eval which takes an expression and returns a value. pyexec() on the other hand takes an arbitrary block of code and triggers a python exec call.As an example, here is some html which demonstates creating a link in a web page which causes RV to start playing when pressed:

```
<script type="text/javascript">
function play () { rvsession.evaluate("play()"); }
</script>

<p><a href="javascript:play()">Play</a></p>
```

If inlining the Mu or Python code in each call back becomes onerous you can upload function definitions and even whole classes all in one evaluate call and then call the defined functions later. For complex applications this may be the most sane way to handle call back evaluation.

49.2 14.2 Getting Event Call Backs in JavaScript

RV generates events which can be converted into call backs in JavaScript. This differs slightly from how events are handled in Mu and Python.

Signal	Events
eventString	Any internal RV event and events generated by the command sendInternalEvent() command in Mu or Python
eventKey	Any key- event (e.g. key-down–a)
eventPointer	Any pointer- event (e.g. pointer-1–push) or tablet event (e.g. stylus-pen–push)
eventDrag-Drop	Any dragdrop- event

Table 14.1:JavaScript Signals Produced by Events

The rvsession object contains signal objects which you can connect by supplying a call back function. In addition you need to supply the name of one or more events as a regular expression which will be matched against incoming events. For example:

```
function callback_string (name, contents, sender)
{
    var x = name + " " + contents + " " + sender;
    rvsession.evaluate("print(\"callback_string " + x + "\\n\");");
}

rvsession.eventString.connect(callback_string);
rvsession.bindToRegex("source-group-complete");
```

connects the function callback_string() to the eventString signal object and binds to the source-group-complete RV event. For each event the proper signal object type must be used. For example pointer events are not handled by eventString but by the eventPointer signal. There are four signals available: eventString, eventKey, eventPointer, and eventDragDrop. See tables describing which events generate which signals and what the signal call back arguments should be.In the above example, any time media is loaded into RV the callback_string() function will be called. Note that there is a single callback for each type of event. In particular if you want to handle both the "new-source" and the "frame-changed" events, your eventString handler must handle both (it can distinguish between them using the "name" parameter passed to the handler. To bind the handler to both events you can call "bindToRegex" multiple times, or specify both events in a regular expression:

```
rvsession.bindToRegex("source-group-complete|frame-changed");
```

The format of this regular expression is specified on the qt-project website .

Argument	Description
eventName	The name of the RV event. For example " **source-group-complete** "
contents	A string containing the event contents if it has any
senderName	Name of the sender if it has one

Table 14.2:eventString Signal Arguments

Argument	Description
eventName	The name of the RV event. For example " **source-group-complete** "
key	An integer representing the key symbol
modifiers	An integer the low order five bits of which indicate the keyboard modifier state

Table 14.3:eventKey Signal Arguments

Argument	Description
eventName	The name of the RV event. For example " **source-group-complete** "
x	The horizontal position of the mouse as an integer
y	The vertical position of the mouse as an integer
w	The width of the event domain as an integer
h	The height of the event domain as an integer
startX	The starting horizontal position of a mouse down event
startY	The starting vertical position of a mouse down event
buttonStates	An integer the lower order five bits of which indicate the mouse button states
activationTime	The relative time at which button activation occurred or 0 for regular pointer events

Table 14.4:eventPointer Signal Arguments

Argument	Description
eventName	The name of the RV event. For example " **source-group-complete** "
x	The horizontal position of the mouse as an integer
y	The vertical position of the mouse as an integer
w	The width of the event domain as an integer
h	The height of the event domain as an integer
startX	The starting horizontal position of a mouse down event
startY	The starting vertical position of a mouse down event
buttonStates	An integer the lower order five bits of which indicate the mouse button states
dragDropType	A string the value of which will be one of "enter", "leave", "move", or "release"
contentType	A string the value of which will be one of "file", "url", or "text"
stringContent	The contents of the drag and drop event as a string

Table 14.5:eventDragDrop Signal Arguments

49.3 14.3 Using the webview Example Package

This package creates one or more docked Qt WebEngine instances, configurable from the command line as described below. JavaScript code running in the webviews can execute arbitrary Mu code in RV by calling the rvsession.evaluate() function. This package is intended as an example.These command-line options should be passed to RV after the -flags option. The webview options below are shown with their default values, and all of them can apply to any of four webviews in the Left, Right, Top, and Bottom dock locations.

```
shell> rv -flags ModeManagerPreload=webview
```

The above forces the load of the webview package which will display an example web page. Additional arguments can be supplied to load specific web pages into additional panes. While this will just show the sample html/javascript file that comes with the package in a webview docked on the right. To see what's happening in this example, bring up the Session Manager so you can see the Sources appearing and disappearing, or switch to the defaultLayout view. Note that you can play while reconfiguring the session with the javascript checkboxes.The following additional arguments can be passed via the -flags mechanism. In the below, **POS** should be replaced by one of Left, Right, Bottom, or Top.

ModeManagerPreload=webview

Force loading of the webview package. The package should not be loaded by default but does need to be installed. This causes rv to treat the package as if it were loaded by the user.

webviewUrlPOS=URL

A webview pane will be created at POS and the URL will be loaded into it. It can be something from a web server or a file:// URL. If you force the package to load, but do not specify any URL, you'll get a single webview in the Right dock lockation rendering the sample html/javascript page that ships with the package. Note that the string "EQUALS" will be replaced by an "=" character in the URL.

webviewTitlePOS=string

Set the title of the webview pane to string.

webviewShowTitlePOS=true or false

A value of true will show and false will remove the title bar from the webview pane.

webviewShowProgressPOS=true or false

Show a progress bar while loading for the web pane.

webviewSizePOS=integer

Set the width (for right and left panes) or height (for top and bottom panes) of the web pane.

An example using all of the above:

```
shell> rv -flags ModeManagerPreload=webview \
     webviewUrlRight=file:///foo.html \
     webviewShowTitleRight=false \
     webviewShowProgressRight=false \
     webviewSizeRight=200 \
     webviewUrlBottom=file:///bar.html \
     webviewShowTitleBottom=false \
     webviewShowProgressBottom=false \
     webviewSizeBottom=300
```

CHAPTER

FIFTY

CHAPTER 15 - HIERARCHICAL PREFERENCES

Each RV user has a Preferences file where her personal rv settings are stored. Most preferences are viewed and edited with the Preferences dialog (accessed via the RV menu), but preferences can also be programmatically read and written from custom code via the readSetting and writeSetting Mu commands. The preferences files are stored in different places on different platforms.

Platform	Location
macOS	$HOME/Library/Preferences/com.autodesk.OpenRV.plist
Linux	$HOME/.config/Autodesk/OpenRV.conf
Windows 7	~$HOME/AppData/Roaming/Autodesk/OpenRV.ini

Initial values of preferences can be overridden on a site-wide or show-wide basis by setting the environment variable RV_PREFS_OVERRIDE_PATH to point to one or more directories that contain files of the name and type listed in the above table. For example, if you have your RV.conf file under $HOME/Documents/ you can set RV_PREFS_OVERRIDE_PATH=$HOME/Documents. Each of these overriding preferences file can provide default values for one or more preferences. A value from one of these overriding files will override the users's preference only if the user's preferences file has no value for this preference yet.In the simplest case, if you want to provide overriding initial values for all preferences, you should

1. Delete your preferences file.
2. Start RV, go to the Preferences dialog, and adjust any preferences you want.
3. Close the dialog and exit RV.
4. Copy your preferences file into the RV_PREFS_OVERRIDE_PATH.

If you want to override at several levels (say per-site and per-show), you can add preferences files to any number of directories in the PATH, but you'll have to edit them so that each only contains the preferences you want to override with that file. Preferences files found in directories earlier in the path will override those found in later directories.Note that this system only provides the ability to override initial settings for the preferences. Nothing prevents the user from changing those settings after initialization.It's also possible to create show/site/whatever-specific preferences files that **always** clobber the user's personal preferences. This mechanism is exactly analogous to the above, except that the name of the environment variable that holds paths to clobbering prefs files is RV_PREFS_CLOBBER_PATH. Again, the user can freely change any "live" values managed in the Preferences dialog, but in the next run, the clobbering preferences will again take precedence. Note that a value from a clobbering file (at any level) will take precedence over a value from an overriding file (at any level).

CHAPTER

FIFTYONE

CHAPTER 16 - NODE REFERENCE

This chapter has a section for each type of node in RV's image processing graph. The properties and descriptions listed here are the default properties. Any top level node that can be seen in the session manager can have the "name" property of the "ui" component set in order to control how the node is listed.

51.1 RVCache

The RVCache node has no external properties.

51.2 RVCacheLUT and RVLookLUT

The RVCacheLUT is applied in software before the image is cached and before any software resolution and bit depth changes. The RVLookLUT is applied just before the display LUT but is per-source.

Property	Type	Size	Description
lut.lut	float	div 3	Contains either a 3D or a channel look LUT
lut.prelut	float	div 3	Contains a channel pre-LUT
lut.inMatri	float	16	Input color matrix
lut.outMat	float	16	Output color matrix
lut.scale	float	1	LUT output scale factor
lut.offset	float	1	LUT output offset
lut.file	string	1	Path of LUT file to read when RV session is loaded
lut.size	int	1 or 3	With 1 size value, the look LUT is a channel LUT of the specified size, if there are 3 values the look LUT is a 3D LUT with the dimensions indicated
lut.active	int	1	If non-0 the LUT is active
lut:output.s	int	1 or 3	The resampled LUT output size
lut:output.l	float or half	div 3	The resampled output LUT
lut:output.p	float or half	div 3	The resampled output pre-LUT

51.3 RVCDL

This node can be used to load CDL properties from CCC, CC, and CDL files on disk.

Property	Type	Size	Description
node.active	int	1	If non-0 the CDL is active. A value of 0 turns off the node.
node.colorspace	string	1	Can be “rec709”, “aces”, or “aceslog” and the default is “rec709”.
node.file	string	1	Path of CCC, CC, or CDL file from which to read properties.
node.slope	float[3]	1	Color Decision List per-channel slope control
node.offset	float[3]	1	Color Decision List per-channel offset control
node.power	float[3]	1	Color Decision List per-channel power control
node.saturation	float	1	Color Decision List saturation control
node.noClamp	int	1	Set to 1 to remove clamping from CDL equations

51.4 RVChannelMap

This node can be used to remap channels that may have been labeled incorrectly.

Property	Type	Size	Description
for-mat.cha	string	>= 0	An array of channel names. If the property is empty the image will pass though the node unchanged. Otherwise, only those channels appearing in the property array will be output. The channel order will be the same as the order in the property.

51.5 RVColor

The color node has a large number of color controls. This node is usually evaluated on the GPU, except when normalize is 1. The CDL is applied after linearization and linear color changes.

Property	Type	Size	Description
color.normalize	int	1	Non-0 means to normalize the incoming pixels to [0,1]
color.invert	int	1	If non-0, invert the image color using the inversion matrix (See User's Manual)
color.gamma	float[:	1	Apply a gamma. The default is [1.0, 1.0, 1.0]. The three values are applied to R G and B channels independently.
color.offset	float[:	1	Color bias added to incoming color channels. Default = 0 (not bias). Each component is applied to R G B independently.
color.scale	float[:	1	Scales each channel by the respective float value.
color.exposure	float[:	1	Relative exposure in stops. Default = [0, 0, 0], See user's manual for more information on this. Each component is applied to R G and B independently.
color.contrast	float[:	1	Contrast applied per channel (see User's Manual)
color.saturation	float	1	Relative saturation (see User's Manual)
color.hue	float	1	Hue rotation in radians (see User's Manual)
color.active	int	1	If 0, do not apply any color transforms. Turns off the node.
CDL.slope	float[:	1	Color Decision List per-channel slope control
CDL.offset	float[:	1	Color Decision List per-channel offset control
CDL.power	float[:	1	Color Decision List per-channel power control
CDL.saturation	float	1	Color Decision List saturation control
CDL.noClamp	int	1	Set to 1 to remove clamping from CDL equations
luminance-LUT.lut	float	div 3	Luminance LUT to be applied to incoming image. Contains R G B triples one after another. The LUT resolution
luminance-LUT.max	float	1	A scale on the output of the Luminance LUT
luminance-LUT.active	int	1	If non-0, luminance LUT should be applied
luminance-LUT:output.size	int	1	Output Luminance lut size
luminance-LUT:output.lut	float	div 3	Output resampled luminance LUT

51.6 RVDispTransform2D

This node is used to do any scaling or translating of the corresponding view group.

Property	Type	Size	Description
transform.translate	float[2]	1	Viewing translation
transform.scale	float[2]	1	Viewing scale

51.7 RVDisplayColor

This node is used by default by any display group as part of its color management pipeline.

Prop-erty	Type	Size	Description
color.cha	string	1	A four character string containing any of the characters [RGBA10]. The order allows permutation of the normal R G B and A channels as well as filling any channel with 1 or 0.
color.cha	int	1	If 0 pass the channels through as they are. When the value is 1, 2, 3, or 4, the R G B or A channels are used to flood the R G and B channels. When the value is 5, the luminance of each pixel is computed and displayed as a gray scale image.
color.gar	float	1	A single gamma value applied to all channels, default = 1.0
color.sR(	int	1	If non-0 a linear to sRGB space transform occurs
color.Re	int	1	If non-0 the Rec709 transfer function is applied
color.bri	float	1	In relative stops, the final pixel values are brightened or dimmed according to this value. Occurs after all color space transforms.
color.out	int	1	If non-0 pass pixels through an out of range filter. Channel values in the (0,1] are set to 0.5, channel values [-inf,0] are set to 0 and channel values (1,inf] are set to 1.0.
color.act	int	1	If 0 deactivate the display node
lut.lut	float	div 3	Contains either a 3D or a channel display LUT
lut.prelu	float	div 3	Contains a channel pre-LUT
lut.scale	float	1	LUT output scale factor
lut.offset	float	1	LUT output offset
lut.inMa	float	16	Input color matrix
lut.outM	float	16	Output color matrix
lut.file	string	1	Path of LUT file to read when RV session is loaded
lut.size	int	1 or 3	With 1 size value, the display LUT is a channel LUT of the specified size, if there are 3 values the display LUT is a 3D LUT with the dimensions indicated
lut.active	int	1	If non-0 the display LUT is active
lut:outpu	int	1 or 3	The resampled LUT output size
lut:outpu	float or half	div 3	The resampled output LUT
lut:outpu	float or half	div 3	The resampled output pre-LUT

51.8 RVDisplayGroup and RVOutputGroup

The display group provides per device display conditioning. The output group is the analogous node group for RVIO. The display groups are never saved in the session, but there is only one output group and it is saved for RVIO. There are no user external properties at this time.

51.9 RVDisplayStereo

This node governs how to handle stereo playback including controlling the placement of stereo sources.

Property	Type	Size	Description
rightTransform.flip	int	1	Flip the right eye top to bottom.
rightTransform.flop	int	1	Flop the right eye left to right.
rightTransform.rotate	float	1	Rotation of right eye in degrees.
rightTransform.translate	float[2]	1	Translation offset in X and Y for the right eye.
stereo.relativeOffset	float	1	Relative stereo offset for both eyes.
stereo.rightOffset	float	1	Stereo offset for right eye only.
stereo.swap	int	1	If set to 1 treat left eye as right and right eye as left.
stereo.type	string	1	Stereo mode in use. For example: left, right, pair, mirror, scanline, anaglyph, checker… (default is off)

51.10 RVFileSource

The source node controls file I/O and organize the source media into layers (in the RV sense). It has basic controls needed to mix the layers together.

Name	Type	Size	Description
media.mov	string	> 1	The movie, image, audio files and image sequence names. Each name is a layer in the source.There is typically at least one value in this property
group.f	float	1	Overrides the fps found in any movie or image file or if none is found overrides the default fps of 24.
group.v	float	1	Relative volume. This can be any positive number or 0.
group.a	float	1	Audio offset in seconds. All audio layers will be offset.
group.r	int	1	Shifts the start and end frame numbers of all image media in the source.
group.r	int	1	Resets the start frame of all image media to given value. This is an optional property. It must be created to be set and removed to unset.
group.b	float	1	Range of [-1,1]. A value of 0 means the audio volume is the same for both the left and right channels.
group.n	int	1	Do not use audio tracks in movies files
cut.in	int	1	The preferred start frame of the sequence/movie file
cut.out	int	1	The preferred end frame of the sequence/movie file
request.re	int	1	If the value is 1 and the image format can read multiple channels, it is requested to read all channels in the current image layer and view.
request.in	string	2, 3, or 4	This array is of the form: type, view, [layer[, channel]]. The type describes what is defined in the remainder of the array. The type may be one of ”view”, ”layer”, or ”channel”. The 2nd element of the array must be defined and is the value of the view. If there are 3 elements defined then the 3rd is the layer name. If there are 4 elements defined then the 4th is the channel name.
request.st	string	0 or 2	If there are values in this property, they will be passed to the image reader when in stereo viewing mode as requested view names for the left and right eyes.
attributes	string int, or float	1	This optional container of properties will get automatically included in the metadata associated with the source. The key can be any string and will be displayed as the metadata item name when displayed in the Image Info. The value of the property will be displayed as the value of the metadata.

51.11 RVFolderGroup

The folder group contains either a SwitchGroup or LayoutGroup which determines how it is displayed.

Name	Type	Size	Description
ui.name	string	1	This is a user specified name which appears in the user interface.
mode.viewType	string	1	Either “switch” or “layout”. Determines how the folder is displayed.

51.12 RVFormat

This node is used to alter geometry or color depth of an image source. It is part of an RVSourceGroup.

Property	Type	Size	Description
geometry.xfit	int	1	Forces the resolution to a specific width
geometry.yfit	int	1	Forces the resolution to a specific height
geome-try.xresize	int	1	Forces the resolution to a specific width
geome-try.yresize	int	1	Forces the resolution to a specific height
geome-try.scale	float	1	Multiplier on incoming resolution. E.g., 0.5 when applied to 2048x1556 results in a 1024x768 image.
geome-try.resampleMe	string	1	Method to use when resampling. The possible values are area, cubic, and linear,
crop.active	int	1	If non-0 cropping is active
crop.xmin	int	1	Minimum X value of crop in pixel space
crop.ymin	int	1	Minimum Y value of crop in pixel space
crop.xmax	int	1	Maximum X value of crop in pixel space
crop.ymax	int	1	Maximum Y value of crop in pixel space
uncrop.active	int	1	In non-0 uncrop region is used
uncrop.x	int	1	X offset of input image into uncropped image space
uncrop.y	int	1	Y offset of input image into uncropped image space
uncrop.width	int	1	Width of uncropped image space
uncrop.height	int	1	Height of uncropped image space
color.maxBitDe	int	1	One of 8, 16, or 32 indicating the maximum allowed bit depth (for either float or integer pixels)
color.allowFloa	int	1	If non-0 floating point images will be allowed on the GPU otherwise, the image will be converted to integer of the same bit depth (or the maximum bit depth).

51.13 RVImageSource

The RV image source is subset of what RV can handle from an external file (basically just EXR). Image sources can have multiple views each of which have multiple layers. However, all views must have the same layers. Image sources cannot have layers within layers, orphaned channels, empty views, missing views, or other weirdnesses that EXR can have.

Name	Type	Size	Description
me- dia.movi	string	> 1	The movie, image, audio files and image sequence names. Each name is a layer in the source.There is typically at least one value in this property.
me- dia.name	string	1	The name for this image.
cut.in	int	1	The preferred start frame of the sequence/movie file.
cut.out	int	1	The preferred end frame of the sequence/movie file.
im- age.chan	string	1	String representing the channels in the image.
im- age.layer	string	> 1	List of strings representing the layers in the image.
im- age.view	string	> 1	List of strings representing the views in the image.
im- age.defau	string	1	String representing the layer from image.layers that should be treated as default layer.
im- age.defau	string	1	String representing the view from image.views that should be treated as default view.
im- age.start	int	1	First frame of the source.
im- age.end	int	1	Last frame of the source.
im- age.inc	int	1	Number of frames to step by.
im- age.fps	float	1	Frame rate of source in float ratio of frames per second.
im- age.pixel	float	1	Image aspect ratio as a float of width over height.
im- age.uncro	int	1	Height of uncropped image space.
im- age.uncro	int	1	Width of uncropped image space.
im- age.uncro	int	1	X offset of image into uncropped image space.
im- age.uncro	int	1	Y offset of image into uncropped image space.
im- age.widtl	int	1	Image width in integer pixels.
im- age.heigl	int	1	Image height in integer pixels.
re- quest.ima	string	Any	Any values are considered image channel names. These are passed to the image readers with the request that only these layers be read from the image pixels.
re- quest.ima	string	2, 3, or 4	This array is of the form: type, view, [layer[, channel]]. The type describes what is defined in the remainder of the array. The type may be one of ”view”, ”layer”, or ”channel”. The 2nd element of the array must be defined and is the value of the view. If there are 3 elements defined then the 3rd is the layer name. If there are 4 elements defined then the 4th is the channel name.
re- quest.ster	string	0 or 2	If there are values in this property, they will be passed to the image reader when in stereo viewing mode as requested view names for the left and right eyes.
at- tributes.k	string int, or float	1	This optional container of properties will get automatically included in the metadata associated with the source. The key can be any string and will be displayed as the metadata item name when displayed in the Image Info. The value of the property will be displayed as the value of the metadata.

51.14 RVLayoutGroup

The source group contains a single chain of nodes the leaf of which is an RVFileSource or RVImageSource. It has a single property.

Name	Type	Size	Description
ui.name	string	1	This is a user specified name which appears in the user interface.
layout.mode	string	1	The string mode that dictates the way items are layed out. Possible values are: packed, packed2, row, column, and grid (default is packed).
layout.spacin	float	1	Scale the items in the layout. Legal values are between 0.0 and 1.0.
layout.gridCo	int	1	When in grid mode constrain grid to this many columns. If this set to 0, then the number of columns will be determined by gridRows. If both are 0, then both will be automatically calculated.
layout.gridRo	int	1	When in grid mode constrain grid to this many rows. If this is set to 0, then the number of rows will be determined by gridColumns. This value is ignored when gridColumns is non-zero.
timing.retimel	int	1	Retime all inputs to the output fps if 1 otherwise play back their frames one at a time at the output fps.

51.15 RVLensWarp

This node handles the pixel aspect ratio of a source group. The lens warp node can also be used to perform radial and/or tangential distortion on a frame. It implements the Brown's distortion model (similar to that adopted by OpenCV or Adobe Lens Camera Profile model) and 3DE4's Anamorphic Degree6 model. This node can be used to perform operations like lens distortion or artistic lens warp effects.

Name	Type	Size	Description
warp.pixelAspectRatio	float	1	If non-0 set the pixel aspect ratio. Otherwise use the pixel aspect ratio reported by the inco
warp.model	string		Lens model: choices are “brown”, “opencv”, “pfbarrel”, “adobe”, “3de4_anamorphic_deg
warp.k1	float	1	Radial coefficient for r^2 (default 0.0)Applicable to “brown”, “opencv”, “pfbarrel”, “adob
warp.k2	float	1	Radial coefficient for r^4 (default 0.0)Applicable to “brown”, “opencv”, “pfbarrel”, “adob
warp.k3	float	1	Radial coefficient for r^6 (default 0.0)Applicable to “brown”, “opencv”, “adobe”.
warp.p1	float	1	First tangential coefficient (default 0.0)Applicable to “brown”, “opencv”, “adobe”.
warp.p2	float	1	Second tangential coefficient (default 0.0)Applicable to “brown”, “opencv”, “adobe”.
warp.cx02	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cy02	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cx22	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cy22	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cx04	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cy04	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cx24	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cy24	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cx44	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cy44	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cx06	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cy06	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)
warp.cx26	float	1	Applicable to “3de4_anamorphic_degree_6”. (default 0.0)

Table 1 – continued from previous page

Name	Type	Size	Description
warp.cy26	float	1	Applicable to "3de4_anamorphic_degree_6". (default 0.0)
warp.cx46	float	1	Applicable to "3de4_anamorphic_degree_6". (default 0.0)
warp.cy46	float	1	Applicable to "3de4_anamorphic_degree_6". (default 0.0)
warp.cx66	float	1	Applicable to "3de4_anamorphic_degree_6". (default 0.0)
warp.cy66	float	1	Applicable to "3de4_anamorphic_degree_6". (default 0.0)
warp.center	float[2]	1	Position of distortion center in normalized values [0. . . 1] (default [0.5 0.5]. Applicable to
warp.offset	float[2]	1	Offset from distortion center in normalized values [0. . . 1.0] (default [0.0 0.0]). Applicable
warp.fx	float	1	Normalized FocalLength in X (default 1.0).Applicable to "brown", "opencv", "adobe", "3
warp.fy	float	1	Normalized FocalLength in Y (default 1.0).Applicable to "brown", "opencv", "adobe", "3
warp.cropRatioX	float	1	Crop ratio of fovX (default 1.0). Applicable to all models.
warp.cropRatioY	float	1	Crop ratio of fovY (default 1.0). Applicable to all models.
node.active	int	1	If 0, do not apply any warp/pixel aspect ratio transform. Turns off the node. (default 1)

Example use case: Using OpenCV to determine lens distort parameters for RVLensWarp node based on GoPro footage. First capture some footage of a checkboard with your GoPro. Then you can use OpenCV camera calibration approach on this footage to solve for k1,k2,k3,p1 and p2. In OpenCV these numbers are reported back as follows. For example our 1920x1440 Hero3 Black GoPro solve returned:

```
fx=829.122253 0.000000 cx=969.551819
0.000000 fy=829.122253 cy=687.480774
0.000000 0.000000 1.000000
k1=-0.198361 k2=0.028252 p1=0.000092 p2=-0.000073
```

The OpenCV camera calibration solve output numbers are then translated/normalized to the RVLensWarpode property values as follows:

```
warp.model = "opencv"
warp.k1 = k1
warp.k2 = k2
warp.p1 = p1
warp.p2 = p2
warp.center = [cx/1920 cy/1440]
warp.fx = fx/1920
warp.fy = fy/1920
```

e.g. mu code:

```
set("#RVLensWarp.warp.model", "opencv");
set("#RVLensWarp.warp.k1", -0.198361);
set("#RVLensWarp.warp.k2", 0.028252);
set("#RVLensWarp.warp.p1", 0.00092);
set("#RVLensWarp.warp.p2", -0.00073);
setFloatProperty("#RVLensWarp.warp.offset", float[]{0.505, 0.4774}, true);
set("#RVLensWarp.warp.fx", 0.43185);
set("#RVLensWarp.warp.fy", 0.43185);
```

Example use case: Using Adobe LCP (Lens Camera Profile) distort parameters for RVLensWarp node. Adobe LCP files can be located in '/Library/Application Support/Adobe/CameraRaw/LensProfiles/1.0' under OSX. Adobe LCP file parameters maps to the RVLensWarp node properties as follows:

```
    warp.model = "adobe"
    warp.k1 = stCamera:RadialDistortParam1
    warp.k2 = stCamera:RadialDistortParam2
    warp.k3 = stCamera:RadialDistortParam3
    warp.p1 = stCamera:TangentialDistortParam1
    warp.p2 = stCamera:TangentialDistortParam2
    warp.center = [stCamera:ImageXCenter stCamera:ImageYCenter]
    warp.fx = stCamera:FocalLengthX
    warp.fy = stCamera:FocalLengthY
```

51.16 RVLinearize

The linearize node has a large number of color controls. The CDL is applied before linearization occurs.

Property	Type	Size	Description
color.alpha	int	1	By default (0), uses the alpha type reported by the incoming image. Otherwise, 1 means the alpha is premultiplied, 0 means the incoming alpha is unpremultiplied.
color.YUV	int	1	If the value is non-0, convert the incoming pixels from YUV space to linear space.
color.logty	int	1	The default (0), means no log to linear transform, 1 uses the cineon transform (see cineon.whiteCodeValue and cineon.blackCodeValue below), 2 means use the Viper camera log to linear transform, and 3 means use LogC log to linear transform.
color.sRGE	int	1	If the value is non-0, convert the incoming pixels from sRGB space to linear space.
color.Rec7(	int	1	If the value is non-0, convert the incoming pixels using the inverse of the Rec709 transfer function.
color.fileGa	float	1	Apply a gamma to linearize the incoming image. The default is 1.0.
color.active	int	1	If 0, do not apply any color transforms. Turns off the node.
color.ignor	int	1	If non-0, ignore any non-Rec 709 chromaticities reported by the incoming image.
CDL.slope	float[	1	Color Decision List per-channel slope control.
CDL.offset	float[	1	Color Decision List per-channel offset control.
CDL.powe	float[	1	Color Decision List per-channel power control.
CDL.satura	float	1	Color Decision List saturation control.
CDL.noCla	int	1	Set to 1 to remove clamping from CDL equations.
CDL.active	int	1	If non-0 the CDL is active.
lut.lut	float	div 3	Contains either a 3D or a channel file LUT.
lut.prelut	float	div 3	Contains a channel pre-LUT.
lut.inMatri	float	16	Input color matrix.
lut.outMatr	float	16	Output color matrix.
lut.scale	float	1	LUT output scale factor.
lut.offset	float	1	LUT output offset.
lut.file	string	1	Path of LUT file to read when RV session is loaded.
lut.size	int	1 or 3	With 1 size value, the file LUT is a channel LUT of the specified size, if there are 3 values the file LUT is a 3D LUT with the dimensions indicated.
lut.active	int	1	If non-0 the file LUT is active.
lut:output.s	int	1 or 3	The resampled LUT output size.
lut:output.l	float or half	div 3	The resampled output LUT.

51.17 OCIO (OpenColorIO), OCIOFile, OCIOLook, and OCIODisplay

OpenColorIO nodes can be used in place of existing RV LUT pipelines. Properties in RVColorPipelineGroup, RVLinearizePipelineGroup, RVLookPipelineGroup, and RVDisplayPipelineGroup determine whether or not the OCIO nodes are used. All OCIO nodes have the same properties and function, but their location in the color pipeline is determined by their type. The exception is the generic OCIO node which can be created by the user and used in any context.

For more information, see *Chapter 11 - OpenColorIO*

Property	Type	Size	Description
ocio.lut	float	div 3	Contains a 3D LUT, size determined by ocio.lut3DSize
lut.prelut	float	div 3	Currently unused
ocio.active	int	1	Non-0 means node is active
ocio.lut3DSize	int	1	3D LUT size of all dimensions (default is 32)
ocio.inSpace	string	1	Name of OCIO input colorspace
ocio_context. *name*	string	1	Name/Value pairs for OCIO context

51.18 RVOverlay

Overlay nodes can be used with any source. They can be used to draw arbitrary rectangles and text over the source but beneath any annotations. Overlay nodes can hold any number of 3 types of components: **rect** components describe a rectangle to be rendered, **text** components describe a string (or an array of strings, one per frame) to be rendered, and **window** components describe a matted region to be indicated either by coloring the region outside the window, or by outlining it. The coordiates of the corners of the window may be animated by specifying one number per frame.In the below the “ **id** ” in the component name can be any string, but must be different for each component of the same type.

Property	Type	Size	Description
overlay.nextRectId	int	1	(unused)
overlay.nextTextId	int	1	(unused)
overlay.show	int	1	If 1 display any rectangles/text/window entries. If 0 do not.
matte.show	int	1	If 1 display the source specific matte, not the global
matte.aspect	float	1	Aspect ratio of the source’s matte
matte.opacity	float	1	Opacity of the source’s matte
matte.heightVisible	float	1	Fraction of the source height that is still visible from the matte.
matte.centerPoint	float[2]	1	The center of the matte stored as X, Y in normalized coordinates.
rect: *id* .color	float[4]	1	The color of the rectangle
rect: *id* .width	float	1	The width of the rectangle in the normalized coordinate system
rect: *id* .height	float	1	The height of the rectangle in the normalized coordinate system
rect: *id* .position	float[2]	1	Location of the rectangle in the normalized coordinate system
rect: *id* .active	int	1	If 0, rect will not be rendered
rect: *id* .eye	int	1	If absent, or set to 2, the rectangle will be rendered in both stereo eyes. If set to 0 or 1
text: *id* .pixelScale	float[2]	1	X and Y scaling factors for position, IE expected source resolution, if present and non
text: *id* .position	float[2]	1	Location of the text (coordinate are normalized unless pixelScale is set, in which case
text: *id* .color	float[4]	1	The color of the text
text: *id* .spacing	float	1	The spacing of the text
text: *id* .size	float	1	The size of the text
text: *id* .scale	float	1	The scale of the text
text: *id* .rotation	float	1	(unused)
text: *id* .font	string	1	The path to the .ttf (TrueType) font to use (Default is Luxi Serif)
text: *id* .text	string	N	Text to be rendered, if multi-valued there should be one string per frame in the expect
text: *id* .origin	string	1	The origin of the text box. The position property will store the location of the origin,
text: *id* .eye	int	1	If absent, or set to 2, the rectangle will be rendered in both stereo eyes. If set to 0 or 1
text: *id* .active	int	1	If active is 0, the text item will not be rendered
text: *id* .firstFrame	int	1	If the “text” property is multi-valued, this property indicates the frame number corres
text: *id* .debug	int	1	(unused)
window: *id* .eye	int	1	If absent, or set to 2, the rectangle will be rendered in both stereo eyes. If set to 0 or 1
window: *id* .antialias	int	1	If 1, outline/window edge drawing will be antialiased. Default 0.
window: *id* .windowActive	int	1	If windowActive is 0, the window “matting” will not be rendered

Property	Type	Size	Description
window: *id* .outlineActive	int	1	If outlineActive is 0, the window outline will not be rendered
window: *id* .outlineWidth	float	1	Assuming antialias = 1, nominal width in image-space pixels of the outline (and the d
window: *id* .outlineBrush	string	1	Assuming antialias = 1, brush used to stroke the outline (choices are “gauss” or “solid
window: *id* .windowColor	float[4]	1	The color of the window “matting”.
window: *id* .outlineColor	float[4]	1	The color of the window outline.
window: *id* .imageAspect	float	1	The expected imageAspect of the media. If imageAspect is present and non-zero, norr
window: *id* .pixelScale	float[2]	1	X and Y scaling factors for window coordinates, IE expected source resolution. Used
window: *id* .firstFrame	int	1	If any of the window coord properties is multi-valued, this property indicates the fram
window: *id.windowULx*	float	N	Upper left window corner (x coord).
window: *id.windowULy*	float	N	Upper left window corner (y coord).
window: *id.windowLLx*	float	N	Lower left window corner (x coord).
window: *id.windowLLy*	float	N	Lower left window corner (y coord).
window: *id.windowURx*	float	N	Upper right window corner (x coord).
window: *id.windowURy*	float	N	Upper right window corner (y coord).
window: *id.windowLRx*	float	N	Lower right window corner (x coord).
window: *id.windowLRy*	float	N	Lower right window corner (y coord).

51.19 RVPaint

Paint nodes are used primarily to store per frame annotations. Below *id* is the value of nextID at the time the paint command property was created, *frame* is the frame on which the annotation will appear, *user* is the username of the user who created the property.

Property	Type	Size	Description
paint.nex	int	1	A counter used by the annotation mode to uniquely tag annotation pen strokes and text.
paint.nex	int	1	(unused)
paint.sho	int	1	If 1 display any paint strokes and text entries. If 0 do not.
paint.exc	string	N	(unused)
paint.inc	string	N	(unused)
pen: *id* : *frame* : *user* .color	float[	1	The color of the pen stroke
pen: *id* : *frame* : *user* .width	float	1	The width of the pen stroke
pen: *id* : *frame* : *user* .brush	string	1	Brush style of "gauss" or "circle" for soft or hard lines respectively
pen: *id* : *frame* : *user* .points	float[	N	Points of the stroke in the normalized coordinate system
pen: *id* : *frame* : *user* .debug	int	1	If 1 show multicolored bounding lines around the stroke.
pen: *id* : *frame* : *user* .join	int	1	The joining style of the stroke:NoJoin = 0; BevelJoin = 1; MiterJoin = 2; RoundJoin = 3;
pen: *id* : *frame* : *user* .cap	int	1	The cap style of the stroke:NoCap = 0; SquareCap = 1; RoundCap = 2;
pen: *id* : *frame* : *user* .splat	int	1	
pen: *id* : *frame* : *user* .mode	int	1	Drawing mode of the stroke (Default if missing is 0):RenderOverMode = 0; RenderEraseMode = 1;
text: *id* : *frame* : *user* .position	float[	1	Location of the text in the normalized coordinate system
text: *id* : *frame* : *user* .color	float[	1	The color of the text
text: *id* : *frame* : *user* .spacing	float	1	The spacing of the text
text: *id* : *frame*	float	1	The size of the text

51.20 RVPrimaryConvert

The primary convert node can be used to perform primary colorspace conversion with illuminant adaptation on a frame that has been linearized. The input and output colorspace primaries are specified in terms of input and output chromaticities for red, green, blue and white points. Illuminant adaptation is implemented using the Bradford transform where the input and output illuminant are specified in terms of their white points. Illuminant adaptation is optional. Default values are set for D65 Rec709.

Property	Type	Size	Description
node.active	int	1	If non-zero node is active. (default 0)
illuminantAdaptation.useBradfordTransform	int	1	If non-zero illuminant adaptation is enabled using Bradford transform. (default 1)
illuminantAdaptaton.inIlluminantWhite	float	1	Input illuminant white point. (default [0.3127 0.3290])
illuminantAdaptation.outIlluminantWhite	float	1	Output illuminant white point. (default [0.3127 0.3290])
inChromaticities.red	float[2]	1	Input chromaticities red point. (default [0.6400 0.3300])
inChromaticities.green	float[2]	1	Input chromaticities green point. (default [0.3000 0.6000])
inChromaticities.blue	float[2]	1	Input chromaticities blue point. (default [0.1500 0.0600])
inChromaticities.white	float[2]	1	Input chromaticities white point. (default [0.3127 0.3290])
outChromaticities.red	float[2]	1	Output chromaticities red point. (default [0.6400 0.3300])
outChromaticities.green	float[2]	1	Output chromaticities green point. (default [0.3000 0.6000])
outChromaticities.blue	float[2]	1	Output chromaticities blue point. (default [0.1500 0.0600])
outChromaticities.white	float[2]	1	Output chromaticities white point. (default [0.3127 0.3290])

51.21 PipelineGroup, RVDisplayPipelineGroup, RVColorPipelineGroup, RVLinearizePipelineGroup, RVLookPipelineGroup and RVViewPipelineGroup

The PipelineGroup node and the RV specific pipeline nodes are group nodes that manages a pipeline of single input nodes. There is a single property on the node which determines the structure of the pipeline. The only difference between the various pipeline node types is the default value of the property.

Property	Type	Size	Description
pipeline.nodes	string	1 or more	The type names of the nodes in the managed pipeline from input to output order.

Node Type	Default Pipeline
PipelineGroup	No Default Pipeline
RVLinearizePipelineGroup	RVLinearize
RVColorPipelineGroup	RVColor
RVLookPipelineGroup	RVLookLUT
RVViewPipelineGroup	No Default Pipeline
RVDisplayPipelineGroup	RVDisplayColor

51.22 RVRetime

Retime nodes are in many of the group nodes to handle any necessary time changes to match playback between sources and views with different native frame rates. You can also use them for "artistic retiming" of two varieties.The properties in the "warp" component (see below) implement a key-framed "speed warping" variety of retiming, where the keys describe the speed (as a multiplicative factor of the target frame rate - so 1.0 implies no difference, 0.5 implies half-speed, and 2.0 implies double-speed) at a given input frame. Or you can provide an explicit map of output frames from input frames with the properties in the "explicit" component (see below). Note that the warping will still make use of what it can of the "standard" retiming properties (in particular the output fps and the visual scale), but if you use explicit retiming, none of the standard properties will have any effect. The "precedence" of the retiming types depends on the active flags: if "explicit.active" is non-zero, the other properties will have no effect., and if there is no explicit retiming, warping will be active if "warp.active" is true. Please note that neither speed warping nor explicit mapping does any retiming of the input audio.

Property	Type	Size	Description
visual.scale	float	1	If extending the length scale is greater than 1.0. If decreasing the length scale is less than 1.0.
visual.offset	float	1	Number of frames to shift output.
audio.scale	float	1	If extending the length scale is greater than 1.0. If decreasing the length scale is less than 1.0.
audio.offset	float	1	Number of seconds to shift output.
output.fps	float	1	Output frame rate in frames per second.
warp.active	int	1	1 if warping should be active.
warp.keyFran	int	N	Input frame numbers at which target speed should change.
warp.keyRate	float	N	Target speed multipliers for each input frame number above (1.0 means no speed change).
explicit.active	int	1	1 if an explicit mapping is provided and should be used.
explicit.firstOut	int	1	The output frame range provided by the Retime node will start with this frame. The last frame provided will be determined by the length of the array in the "inputFrames" property.
explicit.inputFra	int	N	Each element in this array corresponds to an output frame, and the value of each element is the input frame number that will be used to provide the corresponding output frame.

51.23 RVRetimeGroup

The RetimeGroup is mostly just a holder for a Retime node. It has a single property.

Name	Type	Size	Description
ui.name	string	1	This is a user specified name which appears in the user interface.

51.24 RVSequence

Information about how to create a working EDL can be found in the User's Manual. All of the properties in the edl component should be the same size.

Property	Type	Size	Description
edl.frame	int	N	The global frame number which starts each cut
edl.source	int	N	The source input number of each cut
edl.in	int	N	The source relative in frame for each cut
edl.out	int	N	The source relative out frame for each cut
output.fps	float	1	Output FPS for the sequence. Input nodes may be retimed to this FPS.
output.size	int[2]	1	The virtual output size of the sequence. This may not match the input sizes.
out-put.interactiveSize	int	1	If 1 then adjust the virtual output size automatically to the window size for fram-ing.
output.autoSize	int	1	Figure out a good size automatically from the input sizes if 1. Otherwise use output.size.
mode.useCutInfo	int	1	Use cut information on the inputs to determine EDL timing.
mode.autoEDL	int	1	If non-0, automatically concatenate new sources to the existing EDL, otherwise do not modify the EDL

51.25 RVSequenceGroup

The sequence group contains a chain of nodes for each of its inputs. The input chains are connected to a single RVSe-quence node which controls timing and switching between the inputs.

Name	Type	Size	Description
ui.name	string	1	This is a user specified name which appears in the user interface.
tim-ing.retimeInputs	int	1	Retime all inputs to the output fps if 1 otherwise play back their frames one at a time at the output fps.

51.26 RVSession

The session node is a great place to store centrally located information to easily access from any other node or location. Almost like a global grab bag.

Name	Type	Size	Description
matte.aspect	float	1	Centralized setting for the aspect ratio of the matte used in all sources. Float ratio of width divided by height.
matte.centerPoi	float[2	1	Centralized setting for the center of the matte used in all sources. Value stored as X, Y in normalized coordinates.
matte.heightVis	float	1	Centralized setting for the fraction of the source height that is still visible from the matte used in all sources.
matte.opacity	float	1	Centralized setting for the opacity of the matte used in all sources. 0 == clear 1 == opaque.
matte.show	int	1	Centralized setting to turn on or off the matte used in all sources. 0 == OFF 1 == ON.

51.27 RVSoundTrack

Used to construct the audio waveform textures.

Property	Type	Size	Description
audio.volume	float	1	Global audio volume
audio.balance	float	1	[-1,1] left/right channel balance
audio.offset	float	1	Globl audio offset in seconds
audio.mute	int	1	If non-0 audio is muted

51.28 RVSourceGroup

The source group contains a single chain of nodes the leaf of which is an RVFileSource or RVImageSource. It has a single property.

Name	Type	Size	Description
ui.name	string	1	This is a user specified name which appears in the user interface.

51.29 RVSourceStereo

The source stereo nodes are used to control independent eye transformations.

Property	Type	Size	Description
stereo.swap	int	1	If non-0 swap the left and right eyes
stereo.relativeOffset	float	1	Offset distance between eyes, default = 0. Both eyes are offset.
stereo.rightOffset	float	1	Offset distance between eyes, default = 0. Only right eye is offset.
rightTransform.flip	int	1	If non-0 flip the right eye
rightTransform.flop	int	1	If non-0 flop the right eye
rightTransform.rotate	float	1	Right eye rotation in degrees
rightTrans-form.translate	float[2]	1	independent 2D translation applied only to right eye (on top of offsets)

51.30 RVStack

The stack node is part of a stack group and handles control for settings like compositing each layer as well as output playback timing.

Property	Type	Size	Description
output.fps	float	1	Output FPS for the stack. Input nodes may be retimed to this FPS.
output.size	int[2]	1	The virtual output size of the stack. This may not match the input sizes.
out-put.autoSize	int	1	Figure out a good size automatically from the input sizes if 1. Otherwise use out-put.size.
out-put.chosenAudi	string	1	Name of input which becomes the audio output of the stack. If the value is .all. then all inputs are mixed. If the value is .first. then the first input is used.
compos-ite.type	string	1	The compositing operation to perform on the inputs. Valid values are: over, add, difference, -difference, and replace
mode.useCutIn	int	1	Use cut information on the inputs to determine EDL timing.
mode.strictFran	int	1	If 1 match the timeline frames to the source frames instead of retiming to frame 1.
mode.alignStar	int	1	If 1 offset all inputs so they start at same frame as the first input.

51.31 RVStackGroup

The stack group contains a chain of nodes for each of its inputs. The input chains are connected to a single RVStack node which controls compositing of the inputs as well as basic timing offsets.

Name	Type	Size	Description
ui.name	string	1	This is a user specified name which appears in the user interface.
tim-ing.retimeInputs	int	1	Retime all inputs to the output fps if 1 otherwise play back their frames one at a time at the output fps.

51.32 RVSwitch

The switch node is part of a switch group and handles control for output playback timing.

Property	Type	Size	Description
output.fps	float	1	Output FPS for the switch. This is normally determined by the active input.
output.size	int[2]	1	The virtual output size of the stack. This is normally determined by the active input.
output.autoSize	int	1	Figure out a good size automatically from the input sizes if 1. Otherwise use output.size.
output.input	string	1	Name of the active input node.
mode.useCutInfo	int	1	Use cut information on the inputs to determine EDL timing.
mode.alignStartFram	int	1	If 1 offset all inputs so they start at same frame as the first input.

51.33 RVSwitchGroup

The switch group changes it behavior depending on which of its inputs is “active”. It contains a single Switch node to which all of its inputs are connected.

Name	Type	Size	Description
ui.name	string	1	This is a user specified name which appears in the user interface.

51.34 RVTransform2D

The 2D transform node controls the image transformations. This node is usually evaluated on the GPU.

Property	Type	Size	Description
transform.flip	int	1	non-0 means flip the image (vertically)
transform.flop	int	1	non-0 means flop the image (horizontally)
trans-form.rotate	float	1	Rotate the image in degrees about its center.
pixel.aspectRati	float	1	If non-0 set the pixel aspect ratio. Otherwise use the pixel aspect ratio reported by the incoming image.
trans-form.translate	float[2]	1	Translation in 2D in NDC space
trans-form.scale	float[2]	1	Scale in X and Y dimensions in NDC space
sten-cil.visibleBox	float	4	Four floats indicating the left, right, top, and bottom in NDC space of a stencil box.

51.35 RVViewGroup

The RVViewGroup node has no external properties.

CHAPTER

FIFTYTWO

CHAPTER 17 - ADDITIONAL GLSL NODE REFERENCE

This chapter describes the list of GLSL custom nodes that come bundled with RV. These nodes are grouped into five sections within this chapter based on the nodes "evaluationType" i.e. color, filter, transition, merge or combine. Each sub-section within a section describes a node and its parameters. For a complete description of the GLSL custom node itself, refer to the chapter on that topic i.e. "Chapter 3: Writing a Custom GLSL Node".The complete collection of GLSL custom nodes that come with each RV distribution are stored in the following two files located at:

```
 Linux & Windows:
<RV install dir>/plugins/Nodes/AdditionalNodes.gto
<RV install dir>/plugins/Support/additional_nodes/AdditionalNodes.zip

Mac:
<RV install dir>/Contents/PlugIns/Nodes/AdditionalNodes.gto
<RV install dir>/Contents/PlugIns/Support/additional_nodes/AdditionalNodes.zip
```

The file "AdditionalNodes.gto" is a GTO formatted text file that contains the definition of all the nodes described in this chapter. All of the node definitions found in this file are signed for use by all RV4 versions. The GLSL source code that implements the node's functionality is embedded within the node definition's function block as an inlined string. In addition, the default values of the node's parameters can be found within the node definition's parameter block. The accompanying support file "AdditionalNodes.zip" is a zipped up collection of individually named node ".gto" and ".glsl" files. Users can unzip this package and refer to each node's .gto/.glsl file as examples of custom written RV GLSL nodes. Note the file "AdditionalNodes.zip" is not used by RV. Instead RV only uses "AdditionalNodes.gto" which was produced from all the files found in "AdditionalNodes.zip".These nodes can be applied through the session manager to sources, sequences, stacks, layouts or other nodes. First you select a source (for example) and from the session manager "+" pull menu select "New Node by Type" and type in the name of the node in the entry box field of the "New Node by Type" window.

52.1 17.1 Color Nodes

This section describes all the GLSL nodes of evaluationType "color" found in "AdditionalNodes.gto".

52.1.1 17.1.1 Matrix3x3

This node implements a 3x3 matrix multiplication on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters:

Property	Type	Default
node.parameters.m33	float[9]	[1 0 0 0 1 0 0 0 1]

52.1.2 17.1.2 Matrix4x4

This node implements a 4x4 matrix multiplication on the RGBA channels of the inputImage. The inputImage alpha channel is affected by this node.Input parameters:

Property	Type	Default
node.parameters.m44	float[16]	[1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]

52.1.3 17.1.3 Premult

This node implements the "premultiply by alpha" operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.4 17.1.4 UnPremult

This node implements the "unpremultiply by alpha" (i.e. divide by alpha) operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.5 17.1.5 Gamma

This node implements the gamma (i.e. pixelColor^gamma) operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node. Input parameters:

Property	Type	Default
node.parameters.gamma	float[3]	[0.4545 0.4545 0.4545]

52.1.6 17.1.6 CDL

This node implements the Color Description List operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Parameter lumaCoefficients defaults to full range Rec709 luma values.Input parameters:

Property	Type	Default
node.parameters.slope	float[3]	[1 1 1]
node.parameters.offset	float[3]	[0 0 0]
node.parameters.power	float[3]	[1 1 1]
node.parameters.saturation	float	[1]
node.parameters.lumaCoefficients	float[3]	[0.2126 0.7152 0.0722]
node.parameters.minClamp	float	[0]
node.parameters.maxClamp	float	[1]

52.1.7 17.1.7 CDLForACESLinear

This node implements the Color Description List operation in ACES linear colorspace on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Parameter lumaCoefficients defaults to full range Rec709 luma values.If the inputImage colorspace is NOT in ACES linear, but in some X linear colorspace; then one must set the 'toACES' property to the X-to-ACES colorspace conversion matrix and similarly the 'fromACES' property to the ACES-to-X colorspace conversion matrix.Input parameters:

Property	Type	Default
node.parameters.slope	float[3]	[1 1 1]
node.parameters.offset	float[3]	[0 0 0]
node.parameters.power	float[3]	[1 1 1]
node.parameters.saturation	float	[1]
node.parameters.lumaCoefficients	float[3]	[0.2126 0.7152 0.0722]
node.parameters.toACES	float[16]	[1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]
node.parameters.fromACES	float[16]	[1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]
node.parameters.minClamp	float	[0]
node.parameters.maxClamp	float	[1]

52.1.8 17.1.8 CDLForACESLog

This node implements the Color Description List operation in ACES Log colorspace on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Parameter lumaCoefficients defaults to full range Rec709 luma values.If the inputImage colorspace is NOT in ACES linear, but in some X linear colorspace; then one must set the 'toACES' property to the X-to-ACES colorspace conversion matrix and similarly the 'fromACES' property to the ACES-to-X colorspace conversion matrix.Input parameters:

Property	Type	Default
node.parameters.slope	float[3]	[1 1 1]
node.parameters.offset	float[3]	[0 0 0]
node.parameters.power	float[3]	[1 1 1]
node.parameters.saturation	float	[1]
node.parameters.lumaCoefficients	float[3]	[0.2126 0.7152 0.0722]
node.parameters.toACES	float[16]	[1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]
node.parameters.fromACES	float[16]	[1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1]
node.parameters.minClamp	float	[0]
node.parameters.maxClamp	float	[1]

52.1.9 17.1.9 SRGBToLinear

This linearizing node implements the sRGB to linear transfer function operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.10 17.1.10 LinearToSRGB

This node implements the linear to sRGB transfer function operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.11 17.1.11 Rec709ToLinear

This linearizing node implements the Rec709 to linear transfer function operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.12 17.1.12 LinearToRec709

This node implements the linear to Rec709 transfer function operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.13 17.1.13 CineonLogToLinear

This linearizing node implements the Cineon Log to linear transfer function operation on the RGB channels of the inputImage. The implementation is based on Kodak specification "The Cineon Digital Film System". The inputImage alpha channel is not affected by this node.Input parameters: (values must be specified within the range [0..1023])

Property	Type	Default
node.parameters.refBlack	float	95
node.parameters.refWhite	float	685
node.parameters.softClip	float	0

52.1.14 17.1.14 LinearToCineonLog

This node implements the linear to Cineon Log film transfer function operation on the RGB channels of the inputImage. The implementation is based on Kodak specification "The Cineon Digital Film System". The inputImage alpha channel is not affected by this node.Input parameters: (values must be specified within the range [0..1023])

Property	Type	Default
node.parameters.refBlack	float	95
node.parameters.refWhite	float	685

52.1.15 17.1.15 ViperLogToLinear

This linearizing node implements the Viper Log to linear transfer function operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.16 17.1.16 LinearToViperLog

This node implements the linear to Viper Log transfer function operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.17 17.1.17 RGBToYCbCr601

This node implements the RGB to YCbCr 601 conversion operation on the RGB channels of the inputImage. Implementation is based on ITU-R BT.601 specification. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.18 17.1.18 RGBToYCbCr709

This node implements the RGB to YCbCr 709 conversion operation on the RGB channels of the inputImage. Implementation is based on ITU-R BT.709 specification. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.19 17.1.19 RGBToYCgCo

This node implements the RGB to YCgCo conversion operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.20 17.1.20 YCbCr601ToRGB

This node implements the YCbCr 601 to RGB conversion operation on the RGB channels of the inputImage. Implementation is based on ITU-R BT.601 specification. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.21 17.1.21 YCbCr709ToRGB

This node implements the YCbCr 709 to RGB conversion operation on the RGB channels of the inputImage. Implementation is based on ITU-R BT.709 specification. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.22 17.1.22 YCgCoToRGB

This node implements the YCgCo to RGB conversion operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.23 17.1.23 YCbCr601FRToRGB

This node implements the YCbCr 601 "Full Range" to RGB conversion operation on the RGB channels of the inputImage. Implementation is based on ITU-R BT.601 specification. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.24 17.1.24 RGBToYCbCr601FR

This node implements the RGB to YCbCr 601 "Full Range" conversion operation on the RGB channels of the inputImage. Implementation is based on ITU-R BT.601 specification. The inputImage alpha channel is not affected by this node.Input parameters: None

52.1.25 17.1.27 Saturation

This node implements the saturation operation on the RGB channels of the inputImage. The inputImage alpha channel is not affected by this node.Parameter lumaCoefficients defaults to full range Rec709 luma values.Input parameters:

Property	Type	Default
node.parameters.saturation	float	[1]
node.parameters.lumaCoefficients	float[3]	[0.2126 0.7152 0.0722]
node.parameters.minClamp	float	[0]
node.parameters.maxClamp	float	[1]

52.2 17.2 Transition Nodes

This section describes all the GLSL nodes of evaluationType "transition" found in "AdditionalNodes.gto".

52.3 17.2.1 CrossDissolve

This node implements a simple cross dissolve transition effect on the RGBA channels of two inputImage sources beginning from startFrame until (startFrame + numFrames -1). The inputImage alpha channel is affected by this node.Input parameters:

Property	Type	Default
node.parameters.startFrame	float	40
node.parameters.numFrame	float	20

52.4 17.2.2 Wipe

This node implements a simple wipe transition effect on the RGBA channels of two inputImage sources beginning from startFrame until (startFrame + numFrames -1). The inputImage alpha channel is affected by this node.Input parameters:

Property	Type	Default
node.parameters.startFrame	float	40
node.parameters.numFrame	float	20

CHAPTER

FIFTYTHREE

GTO: THE KITCHEN SINK OF DATA

File Format, Protocols, and Utilities.

53.1 Overview

Historically, GTO format's primary usage is storage of static geometric data (cached geometry). As such, the types of data you might find in a GTO file are things like polygonal meshes, various types of subdivision surfaces, NURBS or UBS surfaces, coordinate systems, hierarchies of objects, material bindings, and even images.

From a historic point of view, the GTO file format is most closely related to the original inventor file format, the Stanford PLY format and the PDB particle format. Like the Wavefont PDB file format, there are a limited number of simple GTO data types (float, int, string, boolean). Like the inventor file format, a GTO can hold an entire transformation hierarchy including geometric leaf nodes. Like the PLY format, the GTO format can contain an arbitrary amount of data per primitive. Most importantly however, the GTO file format is intended to be very OBJ-like; its relatively easy to read and write and easy to ignore data you don't want or know about.

GTO files can be either binary or text files. Binary files are the preferred format for large data sets. The GTO text format is intended to be human readable/editable; the syntax is simple and concise. The text format is useful when storing "bag of parameters" files and similar data.

The binary file is either big or little endian on disk, but should be readable on any platform.

The GTO reader can be use libz to read and write compressed files natively. We find that compressed GTO files created by most 3D programs are approximately 60% leaner than uncompressed files.

GTO files conceptually contain objects which are optionally composed of nested namespaces called components. Components are further composed of properties. A property contains an array of one of the predefined data types with up to a four dimensional "shape". For example, you might have an object which looks something like this:

```
Object “cube”
   Component “points”
     Property float [3][8] “position” Property float[1][8] “mass”
     Property byte[1][8] “type” Property short[1][8] “size”
   Component “indices”
     Property int[1][32] “vertex”
```

Using the terminology above, the object "cube" contains five properties: **position** , **mass** , **type** , **size** , and **vertex** . The **points** component describes the points that make up the cube vertices. Each point has a position and mass stored in properties of the same name. The position property data is composed of eight float [3] data items (or 8 3D points). The **mass** property is composed of a 8 scalar floating point values (one for each point).

The **elements** component contains two properties. **type** indicates the type of the element (for example, triangle, quad, or triangle strip). In this case the elements might all be quads. **size** indicates the number of **vertices** in each of the eight faces (elements) of the cube—(4 for a cube). The vertex property of the **indices** component contains the actual indices: 4 per face for a total of 32.

Of course you could store much more data with the cube object if you wanted to. For example, if you wanted velocity or color per point, this would be another property in the points component.

The meaning of this data is another story altogether. Its all handled by protocol. One application may store things in the GTO file that another application has no method of interpreting even though it can read that data and modify it. In the example above, you need to know to expect that polygonal data is stored in the given properties. The same data could be stored with different property names and a more complex layout. (The "polygon" protocol described later in this document is different and more involved than the above example.)

GTO was (and still is) used by a number of film post production facilities for geometry caching. 3D scenes are evaluated and the final geometry is written into GTO files which are later consumed by a renderer (e.g. RenderMan).

A newer geometry caching format called Alembic was introduced by ILM in 2010 which has a similar purpose and has taken over that role.

53.2 New in Version 4

Version 4 adds two new features: nested components and property types with up to four dimensions.

Version 3 files always had the same structure of objects.component.property. Nested components allows any number of component names between the object and property:

```
object.component1.component2.component3.property
```

In text GTO files this looks like this:

```
 object
    {
       component1
       {
         component2
         {
           component3
           {
            property ...
            }
         }
       }
    }
```

Where version 3 allowed only a single "width" for properties, version 4 allows up to four dimensions:

```
 float[4,10,20,30][3] = [ ... ]
```

The above declares 3 4x10x20x30 float data object.

53.3 Overview

53.4 Binary Format

The GTO file has six major sections which appear in the following order.

1. **Header** (Gto::Header). The header structure contains the GTO magic number (used to determine endianness), the version of the GTO specification that the file was written as, and the number of top level objects in the file. There is one instance of a header in the file. Finally, the header indicates how many strings are in the string table.

```
 Magic = 0x0000029f; Cigam = 0x9f020000; // means the file is opposite
endianess

struct Header

{
  uint32 magic;
  uint32 numStrings;
  uint32 numObjects;
  uint32 version;
  uint32 flags; // reserved;
};
```

2. **String Table** . After the header, null terminated strings are written in the file. The order of these strings is important. All names and string properties store indices into the string table instead of actual strings. In order to read the file properly, the string table must be available until the file is completely read. (Unless you don't care about any strings!)

   The index number refers the string number in the table not its byte offset. So the string index 9 (for example) refers to the 10th string in the table (string index 0 is the first string in the table).

3. **ObjectHeader** (Gto::ObjectHeader). The object header indicates what kind of protocol to use to interpret it, the **object** name and the number of components. (More on the object protocol later). The name—like all strings in the GTO file—is stored as a string table entry. If the file header indicated N objects in the file, there will be N ObjectHeaders.

```
struct ObjectHeader
{
uint32 name; // a string table index
uint32 protocolName; // a string table index
uint32 protocolVersion;
uint32 numComponents;
uint32 pad; // unused
};
```

4. **ComponentHeader** (Gto::ComponentHeader). Like the ObjectHeaders the ComponentHeaders will appear together for all objects in order. The component header indicates the number of properties in the component and the name of the component.

```
enum ComponentFlags
{
Transposed = 1 << 0,
Matrix = 1 << 1,
};
```

(continues on next page)

(continued from previous page)

```
struct ComponentHeader
{
uint32 name; // a string table index
uint32 numProperties;
uint32 flags;
uint32 interpretation; // a string table index
uint32 childLevel; // nesting level
};
```

5. **PropertyHeader** (Gto::PropertyHeader). The PropertyHeaders, like the object and component headers, appear en masse in the file. The PropertyHeader contains the name, size, type, and dimension of the property.

```
enum DataType
{
Int, // int32
Float, // float32
Double, // float64
Half, // float16
String, // string table indices
Boolean, // bit
Short, // uint16
Byte // uint8
};
struct Dimensions
{
uint32 x;
uint32 y;
uint32 z;
uint32 w;
}
struct PropertyHeader
{
uint32 name; // string table index
uint64 size;
uint32 type; // DataType enum value
Dimensions dims;
uint32 interpretation; // string table index
};
```

6. **Data** . The last section of the file contains all of the property data. The beginning and end of a properties data are not marked. The size must be consistent with the description of the property used in the PropertyHeader.

In (Text) diagram form the file looks something like this:

File Header
String Table
Object Header …
Component Header …
Property Header …
Property Data …

53.5 Text Format

GTO has a text representation in addition to the binary representation. The text representation is designed for human use; it is intended to be easy to modify or create from scratch in a text editor. It is not intended to compete with XML formats (which are typically only human readable in theory) nor is it intended to be used in place of the binary format which is much faster and more economical for storage of large data sets.

53.5.1 Example of a Cube Stored as a Text GTO

Here's the example from the overview section: a cube stored using the "polygon" protocol:

```
GTOa (4)
# this is a comment
cube : polygon (2)
{
points
{
float[3] position = [ [ -2.5 2.5 2.5 ]
[ -2.5 -2.5 2.5 ]
[ 2.5 -2.5 2.5 ]
[ 2.5 2.5 2.5 ]
[ -2.5 2.5 -2.5 ]
[ -2.5 -2.5 -2.5 ]
[ 2.5 -2.5 -2.5 ]
[ 2.5 2.5 -2.5 ] ]
float mass = [ 1 1 1 1 1 1 1 1 ]
}
elements
{
byte type = [ 2 2 2 2 2 2 ]
short size = [ 4 4 4 4 4 4 ]
}
indices
{
int vertex = [ 0 1 2 3
7 6 5 4
3 2 6 7
4 0 3 7
4 5 1 0
1 5 6 2 ]
}
}
```

The first line of the file is an identifier to tell the parser what variety of GTO file it is: in this case GTOa which indicates a plain ASCII text file. Currently the parser can only handle ASCII encoding; a forthcoming version will allow UTF-8.

Objects are declared using the syntax:

```
OBJECTNAME [ : PROTOCOL [ (PROTOCOL_VERSION) ] ]
{
... object contents ...
}
```

The brackets enclose optional syntax. So the PROTOCOL_VERSION (including the parents) is optional. The PROTOCOL is also optional; if omitted (along with the colon) the protocol defaults to object. In the example, "cube" is the name of the object and "polygon" is the name of the protocol–the protocol version is 2.

Components must be declared inside the object brackets or other components. The brackets denote a *namespace* which is either an object namespace or a component namespace. Component namespaces must always be declared inside of an object or component namespace. Object namespaces can only appear at the top level of the file; in other words, objects cannot be inside another namespace.

Components are declared like this:

```
COMPONENTNAME [as INTERPRETATION]
{
  ... component contents ...
}
```

The INTERPRETATION can be any string. Properties can be declared inside of the component namespace optionally followed by nested component declarations. The property declaration is the most flexible; since some aspects of the property (like its size) can be determined by the parser from the property data, you can omit them.

The property syntax in its most general form is:

```
TYPE[XS,YS,ZS,WS][SIZE] PROPERTYNAME as INTERPRETATION = values ...
```

The brackets around XS,YS,ZS,WS and SIZE are literal in this case; they actually appear in the file. As you can see from the example, some of the property declaration syntax is optional. The SIZE can usually be determined from the values so it may be omitted. The dimensions are assumed to be 1 (or scalar) if it is omitted. The as INTERPRETATION section of the declaration may also be omitted.

What cannot be omitted is the TYPE, PROPERTYNAME, and the assignment of values.

53.5.2 How Strings are Handled in the Text Format

With the exception of keywords and type names, any string in the text GTO file can be either be quoted or non-quoted. Non-quoted strings are restricted to strings which do not represent numbers. In addition, if a string contains punctuation or whitespace, it must be quoted. For example, if the name of the object in the cube example was "four dimensional time-cube" it would have to be declared like this:

```
"four dimensional time-cube" : polygon
{
...
}
```

There is one additional exception: if a string is also a keyword or type name, it must be quoted. For example, here's an exceptional property declaration:

```
int "int" as "as" = 1
```

In this case the quoted string "int" is being used as the property name, but because it is also the name of a GTO type, it must be quoted. The string "as" is being used as an interpretation string and must be quoted because "as" is also a keyword in the the GTO file.

When in doubt quote.

53.5.3 Value Brackets

Generally, a property value and elements of the value are enclosed in brackets:

```
TYPE[DIMENSIONS] PROPERTYNAME = [ [a b ...] [d e ...] ... ]
```

In this documentation, the *value* of a property is everything to the right of the "=" and an *element* is a fixed size collection of numbers or strings. The *size* of a property is the number of elements in its value. So in the example above, the `[a b ...]` portion of the syntax is an *element* .

Bracketing the property value is optional in one circumstance: when the number of elements in the property value is one. For example, these declarations are equivalent:

```
int foo = 1
int foo = [1]
```

If the width of the type is not one (elements are not scalar), then brackets must be put around each element of the property. If the size is one but the width is not one, then the enclosing brackets are still optional:

```
int[2] foo = [1 2]
int[2] foo = [ [1 2] ]
```

If however the size of the property is greater than one, the enclosing value brackets are required:

```
# property of size 3
int[2] foo = [ [1 2] [3 4] [5 6] ]
```

To declare a property with no value use empty brackets:

```
int foo = []
```

53.5.4 The Size of a Property

The size of a property can be declared as part of its type declaration:

```
int[1][4] foo = [1 2 3 4]
```

In this case, "foo" contains four scalar elements. Because the size was specified, the following would be a syntax error:

```
int[1][4] foo = [1 2 3 4 5]
```

The parser would complain because five elements were supplied even though the property was declared as having only four. If no size is specified than the parser will determine the size from the number of elements in the value:

```
int[1] foo = [1 2 3 4 5]
```

So in this case "foo" has five elements. Note that in order to declare the size specifically, you must also declare the element width—even if the width is one. In the last example, because we did not specify the size, the declaration could also have been:

```
int foo = [1 2 3 4 5]
```

In this case it is understood that the type is actually `int[1][5]`.

Additional dimensions can be added to make e.g. a matrix:

```
float[4,4] M = [1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1]
```

This can be extended up to four dimensions:

```
byte[4,1920,1080] eightBit1080pImage = [ ... ]
float[3,32,32,32] floatingPoint3DLUT = [ ... ]
```

53.5.5 Run Length Encoding of Values

In some cases, a value will contain many copies of an element. There is a special syntax for these cases; you can use an ellipsis to indicate that all remaining elements are identical. The ellipsis can only appear directly before the final bracket character.

There is one restriction when using this syntax: the type of the property value must be completely specified (including the size of the property) and the value must be enclosed in brackets. For example:

```
int[1][100] mass = [1 ...]
```

The ellipsis is literal (its actually in the file as three dot characters) The property "mass" will be one for all 100 elements. If the element has a width greater than one:

```
float[3][100] velocity = [ [0 0 0] ... ]
```

The ellipsis is used in place of an element. The following will *not* work:

```
float[3][100] velocity = [ [0 ...] ... ]
```

The intention here is to make all of the velocity elements `[0 0 0]`. However, this syntax is not correct and will produce a parsing error.

53.5.6 Syntax Reference

The grammar for the text GTO file. *INT* is an integer constant. *FLOAT* is a floating point constant, with a possible exponent part. *STRING* is either a quoted or non-quoted string. All other values are literal. Double quoted strings are keywords.

```
file::
     "GTOa" object_list
     "GTOa" ( INT ) object_list

object_list::
     object
     object_list object

object::
     STRING { component_list_opt }
     STRING : STRING { component_list_opt }
     STRING : STRING ( INT ) { component_list_opt }
```

(continues on next page)

(continued from previous page)

```
component_list_opt::
     nothing
     component_list

component_list::
     component
     component_list component

component_block::
     nothing
     property_list
     component_list
     property_list component_list

interp_string_opt::
     nothing
     "as" STRING

component::
     STRING interp_string_opt { component_block }

property_list::
     property
     property_list property

property::
     type STRING interp_string_opt = atomic_value
     type STRING interp_string_opt = [ complex_element_list ]

dimensions::
     INT
     INT "," INT
     INT "," INT "," INT
     INT "," INT "," INT "," INT

type::
     basic_type
     basic_type [ dimensions ]
     basic_type [ dimensions ] [ INT ]

basic_type::
     "float" "int" "string" "short" "byte" "half"
     "bool" "double"

complex_element_list::
     nothing
     element_list
     element_list "..."

element_list::
     element
     element_list element
```

(continues on next page)

(continued from previous page)

```
element::
     atomic_value
     [ atomic_value_list ]

atomic_value_list::
     string_value_list
     numeric_value_list

atomic_value::
     string_value
     numeric_value

string_value_list::
     string_value
     string_value_list string_value

string_value::
     STRING

numeric_value_list::
     numeric_value
     numeric_value_list numeric_value

numeric_value::
     float_num
     int_num

float_num::
     FLOAT
     - FLOAT

int_num::
     INT
     - INT
```

53.6 Types of Property Data

The GTO format pre-defines a small number of data types that can be stored as properties. The currently defined types are:

`doub`	64 bit IEEE floating point.	[Property Type]
`floa`	32 bit IEEE floating point.	[Property Type]
`half`	16 bit IEEE floating point	[Property Type]
`int`	32 bit signed integer.	[Property Type]
`int6`	64 bit signed integer.	[Property Type]
`shor`	16 bit unsigned integer.	[Property Type]
`byte`	8 bit unsigned integer (char).	[Property Type]
`bool`	Bit or bit vector. Not currently implemented.	[Property Type]
`stri`	The string type is stored as a 32 bit integer index into the GTO file's string table. So storing a lot of strings (especially if there is a lot of redundancy) is reasonably cheap. All strings in the GTO file are stored in this manner.	[Property Type]

Each of these data types can be made into a vector of that type. For example the float data type can be made into a point `float[3]` or a matrix `float[16]` . To store a scalar element the size of the vector is 1. (e.g. `float[1]`).

In this document, the types are all specified as 2 dimensional arrays ala the C programming language. Here is a complete list of example type forms:

- `float[3]` — the float triple type.
- `float[1] [1]` - a single floating point number.
- `float[3] []` - any number of float triples.
- `float[3] [3]` - three float triples.
- `float[16] []` - any number of a 16 float element.
- `float[4,4] [1]` - a 4x4 float matrix.
- `float[3,3] []` - any number of 3x3 float matrices.
- `float[4,512,128] [7]` - seven four component 512x128 images.
- `float[3,32,32,32] [1]` - a 3 component 32x32x32 volume.

53.7 Interpretation Strings

Each property can have an additional string stored with it called the "interpretation". The intent is to allow applications to provide specific information about the property. For example, a property of type `float[4]` can be interpreted as a homogeneous 3D coordinate, a quaternion, or an RGBA value. The interpretation field can be used to distinguish between them.

Why not just make new primitive GTO types for these? The format's only purpose is storage of data. By decoupling the interpretation of the data from its storage, each application is allowed to make its own policy while maintaining flexibility for simpler applications.

Here's a simple example of `gtoinfo` output of a file with an image object in it created with `gtoimage` :

```
object "image" protocol "image" v1
    component "image"
      property string[1][1] "originalFile" interpret as "filename"
      property string[1][1] "originalEncoding" interpret as "filetype"
      property string[1][1] "type"
      property int[1][2] "size"
      property float[3][199168] "pixels" interpret as "RGB"
```

Some of the stings will be application specific. Programs that generically edit GTO files should attempt to preserve the interpretation strings.

It is not an error to define the interpretation for a property as the empty string—in other words, unspecified.

The following strings are not currently part of the format specification but are used by the sample implementation. In a future release we may make these "official". It's ok to have multiple space separated strings in the interpretation strings (e.g. "4x4 row-major").

coord	The data can be of any width or type. For width N the data represents a point in N dimensional space.	[Interpretation String]
norma	The data can be of any width or type. For width N the data represents a unit vector prependicular to an N dimensional surface or in the case of N == 2, a curve.	[Interpretation String]
4x4	The width of the property data should be 16. The data is intended to be interpreted as a 4x4 matrix. For example, the **object.globalMatrix** property of the **Coordinate System** protocol would be a “4x4” property.	[Interpretation String]
3x3	The width of the property data should be 9. The data is intended to be interpreted as a 3x3 matrix.	[Interpretation String]
row-m	Indicates that matrix data is in row major ordering.	[Interpretation String]
colum	Indicates that matrix data is in column major ordering.	[Interpretation String]
quate	The width of the property should be four. The data should be interpreted as a quaternion. Presumably the type of a quaternion property would be `float[4]` or `double[4]` since these are the only types that make sense. The first element of the data is the real part followed by the “i”, “j”, and “k” imaginary components.	[Interpretation String]
compl	The width of the property should be two. The data is interpreted as a complex number with the first element being the real part and the second element the imaginary part.	[Interpretation String]
indic	The data type should be an integral type. The property contains indices.	[Interpretation String]
bbox	The data type should have an even width. The property contains bounding boxes.	[Interpretation String]
homog	The width of the property should be two or more. If the width is three, then the data is a two dimensional homogeneous coordinate. If the width is four, then the data is a three dimensional homogeneous coordinate. So for data of width `N` the data represents a homogeneous coordinate in `N-1` dimensions.	[Interpretation String]
RGB	The width of the property should be three. The data represents a color with red, green, and blue components.	[Interpretation String]
BGR	The width of the property should be three. The data represents a color with blue, green, and red components. (Reversed `RGB`)	[Interpretation String]
RGBA	The width of the property should be four. The data represents a color (or pixel) with red, green, blue, and alpha components.	[Interpretation String]
ABGR	The width of the property should be four. The data represents a color (or pixel) with alpha, blue,	[Inter-

53.8 Object Protocols

The Object data interpretation is not defined by the GTO format. However, there are currently some protocols in use that are well defined and these are documented here. Caveat emptor: gto files in the wild may contain more data than these protocols define, but they presumably will obey the protocol if they indicate it by name. It's also possible that some objects may obey more than one protocol yet only indicate that they follow one. Unfortunately, some protocols also specify optional components and properties in case all of this was not confusing enough.

Protocols also have a version number. The version number is an integer; there are no sub-versions. If there are significant changes to a protocol, the version number should be bumped. The version number is not meant as a method of making alternate protocols with the same name. We have had to make three modifications to the protocols since the file format was invented; one to the polygon protocol and one to the transform protocol, and the introduction of a new protocol (connections). The changes are documented in those sections.

In this document, properties are all named "comp.prop", where "comp" is the name of the component the property belongs to and "prop" is the name of the property. This is done to prevent ambiguity when two different properties in different components but with the same name exist. In the GTO file and when using the reader library only the property name will appear.

There are two kinds of protocols: major and minor. Every object must have a major protocol that's stored in the ObjectHeader—this is the main indicator of how to interpret the object data. In addition, the object may also have several minor protocols. These indicate optional data and how to interpret it. The next section describes how these are stored in the file.

53.8.1 Object Protocol

The name of the protocol as it appears in the ObjectHeader is "object" version 1. The protocol does not require any other protocols. Here it is:

`object`	A container for properties which don't fit into other component catagories well. A catch-all data "per-object" component.	[Required Component]
`float[16` `object.` `globalMa`	The global world-space transform for the object.	[Optional Property]
`float[6]` `object.` `bounding`	The global world-space bounding box for the object.	[Optional Property]
`string[1` `object.` `parent`	Name of this object's parent in a scene heirarchy.	[Optional Property]
`string[1` `object.` `name`	The name of the object. This name should be identical to the name in the ObjectHeader.	[Optional Property]
`string[1` `object.` `protocol`	Additional protocols. This property may contain the main protocol name and any other minor protocols that the object adheres to. If a protocol name appears in this property, the object must adhere to that protocol. Its not an error for a program to output this property with only the major protocol as its value; this is of course redundant since the protocol name is required by the ObjectHeader. It is also not an error for this property to exist but contain nothing.	[Optional Property]
`int[1][]` `object.` `protocol`	Additional protocol version numbers. This property may exist if the **object.protocol** property exists. Each entry in this property corresponds to the same entry indexed in the **object.protocol** property. This property must contain the same number of elements that the **object.protocol** property does.	[Optional Property]

You may be asking why the **object** protocol exists at all. The name of an object is stored in the ObjectHeader in the file and in the C++ library is passed to the reader code. The "name" property is redundant right? Well yes. But some programs will output the name both in the ObjectHeader and in an **object** component as the property "name".

The main point of this protocol is to define the **object** component. This component is meant to hold data that is "per object" and which doesn't really fit neatly into other components. The name is one such case. The coordinate system protocol also defines properties in the **object** component and the minor protocols are optionally stored here.

53.8.2 Coordinate System Protocol

The name of the protocol as it appears in the ObjectHeader is "transform" version 3 *1* . The protocol requires the object protocol. Objects which obey the transform protocol will have global matrices and possibly a parent.

object	From the "object" protocol.	[Required Component]
float[16][1 object. globalMatri	A 4x4 matrix of floating point numbers. This matrix describes the world matrix of the coordinate system.	[Required Property]
string[1][1 object. parent	The name of an object to which this coordinate system is parented. Presumably this object (if it appears in the gto file) will also obey the **transform** protocol. If this property does not exist or the name is "" (the empty string) then the coordinate system presumably is a root coordinate system. *2*	[Optional Property]

53.8.3 Particle Protocol

The name of the protocol as it appears in the ObjectHeader is "particle" version 1. The protocol may include the object and transform protocols.

points	The points component is transposable. That means that all of its properties are required to have the same number of elements.	[Required Component]
float[3][points. position		[Optional Property]
float[4][points. position	The position property is intended to hold the position of the particle in its owncoordinate system or world space if it has no coordinate system. The element is either a 3D or 4D (homogeneous) point	[Optional Property]
float[3][points. velocity	The velocity property – if it exists – should hold the velocity vector per-point in the same coordinate system that the "position" property is in.	[Optional Property]
int[1][] points. id	The "id" property should it exist will always be defined as an integer per particle (or other integral type if it ever changes). This number should be unique for each particle. Ideally, multiple GTO files with a point that has the same "id" property for a given particle animation should be the same particle.	[Optional Property]

The **particle** protocol defines the **points** component that many other protocols are derived from. For example, the **NURBS** protocol uses the points defined by the particle protocol as control vertices. There can be any number of properties associated with particles including string per-particle.

The **points** component is marked transposable in the its ComponentHeader. This means that the properties in the component are guaranteed to have the same number of elements. Because of this, the data for the properties in a transposable component may be stored differently than other components. For example, the normal state of affairs is to write data like this:

```
position0 position1 position2 .... positionN
velocity0 velocity1 velocity2 .... velocityN
mass0 mass1 mass2 .... massN
```

So that you must read through all of the particle positions before you can read the first particle's velocity. But this is not usually the best way to read particle data for rendering. You may want to cull the particles as you read them without storing the data. In order to do this the data needs to be laid out like this:

```
position0 velocity0 mass0
position1 velocity1 mass1
position2 velocity2 mass2
...
```

In this case, each particle is scanned in one chunk allowing for optimizations. Obviously this complicates reading, but in the case of giga-particle renderers, this can be a huge memory savings.

53.8.4 Strand Protocol

A **strand** object contains a collection of curves. This is somewhat analogous to an object of protocol **particle** as described above.

`points`		[Required Component]
`float[3][] points.position`	The CVs which make up each curve. The number of CVs per curve can vary by curve type and size.	[Required Component]
`strand`	Information that is relevant to the *all* strands in the object.	[Required Component]
`string[1][] strand.type`	String describing curve type. Currently, supported values are linear for degree 1 curves, or cubic for degree 3 curves.	[Required Property]
`float[1][1] strand.width`	If each end of all curves is the same width, you can just specify that one number instead of the list as with **elements.width** below.	[Optional Property]
`elements`	Information that applies to each separate strand in the object.	[Required Component]
`int[1][] elements.size`	This is a list of the sizes of each curve in this object. For example, if there are two curves in this object, with 4 CVs and 3 CVs respectively, then: `elements.size = [ 4 3 ]`	[Required Property]
`float[2][] elements.width`	This is a list of the widths of each end of each curve. The width for each curve will be linearly interpolated over the length.	[Optional Property]

53.8.5 NURBS Protocol

The name of the protocol as it appears in the ObjectHeader is “NURBS” version 1. The protocol requires the **particle** protocol and optionally includes the **object** and **transform** protocols.

`points`	see **particle** protocol. The points describe data per NURBS control vertex.	[Required Component]
`float[3` `points.` `positio`		[Required Property]
`float[4` `points.` `positio`	The position property holds the control point positions in its own coordinate system or world space if it has no coordinate system. The element is either a 3D or 4D (homogeneous) point. If the type is float[4] the fourth component of the element will be the rational component of the control point position. The control points are laid out in *v*- **major** order (*u* iterates more quickly than *v*).	[Required Property]
`float[1` `points.` `weight`	If the position property is of type `float[3][]` there may optionally be a “weight” property. This property holds the homogeneous (rational) component of the position. Older GTO writers may export data in this manner. The preferred method is to use a `float[4]` element position.	[Optional Property]
`surface`	Properties related to the definition of a NURBS surface are stored in this component.	[Required Component]
`float[1` `surface.` `degree`	The degree of the surface in *u* and *v* .	[Required Property]
`float[1` `surface.` `uKnots`		[Required Property]
`float[1` `surface.` `vKnots`	The NURBS surface knot vectors in *u* and *v* are stored in these properties. The knots are not piled. The usual NURBS restrictions on how numbers may be stored in the knot vectors apply.	[Required Property]
`float[1` `surface.` `uRange`		[Required Property]
`float[1` `surface.` `vRange`	The range of the knot parameters in *u* and *v* .	[Required Property]

The **NURBS** protocol currently does not handle trim curves, points on surface, etc. Ultimately, the intent is to handle the trim curves and other nasties as NURBS curveson-surface which will be stored in additional components. UBS surfaces can be stored as NURBS with non-rational uniform knots.

53.8.6 Polygon Protocol

The name of the protocol as it appears in the ObjectHeader is "polygon" version 2 *3* . The protocol requires the **particle** protocol and optionally includes the **object** and **transform** protocols.

There are a number of alternative configurations of this protocol depending on the value of the **smoothing.method** property. All of these involve the placement of normals in the file.

poin	See **particle** protocol. The points describe data per vertex.	[Required Component]
floa poin posi	The positions for regular polygonal meshes are stored as `float[3]` .	[Required Property]
floa poin norm	Normals per vertex. The **smoothing.method** property will have the value of *Smooth* if this property exists. Note that use of the *Smooth* smoothing method does not require that this property exists. If it does not the method is merely and indication of how the normals should be constructed.	[Optional Property]
norm	This property is required only if the **normals** component exists and the value of **smoothing.method** is *Partitioned* or *Discontinuous* .	[Required Property]
elem	The elements component is transposable. All properties in the elements component must have the same number of elements. Each element corresponds to a polygonal primitive.	[Required Component]
byte elem type	Elements are modeled after the OpenGL primitives of the same name. The vertex order is identical to that defined by GL. The type numbers outside those given here are not defined but reserved for future use. So far, these are the define type numbers: **0 –** Polygon General N-sided polygon. This can be used for any polygon that has 3 or more vertices., **1 –** Triangle A three vertex polygon., **2 –** Quad A four vertex polygon., **3 –** TStrip Triangle strip., **4 –** QStrip Quad strip., **5 –** Fan Triangle fan.	[Required Property]
shor elem size	The size of each primitive. Because the type is short, there is a limit of 65k vertices per primitive.	[Required Property]
shor elem smoo	This property may exist if the value of **smoothing.method** is *Partitioned* . In that case, this property indicates the smoothing group number associated with each element. These can be used to recompute the normals. These numbers are the same as those found in the Wavefront .obj file format's "s" statements. A value of 0 indicates that an element is not in a smoothing group.	[Optional Property]
floa elem norm	Normals per element. The **smoothing.method** property will have the value of *Faceted* if this property exists. Note: the use of *Faceted* smoothing method does not require that this property exists. If it does not, the smoothing method is merely and indication of how the normals should be created.	[Optional Property]
indi	The indices component is transposable. All of its properties are required to have the same number of elements. Each entry in the indices component corresponds to a polygonal vertex. *4*	[Required Component]
int[indi vert	A list of all the polygonal vertex indices in the same order as the **elements.primtives** . The indices refer to the **points.position** property. So if the first polygonal element is a triangle and second is a general four vertex polygon then vertex indices will be something like: `0 1 2 1 0 3 4 ...` which would be grouped as: `(0 1 2) (1 0 3 4) ...` The first group is the triangle and the second the polygon.	[Required Property]
int[indi st	Similar to the vertex indices but indicates indices into *st* coordinates. These are usually stored in the "mappings" component but may also appear in the **points** component.	[Optional Property]
int[indi norm	Indices into stored normals if there are any. The **smoothing.method** property will have the value of *Partitioned* or *Discontinuous* if this property exists.	[Optional Property]

53.8.7 Subdivision Surface Protocols

The name of the protocol as it appears in the ObjectHeader is "catmull-clark" or "loop" depending on the intended subdivision scheme. The protocol requires the **polygon** protocol.

The smoothing and any normals properties on the **polygon** protocol should be ignored if they exist.

The protocol indicates how the surface should be treated. Note that the canonical element type for each of the two schemes is not guaranteed to be the only element type stored in the file. For catmull-clark this means that triangles and general polygons will need to be made into quads. Similarily loop surfaces may have quads and other non-triangle primitives that need to be triangulated.

These protocols do not currently define methods for storing edge creasing parameters. Disclaimer: there are restrictions on what kind of topology surfaces are allowed to have for a given renderer (for example). In most cases surfaces need to be manifold. Some applications can deal with special cases better than others.

53.8.8 Image Protocol

The Image protocol describes image data in the form of an object. This data makes it possible to store texture maps, backgrounds, etc, directly in the GTO file.

When images are stored in a GTO file, use of Gzip compression is highly recommended if the data is unencoded. The supplied Reader and Writer classes default to using zlib compression.

If the image data is encoded, its better not to use compression on the GTO file (especially if the file contains only image data).

`imag`	The image data and other information will be stored in the **image** component.	[Required Component]
`int[` `imag` `size`	The size (and dimension) of the image. There will be N sizes in this property corresponding to the N dimensions of the image.	[Required Property]
`stri` `imag` `type`	The image type. For interactive purposes, the image channels may correspond to a particular fast hardware layout. `RGB` Three channels corresponding to red, green, and blue in that order., `BGR` Three channels corresponding to blue, green, and red in that order. `RGBA` Four channels corresponding to red, green, blue, and alpha in that order. `ABGR` Four channels corresponding to alpha, blue, green, and red in that order. `L` One channel corresponding to luminance. `HSV` Three channels corresponding to hue, saturation, and value. (The HSV color space). `HSL` Three channels corresponding to hue, saturation, and lightness. (The HSL color space). `YUV` Three channels corresponding to the YUV color space.	[Required Property]
`int[` `imag` `cs`	The coordinate system of the image. The value of **image.cs** can be any one of the following: **0 – Lower left origin.** : The first pixel in the image data is the lower left corner of the image data and corresponds to NDC coordinate (0,0). **1 – Upper left origin.** : The first pixel in the image data is the upper left corner of the image data and corresponds to NDC coordinate (0,0).	[Optional Property]

Any one of the following properties are required to hold the actual image data:

byte[N][image. pixels		[Prop- erty]
short[N] image. pixels		[Prop- erty]
string[1 image. type		[Prop- erty]
half[N][image. pixels	The element width determines the number of channels in the image. For example the type `byte[3][]` indicate a 3 channel 8-bit per channel image. The number of elements in this property should be equal to `image.size[0] &#42; image.size[1] &#42; ... image.size[N]` where **image.size** is the property defined above.	[Prop- erty]

Additional Image Properties Used by GTV Files.

The base GTO library does not deal with encoded image data or tiling of images. GTV is a specialization of the GTO format for storing movie frames. Some of the GTV properties are documented here. (See documentation for the GTV library for more info).

string[1] image. encoding	If the pixel data is encoded this property will indicate a method to decode it. Typical values are “jpeg”, “jp2000”, “piz”, “rle”, or “zip”. The pixels will be stored in the **image.pixels** as `byte[1][]` .	[Op- tional- Prop- erty]

53.8.9 Material Protocol

The name of the protocol as it appears in the ObjectHeader is “material”. The material protocol groups a parameters and a method (shader) for rendering. The material protocol can optionally include the **object** protocol.

The material definition is renderer and pipeline dependant. Material assignment is implemented using the **connection** protocol. See Section Inter-Object.

The **material** protocol is intended for use with software renderers. Interactive material definitions may be more easily defined on the assigned object.

`material`	Properties unrelated to parameters appear in the **material** component.	[Required Component]
`string[1][ type`	The value of the **material.type** property is renderer dependant. For a RIB renderer, the value of type might be “Surface”, “Displacement”, “Atmosphere” or a similar shader type name.	[Required Property]
`string[1][ shader`	The name of the shader. For RenderMan-like renderers this might be the name of an “.sl” file.	[Optional Property]
`string[1][ genre`	A property to further identify the material. This is most useful for identifying the target renderer for a material.	[Optional Property]
`parameters`	The set of parameters corresponding to the **material.type.**	Optional Component]

53.8.10 Group Protocol

53.8.11 Inter-Object Connection Protocol

The name of the protocol as it appears in the ObjectHeader is “connection” version 1.

Files which employ the **connection** protocol will typically contain a connection object with the special cookie name “:connections” indicating the purpose of the object as well as preventing namespace pollution. See Section Special Cookies.

Each component in a connection object is a connection type. For example, the “par- ent of” connection type is used to represent transformation hierarchies. In a connection object, there will be a single component called “parent of” which will contain the required properties **parent_of.lhs** and **parent_of.rhs** at a minimum. Some connection types may have additional data in the form of additional properties.

Connection components are transposable. The number of elements in properties comprising a connection component will be consistent. So a single “parent of” component can encode an entire scene transformation hierarchy.

Connection components have the following properties. Note that where **connection_type** occurs in the property name, you would substitute in the actual name of the connection type. (“parent of” for example).

`string[1][]`	`connection_type.lhs`	[Required Property]
`string[1][]`	`connection_type.rhs`	[Required Property]

The left-hand-side and right-hand-side of the connection.

- If the connection is directional, then an arrow indicating the direction would have its tail on the left-hand-side and its head pointing at the right-hand-side.
- If the connection type does not require a direction then these properties are still used to describe the two ends of the connection.
- Each entry will be the name of an object. There is no requirement that the ends of the connection exist in the file. For example, one end of the connection could be an image on disk.
- The empty string is a valid value. You could think of the empty string as indicating a grounded connection.

- It is ok for both ends of the connection to have the same value.

The GTO specification includes a couple of basic connection types.

Transformation hierarchies.

The "parent of" connection type is used to store transformation hierarchies. The connection type requires only the **lhs** and **rhs** properties. Transformation hierarchies are usually tree structures, but can also be DAGs (as is the case with Maya or Inventor).

Using "parent of" as a cyclic generalized network connection is probably an error for most applications. To be safe the topology of a "parent of" network should be a tree.

Material Assignment

The "material" connection type indicates a material assignment to an object. The left-hand-side name is a renderable object in a GTO file The right-hand-side is the name of a material object in a GTO file.

Container Assignment

The "contains" connection type indicates membership in a group or similar type of container object. The LHS is the group or container, the RHS is the object which is a member.

53.8.12 Difference File Protocol

If the **object.protocol** property contains the string "difference" then the object contains difference data; the data is relative to some other reference file.

For example, for animated deforming geometry its advantageous to write a reference file for geometry in its natural undeformed state then write only the **points.position** property in a gto file per frame to store animation. The **difference** minor protocol can apply to any major protocol.

If a reference file and a difference for file it exists, you can reconstruct the file represented by the difference file using the `gtomerge` command. See Section gtomerge.

53.8.13 Sorted Shell File Protocol

If the **object.protocol** property contains the string "sorted" and the object's major protocol is **polygon** then the object contains sorted shell data.

This protocol guarantees that the vertices and elements of shells—isolated sections of polygonal geometry—will be continguous in the **points** and **elements** components of the object.

`shells`	The **shells** component is transposable. Each property in the component should have the same number of elements.	[Required Component]
`int[1][] shells.vertices`	The number of contiguous vertices that make up the Nth element's shell.	[Required Property]
`int[1][] shells.elements`	The number of contiguous elements that make up the Nth element's shell.	[Required Property]

53.8.14 Channels Protocol

This minor protocol declares data mapped onto geometric surfaces. Usually the data is mapped using one of the parameterizations found in the **mappings** component of polygonal or sub-d geometry or possibly using the natural parameterization of a surface as is often the case with NURBS.

Each declared channel appears as a `string[1] []` property of a **channels** component on the geometry. The name of the property is the name of the channel. The property should contain at least one element.

The first element of the property should indicate the name of the mapping to use. This is either the name of one of the properties in the **mappings** component or "natural" indicating that the natural parameterization of the surface should be used.

The second and subsequent elements should contain the name of data to map. This could be a texture map file on disk, an image object in the GTO file, or a special cookie string. The lack of second element can be used as a special cookie.

`channels`	The component holds the names of all the channels on the geometry.	[Required Component]

Example

Here is a cube with "color", "specular", and, "bump" channels assigned.

```
Object "cube" protocol "polygon"
     Component "points"
          Property float[3][8] "position"
     Component "elements"
          Property byte[1][8] "type"
          Property short[1][8] "size"
     Component "indices"
          Property int[1][32] "vertex"
          Property int[1][32] "st"
     Component "mappings"
          Property float[2][24] "st"
     Component "channels"
          Property string[1][2] "color"
          Property string[1][2] "specular"
          Property string[1][2] "bump"
```

The contents of the "channels" properties might be:

```
string[1] cube.channels.color = [ "st" "cube_color.tif" ]
string[1] cube.channels.specular = [ "st" "cube_specular.tif" ]
string[1] cube.channels.bump = [ "st" "cube_bump.tif" ]
```

53.8.15 Animation Curve Protocol

The animation curve protocol defines a single component called **animation** in which each property holds an animation curve or data stream. The property's interpretation string indicates how the data should be evaluated.

Example

Here is a cube with animation curves.

```
Object "cube" protocol "polygon"
     Component "points"
          Property float[3][8] "position"
     Component "elements"
          Property byte[1][8] "type"
          Property short[1][8] "size"
     Component "indices"
          Property int[1][32] "vertex"
     Component "animation"
          Property float[6][2] "xtran" interpret as "bezier"
          Property float[6][2] "ytran" interpret as "bezier"
          Property float[6][2] "ztran" interpret as "bezier"
          Property float[6][5] "xrot" interpret as "bezier"
          Property float[6][8] "yrot" interpret as "bezier"
          Property float[6][10] "zrot" interpret as "bezier"
          Property float[1][100] "xscale" interpret as "stream"
```

In version 1, the transform protocol's object.globalMatrix property used to be of type `float[1] [16]` . This was a mistake that has been corrected in version 2. 2

In version 3 of the **transform** protocol, the **object.parent** property is redundant and therefor deprecated. The **connection** protocol handles the transformation hierarchy information and in a much more elegant manner See Section Inter-Object.

In version 1 of the polygon protocol, the **element.size** and **element.type** properties were combined into an **element.primitive** property. We felt that this was adding unnecessary complexity and because the primitive property was an int, it was taking up extra space.

The indices component in a polygonal object contains values which are analogous to the RenderMan face varying type modifier.

53.9 Naming Conventions

GTO files can contain cross references to parts of themselves, objects outside the file, or virtual/logical objects in applications. Because of the potential morass that can result from complete free-form naming, there are conventions which are part of the file specification.

Failure to follow the guidelines does not mean a GTO file is ill-formed; there's always a good reason to ignore guidelines. But having a basis for consistency is usually a good idea.

Some of these topics are a bit "advanced" in that they build off ideas that present themselves after using the file format for a while. If you are just learning about the format, consider this a reference section and skip it. If you're trying to decrypt a complicated GTO file with strange garbled naming, then this section is for you.

53.9.1 Valid Names

Names should be valid C identifiers, but should not contain the dollar-sign character ($). This means that no whitespace or punctuation is allowed.

Note that this does *not* apply to protocol names.

There is nothing in the sample `Reader` or `Writer` classes which enforces the valid name guideline. However, some applications (Maya) cannot handle names with whitespace and/or punctuation. So plug-ins which implement GTO reading/writing will have to enforce the application's specific naming requirements.

This guideline is broken by Section Special Cookies. It is also broken by Section Cross References.

53.9.2 Exactly Specifying a Property or Component

By convention, the full name or path name of a property is referred to like this:

```
OBJECTNAME.[COMPONENTNAME.]+PROPERTYNAME
```

where there can be any number of COMPONENTNAME parts. When indicating a property name relative to an object then:

```
[COMPONENTNAME.]PROPERTYNAME
```

should suffice. In this manual, names of components and properties are disambiguated using the dot notation. In addition, this is the format of output from the gtoinfo command. There is nothing about the GTO file itself which relates to this notation other than the cross-referencing naming convention discussed below.

53.9.3 Indicating Special Handling

Some objects, components, or properties in the GTO file may contain data for which names are not particularly useful or that may simply pollute the object or component namespace.

In other cases (component names most notably) the name may be used as information necessary to interpret data associated with it.

In order to distinguish these names from run-of-the-mill names, you should include a colon in the name. Names with colons are considered "special cookie" names and objects which have them may be handled differently than other objects.

The **connection** object protocol, for example, requires that a special file object exist to hold data. This object is not necessarily related to a logical object in the application, it is just a container for the connection data. These objects are named using the special cookie syntax. Usually the name is ":connections". See Section Inter-Object.

There is no rule regarding the placement of the colon in the name; it can appear anywhere in the name that is useful for the application. However, if the entire name is a special cookie—there is not additional information encoded in the name beyond itself—the recommend form is to have the colon be the first character.

53.9.4 Cross References Encoded in Names

Sometimes there is a need to have a property or component *refer* to another property, component, or object in the file (or somewhere else).

To cross reference the data in one property with another, simply name the property the full (or partial) path to the referenced property. For example, here's the output of `gtoinfo` on a GTO file which has cross referencing properties:

```
object "gravity" protocol "gravity" v1
     component "field"
          property float[3][1] "direction"
          property float[1][1] "magnitude"
     component ":datastream"
          property float[3][300] "field.direction"
          property float[1][300] "field.magnitude"
```

As you can guess, the intention here is that the properties called "field.direction" and "field.magnitude" in the ":datastream" component are data that is associated with the properties "direction" and "magnitude" in the "field" component.

53.10 Issues and Questionable Aspects of the Format

- There are currently no (publicly available) tools which verify that a file claiming to follow some protocol is correct.
- There is no 3D curve(s) protocol defined.
- The **NURBS** protocol does not handle trim curves. See Section NURBS Surfaces.
- The format does not contain dedicated space for auxillary information like the name and version of the program that wrote the file, the original owner, copyright information, etc. However, our tools use the string table for these type of data—since its not an error have an unused interned string, we store the data as such. In our opinion, this is a fairly innocuous method. You can read unreferenced strings by using the `gtoinfo` command with the `-s` option. Note that these strings are often lost when programs read and write the file. See Section gtoinfo.
- Although the format specification includes transposable components (those marked with the Gto::Matrix flag may be transposed), the current reader/writer library does not handle files with transposed components. It does handle components that are marked as Gto::Matrix but not transposed. See Section Particles.
- The use of special cookie names and special cross-reference names seems to seriously complicate the format if the protocol is not carefully conceived. For example, using `gtomerge` to merge files containing connections does not work—the connections are merged like all the other data in the file. The correct behavior would be to combine the connections, but merge the other object data. Perhaps this is just a case for integrating `gtocombine` into `gtomerge` ?
- Future versions should incorporate some form of check sum or some similar mechanism to do better sanity checking.
- There are many examples of properties whose data indexes into other property data. The most obvious of these are the polygon protocol **indices** properties. In order to combine gto files (concatenate polygonal data together for example) its necessary to know which properties are indexes and which are not. Index properties must be offset to be combined.
- The Boolean (bit fields) and Half data types are not implemented in the supplied writer library. Both of these types are useful in compressing geometric (and image) data.
- Material, Texture, and similar assignments and storage are usually very specialized at any particular production facility. The idea that a single method of encoding this information can be determined or enforced—or even

usefully be stored in a GTO file—is not realistic. However, we hope that some method can be determined that at least preserves a good portion of common data for transfer.

53.11 C++ Library

The GTO Reader/Writer library is written in a subset of C++. The intention was to make the library as portable as possible. Unfortunately we have only tried it on platforms that support gcc 2.95 and greater. It is known to work on various Linux versions and macOS. In either case it has been compiled with gcc.

53.11.1 Gto::Reader class

The Reader class (in namespace Gto) is designed as a fill-in-the-blank API. The user of the class derives from it; the base class defines a number of virtual functions which pass data to the derived class and ask the derived class questions about what data it wants.

The Reader class handles most of the difficult work in reading the file like keeping track of headers, sizes of properties, and the order of data. In addition, it handles the string table and looking up property string values. If the file was written by a machine with different sex (endianess) it will translate the data for you.

In addition, you can compile the GTO library with zlib support. This enables the Reader class to read gzipped GTO files natively and the Writer class to write them. This can be a significant space savings on disk and on saturated networks can make file loading faster. You can also pass a C++ istream object to the Reader if you want to read "in-core".

As the file is read, the Reader class will call its virtual functions to declare objects in the file to the derived class. The derived class is expected to return a non-null pointer if it wishes to later receive data for that object.

`Reader::Reader` *(unsigned int* `mode` *)*	The constructor argument mode indicates how the reader will be used. This value is a bit vector of the following or'ed flags:	
`Reader::None`		
The reader will be used in its standard *streaming* mode. The reader will attempt to read all the data in the file. This is the default value (or 0).		
`Reader::HeaderOnly`		
The reader will stop once it has read the header sections of the GTO file. This is an optimization that applies to binary files only. This option is ignored when reading a text file.		
`Reader::RandomAccess`		
The reader will read the header sections but not the data, however, it will initialize for use of the `Reader::accessObject()` function. Only binary GTO files can be read using the random access mode.		
`Reader::BinaryOnly`		
Only binary GTO files will be accepted by reader.		
`Reader::TextOnly`		
Only text GTO files will be accepte by reader.	[Constructor]	
`Reader::~Reader ()`	Closes file if still open.	[Destructor]
`bool Reader::open` *(const char** `filename` *)*	Open the file. The Reader will attempt to open file filename. If the file does not exist and zlib support is compiled in, the Reader will attempt to look for filename.gz and open it instead.	[Method]
`bool Reader::open` *(std::istream&, const cha** `name` *)*	Reads the GTO file data from a stream. The *name* is supplied to make error messages make sense.	[Method]
`void Reader::close ()`	Close the file and clean up temporary data. If the stream constructor was used, the stream is *not* closed.	[Method]
`std::string& Reader::fail` *(std::string* `why` *)*	Sets the error condition on the Reader and sets the human readable reason to *why* .	[Method]
`std::string& Reader::why ()`	Returns a human readable description of why the last error occured. (Set by the `fail()` function).	[Method]
`const std::string& Reader::stringFromId` *(unsigned int)*	Given a string identifier, this method will return the actual string from the string table.	[Method]
`const StringTable& Reader::stringTable ()`	Returns a reference to the entire string table.	[Method]
`bool Reader::isSwapped ()` *const*	Returns true if the file being read needed to be swapped. This occurs if the machine the file was written on is a different sex than the machine reading the file (for example a Mac PPC written file read on an x86 GNU/Linux box).	[Method]
`unsigned int Reader::readMode ()` *const*	Returns the mode value passed into the Reader constructor.	[Method]
`const std::string&`	Returns the name of the file or stream being read. This is the	[Method]

The following functions are called by the base class.

`void Reader::header` *(const* `Header&` `header` *)*	This function is called by the Reader base class right after the file header has been read (or created).	[Virtual]
`void Reader::descriptic` `()`	This function is called after all file, object, component, and property structures have been read. For binary files, this is just before the data is read. For text files, this is after the entire file has been read.	[Virtual]

The following functions return a `Reader::Request` object. This object takes two parameters: a boolean indicating whether the data in question should be read by the reader and a second optional data `void*` argument of user data to associate with the file data.

`Reader::Request::Reques` *(bool* `want` *, void** `data` *)*	*want* value of true indicates a request for the data in question. *data* can be any void*. *data* is meaningless if the *want* is false.	[Constructor]
`Reader::Request Reader::object` *(const std::string&* `name` *, const std::string&* `protocol` *, unsigned int* `protocolVersion` *, const ObjectInfo&* `header` *)*	This function is called whenever the Reader base class encounters an Object-Header. The derived class should override this function and return a Request object to indicate whether data should be read for the object in question. If it requests not to have data read, the Reader will not call the corresponding component() and property() functions.	[Virtual]
`Reader::Request Reader::component` *(const std::string&* `name` *, const ComponentInfo&* `header` *)*	This function is called when the Reader base class encounters a Component-Header in the GTO file. If the derived class did not express interest in a particular object in the file by returning `Request(false)` from the object() function, the components of that object will not be presented to the derived class. The derived class should return `Request(true)` to indicate that it is interested in the properties of this *component* .	[Virtual]
`Reader::Request Reader::property` *(const std::string&* `name` *, const char* `interpString` , const PropertyInfo& `header`)*	This function is called when the Reader base class encounters a Property-Header in the GTO file. If the derived class did not express interest in a particular object or the component that the property belongs to, the properties of that component will not be presented to the derived class. The derived class should return `Request(true)` to indicate it is interested in the property data.	[Virtual]
`void* Reader::data` (const PropertyInfo&, size_t `byts`)	This function is called before property data is read from the GTO file. The function should return a pointer to memory of at least size bytes into which the data will be read. The type, size, width, etc, of the data can be obtained from the `PropertyInfo` structure.	[Virtual]
`void Reader::dataRead` *(const PropertyInfo&)*	This function is called after the `data()` function if the data was successfully read.	[Virtual]

If you are using the Reader class in `Reader::RandomAccess` mode, you may call these functions after the read function has returned:

`Reader::Objects& Reader::objects ()`	Returns a reference to an std::vector of Reader::ObjectInfo structures. These are only valid after `Reader::open()` has returned. You can use these structures when calling `Reader::accessObject()` .	[Method]
`const Reader::Components& Reader::components ()`	Returns a reference to an std::vector of Reader::ComponentInfo structures. These are only valid after `Reader::open()` has returned. This method is most useful when deciding how to call the `accessObject` function.	[Method]
`const Reader::Properties& Reader::properties ()`	Returns a reference to an std::vector of Reader::PropertyInfo structures. These are only valid after `Reader::open()` has returned. This method is most useful when deciding how to call the `accessObject` function.	[Method]
`Reader::Request Reader::property` *(const std::string&* `name` , *const char* `interpString` , const PropertyInfo& `header`)*	This function is called when the Reader base class encounters a PropertyHeader in the GTO file. If the derived class did not express interest in a particular object or the component that the property belongs to, the properties of that component will not be presented to the derived class. The derived class should return `Request(true)` to indicate it is interested in the property data.	[Method]
`void Reader::accessObject` *(const ObjectInfo&)*	Calling this function on a GTO file opened for `RandomAccess` reading will cause the reader to seek into the file just for the data related to the object passed in. This is most useful when the objects' data cannot be held in memory and the order of retrieval is unknown. The reader attempts to be efficient as possible without using too much	[Virtual]

53.11.2 Gto::Writer class

The Writer class (in namespace Gto) is designed as an API to a state machine. You indicate a conceptual hierarchy to the file and then all the data. The writer handles generating the string table, the header information, etc.

The following is an example that outputs a polygon cube using the **polygon** protocol.

```
float points[3][] =
{ { -2.5, 2.5, 2.5 }, { -2.5, -2.5, 2.5 },
    { 2.5, -2.5, 2.5 }, { 2.5, 2.5, 2.5 },
    { -2.5, 2.5, -2.5 }, { -2.5, -2.5, -2.5 },
    { 2.5, -2.5, -2.5 }, { 2.5, 2.5, -2.5 } };

unsigned char type[] = { 2, 2, 2, 2, 2, 2 };
unsigned char size[] = { 4, 4, 4, 4, 4, 4 };

int indices[] = {0, 1, 2, 3, 7, 6, 5, 4,
          3, 2, 6, 7, 4, 0, 3, 7,
          4, 5, 1, 0, 1, 5, 6, 2 };

Gto::Writer writer;
writer.open("cube.gto");

writer.beginObject("cube", "polygon", 2); // polygon version 2

    writer.beginComponent("points");
```

(continues on next page)

(continued from previous page)

```
          // will write 8 float[3] positions
          writer.property("positions", Gto::Float, 8, 3);
     writer.endComponent();

     writer.beginComponent("elements");
          // one per face
          writer.property("size", Gto::Short, 8, 1, 1);
          writer.property("type", Gto::Byte, 8, 1, 1);
     writer.endComponent();

     writer.beginComponent("indices");
          // one per vertex per face
          writer.property("vertex", Gto::Int, 24, 1, 1);
     writer.endComponent();

writer.endObject();

// repeat writer object blocks if more objects

// output all the data in order declared

     writer.beginData();
     writer.propertyData(type);
     writer.propertyData(size);
     writer.propertyData(indices);
     writer.endData();
```

`Writer::Writer ()`	Creates a new Writer class object. Typically you'll make one of these on the stack. This constructor requires you call the open function to actually start writing the file.	[Constructor]
`Writer::Writer` *(std::ostream&)*	Creates a new Writer class object which will output to the passed C++ output stream.	[Constructor]
`Writer::~Writer ()`	Closes file opened with the `open()` function if still open. The destructor will not close any passed in output stream.	[Destructor]
`bool Writer::open` *(const char* `filename` , FileType `mode` = CompressedGTO)**	Open the file. The Writer will attempt to open file *filename* . If the file is not writable for whatever reason, the function will return false. If *mode* is `CompressedGTO` (the default value), the Writer class will output a binary compressed file. If the value is `BinaryGTO` the file will be binary uncompressed. If *mode* is `TextGTO` a text GTO file will be written. Compressed GTO files can be uncompressed manually using `gzip` . Compression is available only if the library is compiled with zlib support.	[Method]
`bool Writer::open` *(const char* `filename` , bool compress =* `true` *)*	This function exists for backwards compatibility. Use the other `open()` function instead. This function can open a file for binary output only (it cannot write a text GTO file).	[Method]
`void Writer::close ()`	Close the file and clean up temporary data. If the stream constructor was used, the stream is *not* closed.	[Method]
`void Writer::beginObje` *(const char* `name` , const char* `protocol` , unsigned int `version`) const*	Declares an object. Its components and properties must be declared before `endObject()` is called. The *name* is the name of the object as it will appear in the gto file. The *protocol* is the protocol string indicating how the object data will be interpreted and the *version* number indicates the protocol version. The Writer class does not verify that the data output conforms to the protocol.	[Method]
`void Writer::beginComp` *(const char* `name` , bool `transposed` =false)*	Declares a component. The component properties must be declared before a call to endComponent(). The *name* is the name of the component as it will appear in the gto file. The *transposed* flag is optional and indicates whether or not the component property data should be output transposed or one property at a time (the default).	[Method]
`void Writer::property` *(const char* `name` , Gto::DataType `type` , size_t `numElements` , size_t `partsPerElement` =1, const char* `interpString` =0)*	Declare a property. The *name* is the name of the property as it appears in the gto file. The *type* is one of `Gto::Double` , `Gto::Float` , `Gto::Int` , `Gto::String` , `Gto::Byte` , `Gto::Half` , or `Gto::Short` . *numElements* indicates the number of elements of size *partsPerElement* that will be in the property data. So for example, if the property is declared as a Gto::Float of with *partsPerElement* of 3 and there 10 of them, then the writer will expect an array of 30 floats when the propertyData is finally passed to it. The last argument *interpString* is an optional interpretation string that can be stored with the property.	[Method]
`void Writer::endCompon ()`	Closes the declaration of a component started by `beginComponent()` .	[Method]
`void Writer::endObject ()`	Closes the declaration of an object started by `beginObject()` .	[Method]
`void Writer::intern` *(const char* `string`)*	Declares a string to the Writer for inclusion in the file string table. When writing properties of type `Gto::String` , its necessary to call this function before the `beginData()` is called. Each string in the property data must be interned. When outputing the property, the property will be an array of `Gto::Int` in which each int is the result of the `lookup()` function which retrieves a unique int corresponding to interned strings.	[Method]

53.11.3 Gto::RawDataReader/Gto::RawDataWriter classes

These classes provide a quick method of reading the contents of a GTO file into memory for basic editing. The RawDataReader and RawDataWriter both use the same very primitive data structure that can be found in the RawData.h file. For examples of use, see `gtomerge` and `gtofilter` source code.

The RawData class shows how to both read and write using the supplied classes. In addition the reader subclass shows how to convert string data.

53.12 Python Module

The gto module implements a reader/writer library for the Python language. The module is implemented on top of the C++ reader and writer classes. The API is similar to the C++ API, but takes advantage of Python's dynamic typing to "simplify" the design. The Python module also implements a significant number of safety checks not present in the C++ library, making it an ideal way of exploring the Gto file format.

53.12.1 gto.Reader

The Reader class is designed as a fill-in-the-blank API much like the C++ library. The user of the class derives from it; the base class defines a number of functions which you override to pass data to the derived class and receive data from it.

As the file is read, the Reader class will call specific functions in itself to declare objects in the file. The derived class is handed data or asked to return whether or not it is interested in specific properties in the file.

The biggest difference from the C++ Reader class is that the `data()` method of the C++ class, which returns allocated memory for the library to read data into, cannot be overloaded in Python. Instead, the `dataRead()` method of the Python gto.Reader class is handed pre-allocated Python objects containing the data.

status `gto.Reader` *(mode)*	Create a new gto.Reader instance. Possible values for *mode* :	
gto.Reader.NONE		
The reader will be used in its standard *streaming* mode. The reader will attempt to read all the data in the file. This is the default value (or 0).		
gto.Reader.HEADERONL		
The reader will stop once it has read the header sections of the GTO file.		
gto.Reader.RANDOMAC		
The reader will read the header sections but not the data, however, it will initialize for use of the `gto.Reader.accessObject()` method.		
gto.Reader.BINARYONL		
The reader will only accept binary GTO files.		
gto.Reader.TEXTONLY		
The reader will only accept text GTO files.	[Constructor]	
`gto.Reader.open` *(filename)*	Opens and reads the GTO file *filename* . The function will raise a Python exception if the file cannot be opened.	[Method]
wants `gto.Reader.object` *(name,* `protocol` *,* `protocolVersion` *,* `objectInfo` *)*	This function is called by the base class to declare an object in the GTO file. The return value *wants* should evaluate to True or False, indicating whether or not the base class should read the object data. *name* and *protocol* are strings declaring name and protocol of the object, *protocolVersion* is an integer. *objectInfo* is an instance of a generic class which contains the same information as the Gto::ObjectInfo C++ struct.	[Method]
wants `gto.Reader.component` *(name,* `interpretation` *,* `componentInfo` *)*	This function is called by the base class to declare a component in the GTO file. The return value *wants* should evaluate to True or False, indicating whether or not the base class should read the component data. *name* is a string declaring the component name. *componentInfo* is an instance of a generic class which contains the same information as the Gto::ComponentInfo C++ struct.	[Method]
wants `gto.Reader.property` *(name,* `interpretation` *,* `propertyInfo` *)*	This function is called by the base class to declare a property in the GTO file. The return value *wants* should evaluate to True or False, indicating whether or not the base class should read the property data. *name* is a string declaring the full property name. *propertyInfo* is an instance of a generic class which contains the same information as the Gto::PropertyInfo C++ struct.	[Method]
`gto.Reader.dataRead` *(name, data, propertyInfo)*	If a property has been requested, the dataRead() function will eventually be called by the base class with the actual data in the file. The *name* is the name of a property, *data* is a tuple containing the property data, *propertyInfo* is an instance of a generic class which contains the same information as the Gto::PropertyInfo C++ struct.	[Method]
`gto.Reader.stringFromID` *(id)*	Returns the stringTable entry for the given string table id. Since the Python gto module returns strings directly, it is unlikely that you'll need to use this.	[Method]

53.12.2 gto.Writer

`gto.Writer ( )`	Creates a new writer instance, no arguments needed.	[Constructor]
`gto.Writer.open` *(filename, mode)*	Open the file. The Writer will attempt to open file *filename* . If the file is not writable for whatever reason, the function will raise a Python exception. The *mode* argument can be `BINARYGTO` , `COMPRESSEDGTO` (the default) or `TEXTGTO` .	[Method]
`gto.Writer.close ( )`	Close the file and clean up temporary data. Because of Python's garbage-collection, you can never be sure when a class's destructor will be called. Therefore, it is *highly* recommended that you call this method to close your file when it's done writing. You have been warned.	[Method]
`gto.Writer.beginObject` *(name, protocol, version)*	Declares an object. Its components and properties must be declared before endObject() is called. The *name* is the name of the object as it will appear in the gto file. The *protocol* is the protocol string indicating how the object data will be interpreted and the *version* number indicates the protocol version. The Writer class does not verify that the data output conforms to the protocol.	[Method]
`gto.Writer.beginComponent` *(name, interpretation, transposed)*	Declares a component. The component properties must be declared before a call to endComponent(). The *name* is the name of the component as it will appear in the gto file. The *transposed* flag is optional and indicates whether or not the component property data should be output transposed or one property at a time (the default).	[Method]
`gto.Writer.property` *(name, type, numElements, partsPerElement, interpretation)*	Declare a property. The *name* is the name of the property as it appears in the gto file. The *type* is one of gto.DOUBLE, gto.FLOAT, gto.INT, gto.STRING, gto.BYTE, gto.HALF (*Not implemented*), or gto.SHORT. *numElements* indicates the number of elements of size *partsPerElement* that will be in the property data. So for example, if the property is declared as a gto.FLOAT of with *partsPerElement* of 3 and there 10 of them, then the writer will expect a sequence of 30 floats when the propertyData is finally passed to it.	[Method]
`gto.Writer.endComponent ()`	Closes the declaration of a component started by beginComponent().	[Method]
`gto.Writer.endObject ()`	Closes the declaration of an object started by beginObject().	[Method]
`gto.Writer.intern` *(string)*	Declares a string to the Writer for inclusion in the file string table. When writing properties of type gto.String, its necessary to call this function for each string in the property data before the beginData() is called. The Python version of intern() can accept individual strings, as well as lists or tuples of strings.	[Method]
`int gto.Writer.lookup` *(string)*	Retrieve the identifier of the previously interned string. Valid only after beginData() has been called.	[Method]
`gto.Writer.beginData ()`	Begins data declaration to the Writer class. Only calls to lookup(), propertyData(), and endData() are legal after beginData() is called.	[Method]
`gto.Writer.propertyData` *(data)*	The propertyData() function must get exactly *one* parameter. That parameter can be any of the following:	
• A single int, float, string, etc.		
• An instance of mat3, vec3, mat4, vec4, or quat (http://cgkit.sourceforge. net /). DO NOT explicitly cast mat3 or mat4 into a tuple or list: `tuple(mat4(1))` . It will be silently transposed (a bug in the cgtypes code?). ADDING it to a tuple or list is fine: `(mat4(1),)`		

Tuples and lists are flattened out before they are written. As long as the number of atoms is equal to size x width, it'll work. Calls to propertyData() must appear in the same order as declared with the property() method. | [Method] | | `void gto.Writer.endData ()` | Closes the definition of data started by beginData() and finishes writing the gto file. Does *not* actually close the file–use the close() method for that. | [Method] |

53.12.3 Classes used by gto.Reader

These classes will contain the actual strings rather than string table IDs.

gto.ObjectInfo	This class emulates the Gto::ObjectInfo struct from the C++ Gto library. It is passed by the Python gto.Reader class to your derived `object()` method. The only methods implemented are `___getattr__` and `__repr__` . Available attributes are:
• `name` : String	
• `protocolName` : String	
• `protocolVersi` : Integer	
• `numComponents` : Integer	
• `pad` : Integer	[Class]
gto.ComponentInfo	This class emulates the Gto::ComponentInfo struct from the C++ Gto library. It is passed by the Python gto.Reader class to your derived `component()` method. The only methods implemented are `__getattr__` and `__repr__` . Available attributes are:
• `name` : String	
• `numProperties` : Integer	
• `flags` : Integer	
• `interpretatio` : String	
• `pad` : Integer	
• `object` : Instance of gto.ObjectInfo	[Class]
gto.PropertyInfo	This class emulates the Gto::PropertyInfo struct from the C++ Gto library. It is passed by the Python gto.Reader class to your derived `property()` and `dataRead()` methods. The only methods implemented are `__getattr__` and `__repr__` . Available attributes are:
• `name` : String	
• `size` : Integer	
• `type` : Integer	
• `width` : Integer	
• `interpretatio` : String	
• `pad` : Integer	
• `component` : Instance of gto.ComponentIn	[Class]

53.13 Utilities

53.13.1 The `gtoinfo` Utility

Usage: `gtoinfo [OPTIONS] infile.gto`

Options:

`-a/-all`	Output property data and header.
`-d/--dump`	Output property data (no header data is emitted).
`-l/--line`	Output property data one item per line. Can be used with either `-d` or `-s` .
`-h/--header`	Output header data.
`-s/ --strings`	Output sting table data.
`-n/ --numeric-st`	Output sting data as the raw string id instead of the string itself.
`-i/ --interpreta`	Output interpretation string data for components and properties if it exists.
`-r/ --readall`	Force reading of the enitre gto file even if only the header is being output.
`-f/--filter expression`	Only output information for properties who's long name (object.component.propname) matches the shell-like *expression* . Section gtofilter for examples of filter expressions. This option is similar to `gtofilter --include` option.
`--help`	Output usage message.

`gtoinfo` outputs the part of all of the contents of a gto file in human readable form. Its invaluable for debugging or just getting a quick understanding of what a gto file contains.

53.13.2 The `gtofilter` Utility

Usage: ‘ `gtofilter [OPTIONS] -o` *out.gto in.gto* ’

Options:

`-v`	Set verbose output. Whenever a pattern matches gtofilter will inform you.
`-ee/--exclude`	Regular expression which will be used to exclude properties.
`-ie/--include`	Regular expression which will be used to include properties.
`-regex`	Use POSIX regular expression syntax.
`-glob`	Use shell-like regular expression (fnmatch). This is the default.
`-t`	Output text GTO file.
`-nc`	Output uncompressed binary GTO file.
`-o` *out.gto*	Output .gto file

`gtofilter` can be used to remove objects, components, and properties from a gto file. You supply an include shell-like expression and/or an exclude shell-like expression. (The pattern matching is done using the fnmatch() function—see the main page for details.)

The patterns match each full property name. So for example a cube might have these properties:

```
cube.points.position
cube.elements.type
cube.elements.size
cube.indices.vertex
cube.indices.st
cube.indices.normal
cube.normals.normal
cube.mappings.st
cube.smoothing.method
cube.object.globalMatrix
cube.object.parent
```

Using the `--exclude` option, you can remove the object component by doing this:

```
gtofilter --exclude "*.object.*" -o out.gto cube.gto
```

Or if you wanted to pass through only the positions:

```
gtofilter --include "*.*.positions" -o out.gto cube.gto
     -or-
gtofilter --include "*positions" -o out.gto cube.gto
```

53.13.3 The `gtomerge` Utility

Usage: ‘ `gtomerge` *-o outfile.gto infile1.gto infile2.gto …* ’

Options:

`-o outfile.gto`	The resulting merged file to output.
`-t`	Ouput text GTO file.
`-nc`	Ouput uncompressed binary GTO file.

`gtomerge` takes a number of .gto input files and merges them into a single output .gto file. This is done by first creating output geometry that is identical to the first input file and then adding only those properties that are not already defined from subsequent gto files. The order of input files determines what will be in the final output file.

For **difference** files, you can use gtomerge to reconstruct a final file like this:

```
gtomerge -o out.gto difference.gto reference.gto
```

53.13.4 The gto2obj Utility

Usage: ‘ `gto2obj [OPTIONS]` *infile outfile* ’

Options:

`-o NAME`	When outputing GTO files, the name of an object in the GTO file to output. If not specified, the translator will output the first polygon, or subdivision surface it finds.
`-c`	When outputing GTO files, this option will force the protocol to be “catmullclark”.
`-l`	When outputing GTO files, this option will force the protocol to be “loop”.
`-t`	Ouput text GTO file.
`-nc`	Output uncompressed binary GTO file.

`gto2obj` takes either an input GTO file or Wavefront .obj file and outputs the other file type.

```
gto2obj in.obj out.gto
gto2obj in.gto out.obj
gto2obj -c in.obj out.gto ## output obj as subdivision surface
```

53.13.5 The gtoimage Utility

Usage: ‘ `gtoimage` infile outfile’

`-t`	Ouput text GTO file.
`-nc`	Ouput uncompressed binary GTO file.

`gtoimage` reads a TIFF file and converts it into a GTO file containing one image object. 32 bit floating point images, 16 bit and 8 bit integral images are directly converted. `gtoimage` expects the image to be two dimensional with three or four channels where the fourth channel is an optional alpha value. The output object conforms to the **image** protocol. See Section Image.

You can use `gtomerge` to merge the image object into another GTO file. See Section gtomerge.

It is highly recommend that the resulting output GTO file be written with compression or gzipped to reduce its size. Gzipped GTO files can be read directly by the supplied readers.

53.13.6 The RiGtoRibOut Utility

The `RiGtoRibOut` command is useful for:

- It can be used as a debugging tool for the RiGtoPlugin RenderMan plugin.
- It can be used as a drop-in replacement for RiGtoPlugin, for RIB renderers that do not support `Procedural DynamicLoad` , but that *do* support `Procedural RunProgram` . Note that this is substantially slower than using RiGtoPlugin, as all data needs to be translated to ASCII and back. It does have the one space-saving advantage of not needing to save ASCII RIB on disk.

- It could be used to generate RIB files that are read with `ReadArchive`. This is not recommended, as it negates all the advantages of using GTO in the first place. But if nothing else works, this should.

The command-line parameters are the same as the *CONFIG STRING* for RiGtoPlugin. See Section RiGtoPlugin.

53.13.7 The `gtoIO.so` Maya Plug-In

The Maya plugin comes in two parts: the C++ plugin which implements a Maya scene translator and an accompanying MEL script which implements the user interface.

The plugin handles export of NURBS surfaces (but not trim curves), polygonal geometry (which can be written as sub-division surfaces), and generic transforms. A Maya particle export tool is in the works. Additional user defined attributes can be emitted into the GTO file.

The plugin can import everything that it exports and also particle GTO files generated by other applications.

BUGS

The internal performance of Maya has changed between the 4.x and 5.0 versions. In Maya 5.0, the Maya API is *extremely* slow when importing polygonal normals. Importing of normals is turned off in Maya 5.0.

53.13.8 The RiGtoPlugin RenderMan plugin

Here you will find information on using the GTO RenderMan plugin. The documentation is complete enough to get started with, but should be considered a work in progress.

RIB Instantiation

The plugin is instantiated in a RIB Stream using the standard DynamicLoad procedural mechanism, like so:

```
Procedural "DynamicLoad" [ "RiGtoPlugin.so" "CONFIG_STRING" ] [ Bounding Box ]
```

If a bounding box is not known, the infinite box may be used:

```
Procedural "DynamicLoad" [ "RiGtoPlugin.so" "CONFIG_STRING" ] [ -1e6 1e6 -1e6 1e6 -1e6
```

Config String Syntax

The configuration string passed into RiGtoPlugin consists of a variable number of space-separated tokens. They are, in order:

1. Reference Pose GTO File Name
2. Shutter Open GTO File Name (optional)
3. Shutter Close GTO File Name (optional)
4. Primary On List (optional)
5. Primary Off List (optional)
6. Secondary On List (optional)
7. Secondary Off List (optional)

As shown, the only necessary element is the reference GTO file. For objects which do not have movement and do not require on lists or off lists, this is completely sufficient.

The logic behind the geometry instantiation mechanism is as follows:

- Read Reference GTO file The plugin reads all of the geometry in the reference GTO file. As a starting point, the shutter open and close geometry is set equal to the reference geometry.
- If requested, read Shutter Open GTO file The plugin then reads any geometry from the Shutter Open file that matches the name and geometry type of geometry that has already been read from the reference file—this geometry is stored as both the shutter open AND close geometry.
- If requested, read Shutter Close GTO file The plugin then reads any geometry from the Shutter Close file that matches the name and geometry type of geometry that has already been read from the reference file—this geometry is stored as the shutter close geometry
- Instantiate Geometry: For any piece of geometry that appears in BOTH on-lists and does not appear in EITHER off-lists, the plugin calls the appropriate RIB functions to create the requested geometry.

On-List/Off-List Syntax

The syntax of the on-lists and off-lists is as follows:

`NULL` is a special on-list/off-list which is interpreted as *all on* or *none off* .

Otherwise, the on-lists and off-lists are essentially shell-like regular expressions. The following rules apply:

- The `*` character matches any number of characters
- The `?` character matches any single of character
- Bracket expressions `[]` are supported. (See `man 7 regex`)
- Multiple patterns can be strung together with the `|` character.
- The pattern must match the *whole* object name. Thus, the pattern “ `*Sphere1` ” will match the object `nurbsSphere1` but *not* `nurbsSphere1Shape` . This is a very common “gotcha”.

As an example, suppose you wanted to turn off all of the geometry named `LeftLeg*Shape*` and `RightLeg*Shape*` in a render—you would create an off-list that looked like:

```
"LeftLeg*Shape*|RightLeg*Shape*"
```

Cache Management

By default, RiGtoPlugin maintains an internal cache of all of the file sets it has read. The cache is keyed off of Ref-Open-Close filename triplets. The reason for this is to facilitate easy material assignment, which will be discussed in greater detail below in the “Strategy” section.

In situations where memory is precious and the renderer needs as much memory as it can get, it may be advantageous to force RiGtoPlugin to discard its cached file sets. There is special syntax to facilitate this.

- To erase everything in RiGtoPlugin’s cache:

```
Procedural "DynamicLoad" [ "RiGtoPlugin.so" "__FLUSH__" ] [ Bounding Box ]
```

- To erase the cache associated with a given file triplet: (Using REF.gto, OPEN.gto and CLOSE.gto as standins for whatever files were actually passed in)

```
Procedural "DynamicLoad" [ "RiGtoPlugin.so" "REF.gto OPEN.gto CLOSE.gto __FLUSH__" ] [␣
↪Bounding Box ]
```

There is also an environment variable, `TWK_RI_GTO_NO_CACHE` , which if defined and set to anything other than "0", "FALSE", "False" or "false", will cause caching to be turned off entirely.

Environment Variables

`TWK_RI_GT(`	If this environment variable is defined and set to anything except "0", "FALSE", "False", or "false", RiGtoPlugin will treat all catmull-clark subdivision surfaces read from a GTO file as polygons instead.	[Environment Variable]
`TWK_RI_GT(`	If this environment variable is defined and set to anything except "0", "FALSE", "False", or "false", RiGtoPlugin will turn off all caching of geometry data to save memory.	[Environment Variable]

Usage Strategy

The RiGtoPlugin was designed with a particular data structure in mind. Used ideally, there would be a GTO file consisting of all of the geometry corresponding to a particular high-level creature or set in the scene. All of the surfaces corresponding to a hippo or a giraffe or a cyborg-monkey would be in a single GTO file. The animation data for this geometry would be contained in light-weight GTO files that contain only points that have moved and transformation matrices that have moved. The RiGtoPlugin only reads points and matrices from the Shutter-Open and Shutter-Close file, facilitating very light-weight "difference" files for animation data.

Because all of the geometry in a creature will have different materials assigned to it, on-lists and off-lists can be used to separate out only the geometry that shares a particular material.

Suppose we have a creature consisting of many surfaces but only three different materials—skinMtl, eyeMtl and hairMtl. The parts of the model have been named intelligently (for this example) such that the skin parts all have names like Skin*Shape*, the eye parts all have names like Eye*Shape*, and the hair parts all have names like Hair*Shape*. Then, the RIB for declaring this creature with material assignments might look like this:

```
AttributeBegin
Surface "skinShader" [ shader param settings ]
Procedural "DynamicLoad" [ "RiGtoPlugin.so" "thing.ref.gtothing.0013.open.gto thing.0
AttributeEnd

AttributeBegin
Surface "hairShader" [ shader param settings ]
Procedural "DynamicLoad" [ "RiGtoPlugin.so" "thing.ref.gtothing.0013.open.gto thing.0013.
↪close.gto Hair*Shape*" ][-1e6 1e6 -1e6 1e6 -1e6 1e6]
AttributeEnd

AttributeBegin
Surface "eyeShader" [ shader param settings ]
Procedural "DynamicLoad" [ "RiGtoPlugin.so" "thing.ref.gtothing.0013.open.gto thing.0013.
↪close.gto Eye*Shape*" ][-1e6 1e6 -1e6 1e6 -1e6 1e6]
AttributeEnd
```

Because of RiGtoPlugin's cache mechanism, the geometry associated with the file-set thing.*.gto is only read and interpreted one time—the on-lists control which parts of the geometry are instantiated at which times. To nuke the cache of these files (if memory is important), you would use the syntax:

```
Procedural "DynamicLoad" [ "RiGtoPlugin.so" "thing.ref.gtothing.0013.open.gto thing.0
```

Miscellaneous RenderMan Stuff

RiGtoPlugin stores some useful data in attributes that can be used by shaders if desired.

`Pref`	On ALL geometry RiGtoPlugin creates "varying point Pref" as part of its geometry declaration. This data can be accessed by simply putting "varying point Pref" in your shader parameters. The position of the model in the reference GTO file is always used for this parameter value.	[Shader parameter]
`Name`	RiGtoPlugin always places the name of the geometry, as retrieved from the GTO file, in an attribute that may be queried. It is exactly as if the following line of RIB were declared before the geometry were instantiated:	
`Attribute "identifier" "name" ["whatever my name is"]`	[RIB Attribute]	
`RefToWorld` *matrix*	RiGtoPlugin places the transformation matrix *objectToWorld* from the reference model into a user attribute called `refToWorld` . To prevent this attribute from being munged by the current transformation matrix, it is cast as a float[16] instead of a matrix. It is equivalent to this line of RIB:	
`Attribute "user" "float refToWorld[16]" [ the matrix values ]`	[RIB Attribute]	

CHAPTER

FIFTYFOUR

USING LUTS IN OPEN RV

Look up tables (LUTs) are useful for approximating complicated color transforms, especially those which have no known precise mathematical representation. RV provides four points in its color pipeline where LUTs can be applied: just after reading the file and before caching directly after the cache (file LUT), just before display transforms (look LUT), and as one of the display transforms (display LUT). The first three are per-source while there is only a single display LUT for each RV session.

Each of the LUTs can be either a channel LUT or a 3D LUT (the difference is explained below. In the case of a 3D LUT there can also be an additional channel pre-LUT which can be used to shape the data. Both types of LUT are preceded by an input matrix which can scale high dynamic range data into the range of the LUT input (which is the range [0,1]). The values the LUT produces can be outside of the [0,1] range. This makes it possible for any of the LUTs to transform colors outside of the typical [0,1] range on both input and output.

Internally, RV will store the LUT as either half precision floating point or 16 bit integral. Not all hardware is capable of processing LUTs stored as floating point (esp. the 3D LUTs) so if you notice banding or noisy output when using floating point LUT storage, you may have better luck with the 16 bit integral representation. If RV can determine whether the floating point LUTs are usable itself it will default to whatever is appropriate.

When applied in hardware, the LUTs are interpolated when a value is not exactly represented in the LUT. This is usually more of an issue with 3D LUTs than channel LUTs since they have fewer samples per dimension. When interpolating between sample values, RV uses linear interpolation for channel LUTs and tri-linear interpolation for 3D LUTs.

There are a number of ways to create a LUT. For film look simulation, it's often necessary to have special hardware to measure and compare film recorder output. Alternately, you use a lightbox; and assuming you have a well calibrated neutral monitor, "eyeball" the LUT by comparing the film to the monitor.

RV has two different algorithms for applying the LUTs on the GPU: using floating point or fixed-point integer textures. Not all cards are equally capable with 3D LUTs and floating point. If RV detects that the card probably can't do a good job with the floating point hardware it will switch to a fixed-point representation using 16 bit integer LUTs. Sometimes even though the driver reports that the LUTs can be floating point, you will see banding in the final images. If that occurs, try forcing the use fixed-point LUTs by turning off the Floating Point 3D LUTs item on the Rendering tab of the preferences. The fixed point LUT algorithm will perform just as well as floating point in 99% of normal use cases.

54.1 Channel (1D) versus 3D LUTs

A channel LUT (also called a 1D LUT) has three independent look-up tables: one each for the R G and B channels. The alpha channel is not affected by the channel LUT. Channel LUTs may be very high resolution with up to 4096 samples. Each entry in the channel LUT maps an input channel value to an output channel value. The input values are in the [0,1] range, but the output values are unbounded.

Channel LUTs differ from 3D LUTs in one critical way: they can only modify channel values independently of one another. In other words, e.g., the output value of the red channel can only be a result of the incoming red value. In a

3D LUT, this is not the case: the output value of the red channel can be dependent on any or all of the input red, green, and blue values. This is sometimes called channel cross-talk.

The other important difference between channel and 3D LUTs is the number of samples. Channel LUTs are one dimensional and therefor consume much less memory than 3D LUTs. Because of this, channel LUTs can have more samples per-channel than 3D LUTs.

The implication of all this is that channel LUTs are useful for representing functions like gamma or log to linear which don't involve cross-talk between channels whereas 3D LUTs are good for representing more general color transforms and

3D LUTs can be very memory intensive. A 64 × 64 × 64 LUT requires 64 3 × 4 bytes of data (3Mb). You can quickly run out of memory for your image on the graphics card by making the 3D LUT too big (e.g. 128 × 128 × 128, this will slow RV down). RV does not require the 3D LUT to have the same resolution in each dimension. You may find that a particular LUT is smooth or nearly linear in one or more dimensions. In that case you can use a lower resolution in those dimensions.

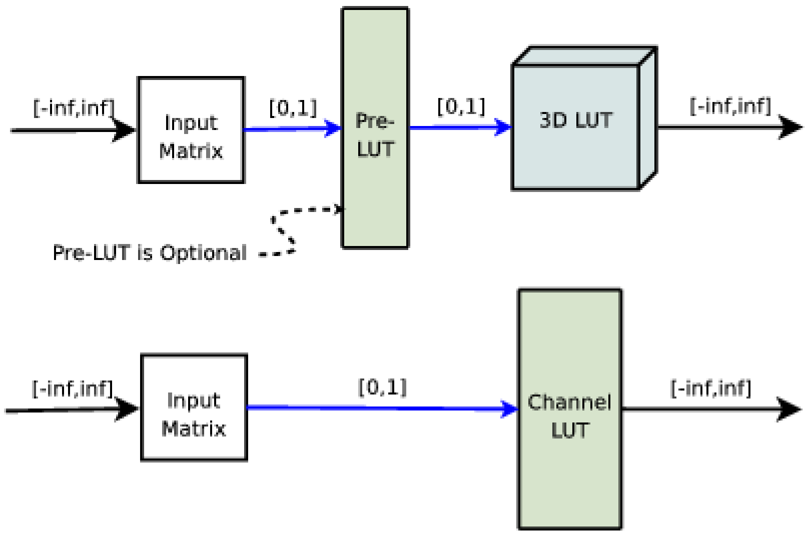
Some graphics cards have resolution issues with 3D textures which can cause loss of precision when RV's 3D LUT feature is enabled. On older NVidia cards and ATI cards in general, 3D textures may be limited to non-floating point color representations. Precision loss when using a LUT can be exacerbated by applying display gamma on these cards. To minimize precision loss on those types of cards, bake monitor gamma and/log-lin conversion directly into the display LUT. With newer GPUs this is not as much of an issue.

54.2 Input Matrix and Pre-LUT

For HDR applications, the incoming data needs to be rescaled and possibly shaped. RV has two separate components which do this: the LUT input matrix and a channel pre-LUT. The input matrix is a general 4 × 4 matrix. For HDR pixels, the matrix is used to scale the incoming pixel to range [0, 1]. The pre-LUT, the channel LUT, and the 3D LUT all take inputs in that range. The figure below shows a diagram of the channel and 3D LUT components and their input and output ranges.

The pre-LUT is identical to the channel LUT in implementation. It maps single channel values to new values. Unlike the general channel LUT, the pre-LUT must always map values in the [0, 1] range into the same range. The purpose of the pre-LUT is to condition the data before it's transformed by the 3D LUT.

For example, it may make sense for 3D LUT input values to be in a non-linear space – like log space. If the incoming pixels are linear they need to be transformed to log before the 3D LUT is applied. By using a relatively high resolution pre-LUT the data can be transformed into that space without precision loss.



3D and Channel LUT Components

54.3 The Pre-Cache LUT

The first LUT that the pixels can be transformed by is the pre-cache LUT. This LUT has the same parameters and features as the other LUTs, but it is applied before the cache. The pre-cache LUT is currently applied by the CPU (not on the GPU) whereas the file, look, and display LUTs are all used by the graphics hardware directly. For this reason the pre-cache LUT is slightly slower than the others.

The pre-cache LUT is useful when a special caching format is desired. For example by using the pre-cache LUT and the color bit depth formatting, you can have RV convert linear OpenEXR data into 8 bit integer format in log space. By using RV's log to linear conversion on the cached 8 bit data you can effectively store high dynamic range data (albeit limited range) and get double the number of frames into the cache. Many encoding schemes are possible by coupling a custom pre-cache LUT, change of bit depth, and the hardware file LUT to decode on the card.

54.4 LUT File Formats

Extension	Type	1D	3D	PreLUT	Float	Input	Output
csp	Rising Sun Research	•	•	•	•	$[-\infty, \infty]$	$[-\infty, \infty]$
rv3dlut	RV 3D		•		•	[0 , 1]	$[-\infty, \infty]$
rvchlut	RV Channel	•			•	[0 , 1]	$[-\infty, \infty]$
3dl	Lustre		•			[0 , 1]	[0 , 1]
cube	IRIDAS		•		•	[0 , 1]	$[-\infty, \infty]$
any	Shake	•			•	[0 , 1]	$[-\infty, \infty]$

LUT Formats (as Supported in RV)

RV supports several of the common LUT file formats. Unfortunately, not all LUT formats are equally capable and some of them are not terribly well defined. In most cases, you need to know the intended use of a particular LUT file. For example, it doesn't make sense to apply a LUT file which expects the incoming pixels to be in Kodak Log space to pixels from an EXR file (which is typically in a linear space). Often there is no way to tell the intended usage of a LUT file other than its file name or possibly comments in the file itself. Most formats do not have a public mechanism to indicate the usage to an application.

To complicate matters, many LUT files are intended to map directly from the pixels in a particular file format directly to your monitor. When using these types of LUTs in RV you should be aware than making any changes to the color using RV's color corrections or display corrections will not produce expected results (because you are operation on pixels in the color space appropriate for the display, rather than in linear space).

One of the more common types of LUT files you are likely to come across is one which maps Kodak Log space to sRGB display space. The file name of that kind of LUT might be log2sRGB or something similar. A variation on that same type of LUT might include an additional component that simulates the look of the pixels when projected from a particular type of film stock. Strictly speaking, you do not need to use log to sRGB LUTs with RV because it implements these functions itself (and they are exact, not approximated). So ideally, if you require film output simulation you have a LUT which only does that one transform. Of course this is often not the case; the world of LUT formats is a complicated one.

54.4.1 RSR LUT Format

Currently, the best LUT format for use with RV is the .csp format. This format handles high dynamic range input and output as well as non-linear and linear pre-LUTs. It maps most closely to RV's internal LUT functions.

There is one type of .csp file which RV does not handle: a channel LUT with a non-linear pre-LUT. This is probably a very rare beast since an equivalent 1D LUT can be created with a linear pre-LUT. An error will occur if you attempt to use a channel LUT with a non-linear pre-LUT.

When RV reads a pre-LUT from this file format and it can determine that the pre-LUT is linear, it will convert the pre-LUT into a matrix and apply it as the LUT input matrix. In that case the non-linear channel pre-LUT is not needed. If the pre-LUT is non-linear (in any channel) RV will construct a channel LUT which is used just before the 3D LUT. Input values in the .csp pre-LUT are normalized and the scaling is then moved to the input matrix. Using a matrix when possible frees up resources for other LUTs and images in the GPU. Any pre-LUT in a .csp file with only two values is by definition a linear pre-LUT.

```
 CSPLUTV100
3D

2 ^\label{preLUTStart}^
0 13.5
0 1
2
0 13.5
0 1
2
0 13.5
0 1 ^\label{preLUTEnd}^

...
```

In the above listing, lines preLUTStart to preLUTEnd are linear pre-LUT values. In this case the pre-LUT values are mapping values int the range [0,13.5] down to [0,1] for processing by a 3D LUT (which is not shown). For a summary of the RSR .csp format see Appendix *G* .

For the most part, it's not necessary to know the distinction between a linear and non-linear pre-LUT in the file. However, the behavior of the pre-LUT outside the bounds of its largest and smallest input values will be different for linear pre-LUTs. Since the pre-LUT is represented as a matrix, it will not clamp values outside the specified range. Non-linear pre-LUTs will clamp values.

CHAPTER

FIFTYFIVE

USING MULTIPLE MEDIA REPRESENTATIONS IN OPEN RV

55.1 Introduction

Multiple Media Representations makes it easy for you to swap between media representations in RV.

Multiple Media Representation (MMR) is the implementation of the new OTIO feature named *Media Multiple References*. With RV MMR, you easily switch between the different source media representations referenced by the timeline.

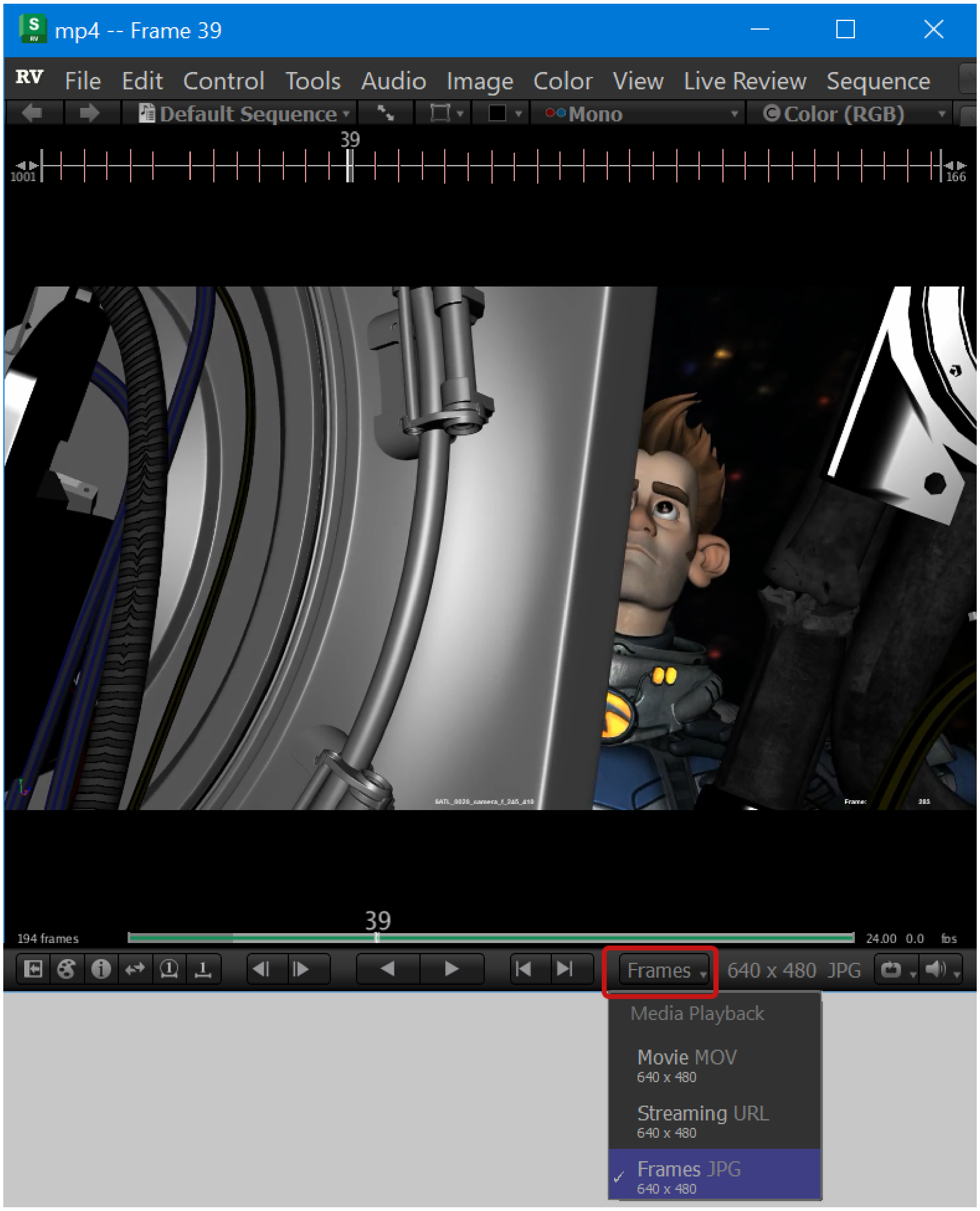
MMR allows you to switch between multiple source media representations since these representations are expressed in the RV graph.

55.2 Swap between source media representations

In RV, the Multiple Media Representations feature allows you to swap between source media representation for any given frame.

To swap source media:

1. Click the Swap drop-down menu.
2. Select the source media to display.

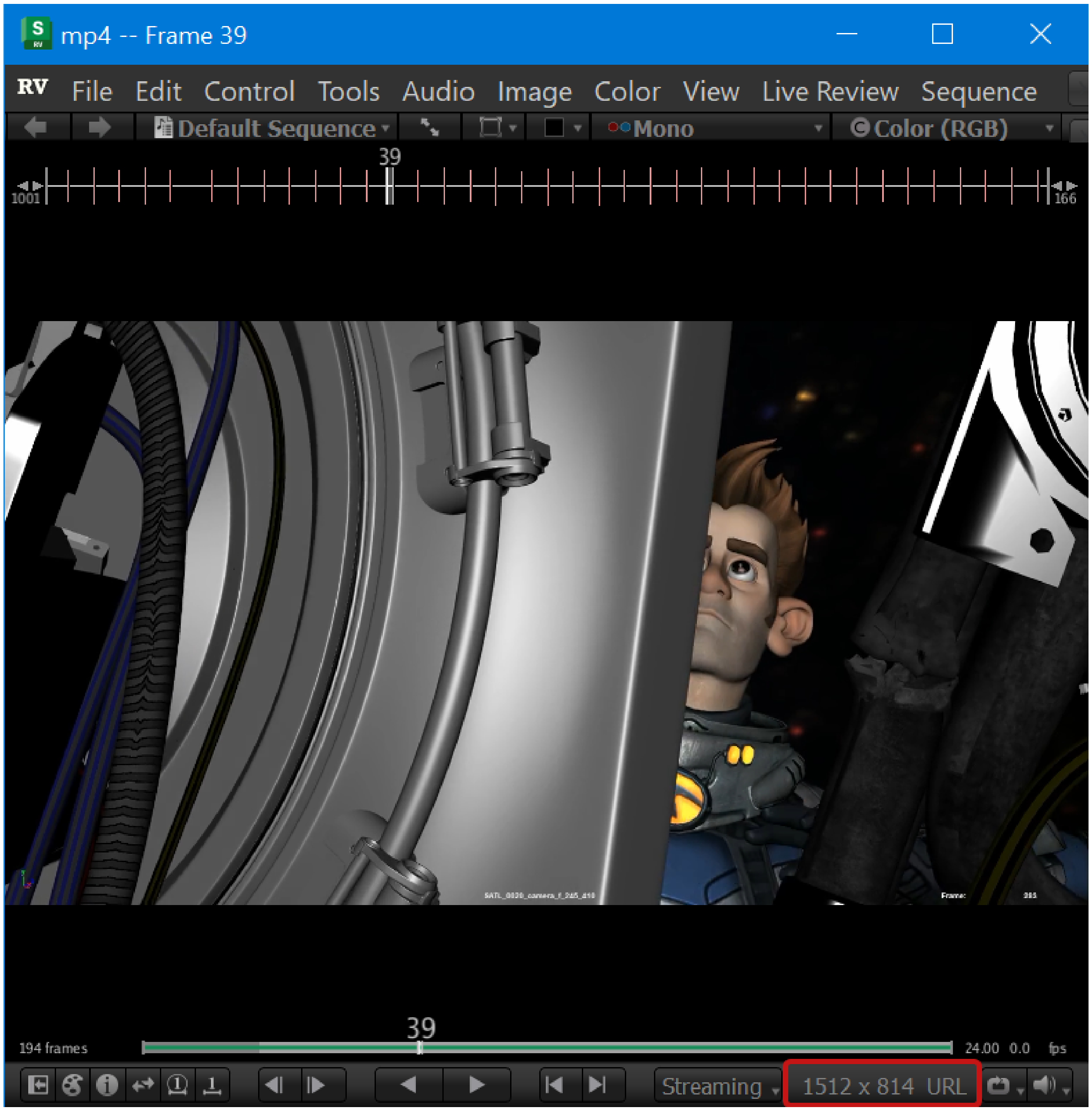


You can only select available source media. Any dimmed option is unavailable to you.

By default, there are three possible options.

- **Frames**: for Image sequences, usually OpenEXR.
- **Movie**: for Movie file.
- **Streaming**: for Media streaming.

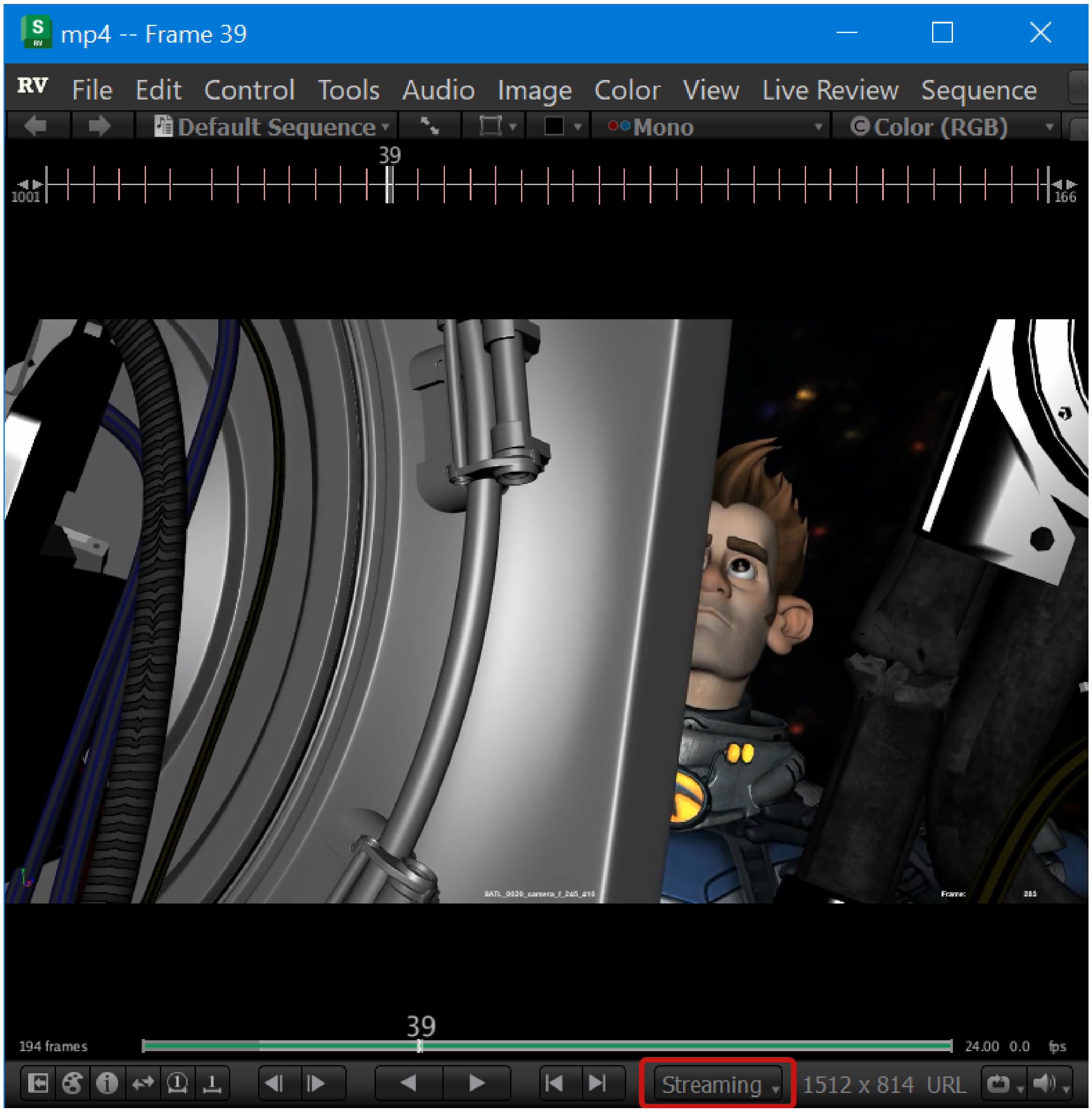
The resolution and type of media your actually viewing are displayed on the toolbar, next to the Swap Media drop-down.



Important: During playback, if you lose access to the source media, RV falls back to the next available source media, in this order: Frames > Movie > Streaming(S3).

55.2.1 How do I know my clip has multiple representations

If the Swap menu appears in the playback area, your clip has multiple media representations. If the menu is not there, the media you see is the only media available and you cannot swap sources.



55.3 Exporting MMR clips to OTIO

When you export a clip to OTIO, the currently selected media representation is set as the active reference in the OTIO file. This requires exporting using a version of OTIO that supports *Media Multiple References*, which means OTIO 0.15 or later. If your version of OTIO does not support MMR, only the current media representation is exported.

For technical information on MMR in OTIO, see *Media Multi-Reference Support*.

55.4 Multiple Media Representations: Technical Details

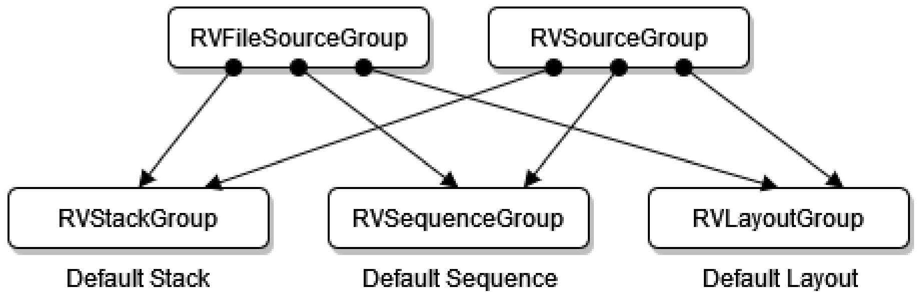
55.4.1 Setting Open RV Fall Back Order on Missing Source Media

During playback, if RV loses access to the source media selected by the user, RV falls back to the next available source media. It follows this order: Frames > Movie > Streaming(S3).

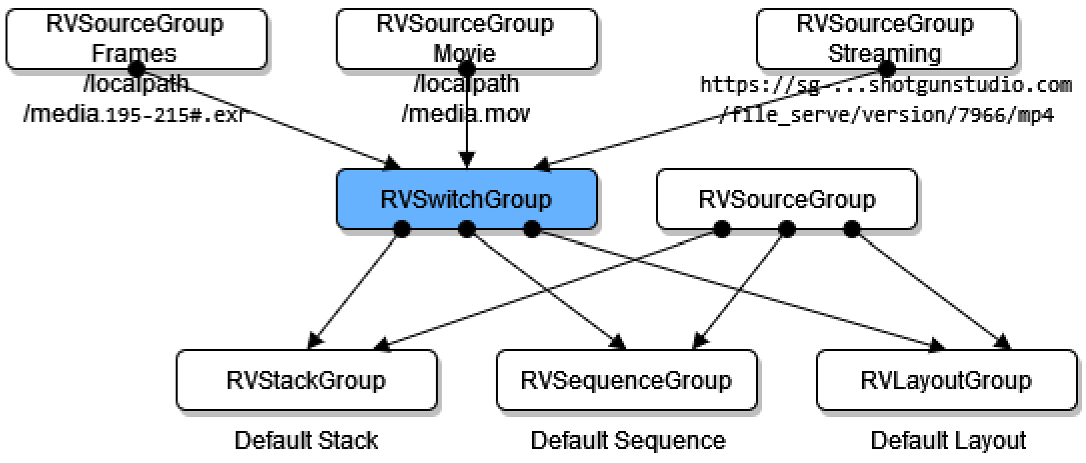
You can implement your own fallback order by overloading the package `RV Multiple Source Media Representation Management`. See the package notes for more details.

55.4.2 Multiple Media Representations Underlying Schema

When *not* using multiple media representations, the RV session is identical to that of previous versions.



When using multiple media representations either with the new `addSourceMediaRep()` command or the new `+mediaRepName` option used together with `addSources()`, `addSourceVerbose()`, or `addSourcesVerbose()`, the `SwitchGroup` gets inserted to connect the different source media representations.



The SwitchIPNode is available in older versions of RV. This means that older versions of RV are able to load a new RV session containing multiple media representations.

55.4.3 New commands

string addSourceMediaRep(string sourceNode or "last" for last created source node, string mediaRepName, string[] mediaRepPathsAndOptions=[], string tag=[])

Add a media representation to an existing source specified by sourceNode with an optional tag.

If the source media representation already exists, addSourceMediaRep() throws the following error:

```
ERROR: Source media representation name already exists: [media representation name]
```

Returns:

string sourceNode - returns the name of the source node created

Params:

string sourceNode - The source node for which to add the media representation or "last" for the last created source.

string mediaRepName - The name of the media representation to add.

string[] mediaRepPathsAndOptions - The paths or URLs of the media representation to add and additional options of the media representation to add, if any.

string tag - An optional tag can be provided which is passed to the generated internal events. The tag can indicate the context in which the addMediaRep() call is occurring (for example, drag, drop, etc.) It's also possible to add a source media representation in RV without calling addSourceMediaRep(): specify the mediaRepName and mediaRepSource options to the addSources(), addSourceVerbose(), or addSourcesVerbose() commands.

Examples:

```
rv.commands.addSourceMediaRep("sourceGroup000000_source", "Movie", ["hippo_numbered.mov
↪"])
rv.commands.addSourceMediaRep("sourceGroup000000_source", "Streaming", ["https://www.
↪acme.com/file_serve/version/7966/mp4"])
rv.commands.addSourceMediaRep("sourceGroup000000_source", "Frames", ["hippo_numbered.195-
↪215#.jpg"])
rv.commands.addSourceMediaRep("sourceGroup000000_source", "Movie", ["left.mov", "right.
↪mov"])
rv.commands.addSourceMediaRep("sourceGroup000000_source", "Movie", ["hippo_numbered.mov",
↪ "+rs", "194", "+pa", "0.0", "+in", "195", "+out", "215"])
```

string tag exemple:

```
rv.commands.addSourcesVerbose(["image_sequence.195-215#.jpg", "+mediaRepName", "Frames",
↪"+mediaRepSource", "last"])
```

void setActiveSourceMediaRep(string sourceNodeOrSwitchNode, string mediaRepName)

Set the active input of the Switch node specified or the ones associated with the specified source node to the given media representation specified by name.

When `sourceNode` is an empty string "", then the media in the first Switch node found at the current frame is swapped with the media representation specified by name.

When `sourceNode` is "all", then the media in all the Switch nodes created with the `rvc.addSourceMediaRep()` command is swapped with the media representation specified by name.

> Note: If the Source Groups do not contain the specified media representation, then `setActiveSourceMediaRep()` throws an error unless `sourceNodeOrSwitchNode` is "" or "all".

- Set the active media representation for the current source to *Streaming*:

```
rv.commands.setActiveSourceMediaRep("", "Streaming")
```

- Set the active media representation for all the sources to *Streaming*:

```
rv.commands.setActiveSourceMediaRep("all", "Streaming")
```

- Set the active media representation for the Switch nodes associated with the specified source to *Streaming*:

```
rv.commands.setActiveSourceMediaRep("sourceGroup000000_source", "Streaming")
```

string sourceMediaRep(string sourceNode)

Returns the name of the media representation currently selected by the Switch Group corresponding to the given `RVFileSource` node.

When `sourceNode` is an empty string "", then `sourceMediaRep()` returns the name of the currently selected media representation corresponding to the first Switch node found at the current frame.

string[] sourceMediaReps(string sourceNode)

Returns the names of the media representations available for the Switch Group corresponding to the given RVFileSource node.

If `sourceNode` is "", then `sourceMediaReps()` returns all the possible source media representation names.

CHAPTER

FIFTYSIX

MU PROGRAMMING LANGUAGE

56.1 1. Overview

Which language should you use to customize RV? In short, we recommend using Python. You can use Python3 in RV in conjunction with Mu, or in place of it. It's even possible to call Python commands from Mu and vice versa. Python is a full peer to Mu as far as RV is concerned.

Mu is principally targeted towards computer graphics applications. The original (usable) version appeared at Tweak Films around 2001-2002. Over the years its syntax and runtime has been refactored and evolved from a simple shading language to a nearly full featured general language. However, Mu is still probably best suited for computer graphics than other computing tasks.

The following discussion is mostly a result of experience in the feature film special effects industry (which I'll call film). One could argue that computing tasks for film are extreme in many cases and that achievements and trends there tend to trickle down into related CG disciplines (like games and television post production). Indeed, over a short period of time the distinction between these CG disciplines has blurred.

Mu is an attempt to unify a number of disparate uses of "scripting" and compiled languages in CG. Using Mu in conjunction with C++ applications tends to hit a sweet spot; since Mu is itself written in C++ attention has been paid to making it easy to embed and use in those applications. The static type system has a number of advantages that lead to "better" user code and often easier to maintain.

Some background on computing in the CG film industry is useful here. To start with, the big computational tasks which affect application language choice fall into these catagories:

56.1.1 1.1 Rendering

While there are a number of different kinds of rendering software, there are two that are used most in film production: renderers based on the Reyes algorithm (Pixar's RenderMan being the obvious example), and ray tracing (Mental Images' Mental Ray possibly producing the most ray-traced pixels on film to date. Pixar's Photorealistic RenderMan has actually become a ray tracer as well, and Mental Ray is really a hybrid scanline renderer, but for purposes of this discussion I'll just resort to blatant over-simplification.) These two algorithms have very different profiles: the Reyes algorithm shines on a fast SIMD architecture with fast memory cache lines, and ray tracing (which typically results in a lot of point sampling) benefits mostly from raw speed. Typically render times follow the so-called *one hour rule* (It *used* to be one hour; now its pushing two to three hours.): the rendering scene for a given shot will increased in complexity until the render time hits one hour at which point the complexity will no longer increase.

Its not atypical for a single frame to require gigabytes of image data (textures) and geometry. This is typically multiplied by the number of frames in a shot — which has unforunately *increased* over the last few years. With the current trend towards stereoscopic film and all CG productions (feature CG animation) this is becoming even more extreme. Tricks like locally caching data, wavelet compression of images, and procedural shaders are often used to reduce the data and throughput complexity. Some newer real world sampling techniques like BDRF, and real time motion and 3D geoemtry capture have futher increased data demands on renderers.

Its not surprising that production renderers have historically been written in C and more recently C++. These languages allow application programmers to very carefully manage computing resources. The cost of course is complexity of the code and long debugging periods. But usually its worth it: the renderer's efficiency can directly translate into time and money (and there is a very strong incentive to minimize these!).

Mu has been used for two purposes with regard to rendering: as a shading language (like the RenderMan shading language) and as scene generation and storage language. The first use was for scene generation as a particle instancing language.

56.1.2 1.2 Compositing

Compositing is similar to rendering: multiple images are rerendered into a new image. In the process pixels may be modified, moved, or generated. While not as compute intensive as 3D rendering, compositing can consume substantial resources as well.

Recently a trend has developed where renderers produce "partially" rendered images which are actually coefficents in the rendering equation; these are then recombined with new coefficients in the compositing software thereby granting more flexibility after rendering. For example, each light in a scene is effectively turned on by itself creating a layer which is then dialed up and down in the compositor (since light can be linearly combined). This has resulted in a much larger input image set than previously.

Compositing software is fairly complex compared to most software. Unlike renders, compositing software requires a good deal of user interface to make it usable.

Like a shading language, a high-level compositing language is typically SIMD. Mu has been used in this context to make a per-pixel compositing scripting language.

56.1.3 1.3 Animation and Modeling

Animation and modeling software can be separate, but the trend has been towards large swiss army knife feature sets which include *everything* you might want to do in animation and modeling combined. The software used for film (Alias' Maya for example) is the result of 20 years of evolution. These programs are huge and typically deal with complexities exceeding CAD software (in some uses they are the CAD software). It would not be far fetched to argue that some of these programs are the most complex software written to date.

Most require very workstation computers, fast graphics cards, fast network I/O and have massive amounts of user interface which is often procedurally generated. They are dump- ing grounds for advanced algorithms spanning computational geometry to computational physicals to signal processing.

Most of the current crop of 3D modeling and animation packages are "scriptable". This usually means they have a fairly high-level interpreted language availble for model and animation construction or even as an extension language for adding novel behavior. Usually the language is also used to control and define the user interface of the application (since this is typically a good design choice for large software projects). In addition to the above, Maya even uses the its language (MEL) as its scene file format.

Mu is ideally suited for these same tasks: its a very high level language with specific built-in types which are typically required for animation and modeling software. It can be used as an interpreted language or it can be compiled. The application can add opaque types and APIs to Mu which easily bind to internal functions. Since Mu was designed for perforance, the language does not become a huge bottleneck even during scene evaluation.

56.1.4 1.4 Simulation

Physical simulation has been a part of CG since its inception. Over time this has only increased. Simulation software for film has usually been embedded in *procedural animation* systems which allow the user to programmaticly control physical elements in time and space.

So to sumarize the computing landscape in CG film:

- **Large datasets are common.** Many applications will use one minus the number of bytes available to them.
- **Performance is really important.** The artist's attention span is closely related to ren- dering and simulation times.
- **Workers usually have informal training for programming.**
- **Workers are skilled tool users.**

56.2 2. Primitive Types

Mu types have two kinds of semantics: reference and value. Types with value semantics are always copied when assigned or passed to a function as an argument. Basic number types all have value semantics.

56.2.1 2.1 Boolean type

Mu has a type for boolean values called `bool` and its two constants true and false.

Boolean values do not cast to other integral values by default as in the C language. So many C/C++ idioms are not applicable in Mu.

```
int a = 0;
if (a) doit(); // error
```

The above example does not work in Mu (Well this isn't entirely true: you can make a cast operator from int and float to bool and get the C++ behavior). The int will not cast to a bool. The correct way to do the above is:

```
int a = 0;
if (a == 0) doit(); // ok
```

All of the conditional constructs take the bool type as their test argument.

56.2.2 2.2 int64, int, short , and byte types.

Mu has four integral value types. The int64, int, and short types are represented as signed twos complement. The int64 type is 64 bits, the int type is 32 bits and the short type is 16 bits. byte is 8 bit unsigned. The binary representation in memory is machine dependant.

The integral types all obey basic arithmetic operations as well as the bitwise operators. None of the integral types cast to the boolean type bool automatically.

There is a distinction between the `byte` and `char` type in Mu. The char type is not assumed to be of any particular size. There are no "unsigned" integral types.

56.2.3 2.3 Floating point types.

The floating point type is called float and is a 32 bit IEEE floating point number. Mu can also be compiled with half (a 16 bit floating point number) as defined by ILM's Imath library. The usual arithmatic operators work on floats. Underflow and overflow should behave consistantly across platforms.

Mu floats do not throw exceptions by default.

56.2.4 2.4 Character type.

The `char` type represents a unicode character.

To make a character constant in Mu, use signle quotes around a single character:

```
char c = 'x';
```

For unicode values when the input file is not encoded as UTF-8 or another accepted coding, you use the unicode escape sequence to specify non-ASCII characters:

```
char c = '\u3080'; \
```

Characters may be cast to strings.

```
string s = 'c'; // string => "c" \
```

(See section on strings for more info). Operations on the char type are all related to distances between characters:

```

int diff = "c" - "a"; // int => 2
char a = 'a';
char b = a + 1; // next character after 'a'
a++; // next character
assert(a == 'b');
```

56.2.5 2.5 The void Type.

The `void` type is useful in the context of function declarations. You cannot make a variable of type `void` since it has no value. The `void` type indicates that absence of a return type for a function definition:

```
  function: print_it (void; int i) { print(i + "\n"); }
```

In the example the function `print_it` has no return value so the type void is used to indicate that.

56.2.6 2.6 The nil Type.

The value `nil` is used to set a reference variable to refer to “nothing”. It is analogous to NULL in C/C++, null in Java, nil in lisp, or None in Python.

`nil` is of type `nil` . The type is special in that it can be automatically cast to any reference type.

You cannot test any type value against nil using ==. Instead operator eq or neq must be used:

```
string s = “hello”; \
if (s eq nil) print("error\n"); // syntax error
```

56.2.7 2.7 Vector Types

The vector types are primitive types which are passed by value. To specify a vector type you use the vector type modifier:

```
vector float[4] hpoint = {1, 2, 3, 4};
vector float[3] point = {1, 2, 3};
vector float[2] ndc = {0.0, 1.0};
```

There are currently only three types which are allowed to be used with the vector type modifier. These types are: float[2], float[3], and float[4].

Vector types allow access to their members using member variables. The member vari- ables are called x, y, z, and w. In addition you can use the indexing notation (operator[]). For the two dimensional type, z and w are not available. For the three dimensional type, w is not available. For example, here is a normalize function:

```
function: normalize (vector float[3]; vector float[3] v)
{
float len = math.sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
return v / len;
}
```

You can also set the members directly:

```
vector float[3] v = {1, 2, 3};
v.x = v.y;
```

To make arrays of vectors, it may be necessary to include parentheses in the type definition:

```
(vector float[3])[] array_of_points;
```

See also symbolic assignment operator for creating type aliases.

56.2.8 2.8 Function Types

Function types are declared much like functions themselves. The type of a function encodes its return type and all its argument types (whether or not it has default values).

For example, this is the type for a function that takes two ints as arguments and returns an int:

```
(int;int,int)
```

Whitespace inside the declaration is ok, but for clairity it is omitted here. In practical usage, you might want to create a variable and assign a function to it like so:

```
function: add (int; int a, int b) { a + b; }
(int;int,int) x = add;
```

Since functions are first class objects in Mu, you can make arrays of them:

```
 (int;int,int)[] int_funcs = { add };
```

For example, this makes a dynamic array of functions which return an int and take two ints and initializes it to have a single element: the function add.

Of all the type declaration syntax, the function type syntax is the most complex. Here's an example of a function type which takes a dynamic array of string and returns a function that returns an int and takes a float[4,4] matrix as an argument:

```
 ((int;float[4,4]);string[]) \
```

See the section on symbolic assignment for how to clean up a mess like the above.

2.8.1 The Ambiguous Function Type

One problem that arises in a type system with function overloading is ambiguity resolution (if ambiguity is allowed). When a function type expression is evaluated for an overloaded function, the result is the ambiguous function type:

```
mu>
(;) => print \
```

In this case, the print function is overloaded and therefor is returned as type (;). An expression or variable of type (;) cannot be evaluated using operator(). However, the type will automatically cast to a callable function type (as seen in this interpreter session):

```
mu> (void;float)(print)
(void;float) => print (void; float)
```

In this case the result is no longer ambiguous. If you attempt to cast to a function type for which there is no overloaded instance, the cast will throw an exception.

56.3 3. Built-in Types

Mu has built-in syntax for various types of arrays, lists, maps, and strings.

56.3.1 3.1 Lists

Lists in Mu can be formed directly by enclosing a comma separated list of values in brackets:

```
[1, 2, 3, 4, 5]
```

Mu will infer the type of the list. The type of a list is expressed by enclosing a type name inside brackets. So the type of the above expression is [int] (a list of int). Mu does not have heterogeneous lists (without defining them yourself). So you can't do something list this:

```
[1, "two", 3.14] // syntax error
```

There are three operators on lists and some syntactic sugar. The operators are cons, head, and tail.

3.1.1 cons

`cons ( **[** 'a **]** ; 'a head, **[** 'a **]** list )` [Function]

Returns a new list with *head* as the first element followed by each of the elements in *list* .

The cons operation can be made by either using the function cons or using the operator :. The first argument (or left operand) is any value. The second argument (or right operand) must be a list of the first argument type. So for a type 'a as the first argument, the second argument must be `['a]` . For example the following are equivalent and result in the list `[1, 2, 3, 4]` .

```
1 : [2, 3, 4]
cos(1, [2, 3, 4])
```

3.1.2 head and tail

`head ( 'a; **[** 'a **]** list )` [Function]

Returns the first element in *list* .

`tail ( [ *'a* ] *;* [ *'a* ] list )` [Function]

Returns the list of all elements in *list except* the first element in the same order they appear in the *list* .

The head and tail functions return the first element in a list (head) or the rest of the list (tail). tail will always return a list or nil if there is no tail.

```
head([1, 2, 3])  1
tail([1, 2, 3])  [2, 3]
tail([1])  nil
```

3.1.3 Pattern Matching with Lists

The list syntax can also be used to pull apart the values in a list by using it in a pattern:

```
let [x, y] = [1, 2];
assert(x == 1);
assert(y == 2);
```

Note that the number of elements in the list must match the number of elements in the pattern. If they do not match an exception will be thrown at runtime (or at compile time if can be detected).

The cons operator : can also be used in patterns. For example:

```
let h : t = [1, 2, 3]
```

The value of h will be 1, the value of t will be the list [2, 3]. This is equivalent to doing the following:

```
  let x = [1, 2, 3],
      h = head(x),
      t = tail(x);
```

The cons pattern will throw if the list on the right-hand-side has the value nil.

56.3.2 3.2 Tuples

Tuples are collections of a fixed number of heterogeneous types. Another way to think of them is as anonymous structures.

Tuples have a special syntax used for construction and destruction (pattern matching) which make them easy to use. To construct a tuple of two or more values, simple enclose them in parenthesis like so:

```
(1, "the number one")
```

Similarily, the type of the above expression is (int,string). So tuple type declarations look similar to tuple values:

```
(int,string) x = (1, "the number one");
```

Tuple values (and tuple types) can be nested as well:

```
((int,string),float) x = ((1, "the number one"), 3.141529);
```

There is one exceptional circumstance with tuple values: if one of the elements is nil, the tuple type will be ill-defined and will produce a syntax error:

```
(1, nil) // error
```

There is currently no way to make a *singleton* tuple value (a tuple of one element).

3.2.1 Indexing a Particular Element in a Tuple

Each element in a tuple is given a numerical name. The first element is called _0 followed by _1 followed by _2, and so on. For example to get the second element:

```
(1, "two")._0  1
(1, "two")._1  "two"
```

The value of the first expression is 1 because the 0th element is begin indexed. It is not possible to dynamically index tuple elements because each element may have a unique type. The best way to think about a tuple is a struct with elements named _0 through _N that you don't need to declare.

3.2.2 Pattern Matching with Tuples

Usually you don't bother with the type annotation and use pattern matching instead:

```
 let x = (1, "the number one");
```

x in this example has type (int,string). The tuple syntax can also be used when pattern matching to pull values out of the tuple (destruction):

```
 let (a, b) = (1, "the number one"); \
```

So in this case a will have the value 1 and be of type int and b will have the value "the number one" and be of type string. Tuple patterns may also be nested:

```
 let (a, (b, c)) = (1.234, (2, "three")); \
```

The tuple pattern will throw if the matching tuple value is nil.

56.3.3 3.3 Dynamic Arrays

Mu has two different kinds of arrays: dynamic and fixed. Unlike C, the type specifier completely encodes the array type. For example:

```
float[] foo;
float foo[]; // syntax error
```

The first line makes a dynamic array of floats. The second line produces a syntax error in Mu because the C array syntax is not permitted. The dynamic array object has a number of member functions on it which can be called:

```
foo.size(); // returns number of elements in foo foo.push_back(1.0); // appends the␣
↪number 1.0 to the end of the array foo.front(); // returns a reference to the first␣
↪element in foo foo.back(); // returns a reference to the last element in foo.clear(); /
↪/ sets the size to 0
foo.resize(10); // resizes foo to 10 elements.
foo.rest(); // returns all but the first element as an array foo[0]; // returns the 0th␣
```

(continues on next page)

(continued from previous page)

```
↪element
```

The array type declarations can be nested:

```
float[][] foo; // array of array of floats foo.push_back( float[]() ); // add a float[]␣
↪to foo foo[0].push_back(1.0); // add a float to the 0th element of foo
```

Note that this is not the same as making a multidimensional array. You cannot currently make a multidimensional dynamic array. However you can make a multidimensional fixed array.

56.3.4 3.4 Fixed Size Arrays

Fixed arrays are similar to dynamic arrays, but the size is encoded in the type:

```
float[10] foo; // make a 10 element fixed array of floats
```

The fixed array types have fewer operations than the dynamic array type (since they are a more restrictive type). However the basic array syntax is shared:

```
foo.size(); // returns size (ok) \
foo.push_back(1.0); // syntax error! (no push_back function)
float x = foo[5]; // 5th element
foo[5] = x;
```

Like dynamic arrays, you can make arrays of arrays and mix them with dynamic arrays:

```
float[10][4] foo; // 4 arrays of 10 floats
float[10][] foo; // a dynamic array of 10 float fixed arrays
```

In addition, fixed arrays may have multiple dimensions. This is done by adding comma separated lists of sizes:

```
float[4,4] foo; // a 4x4 matrix of floats \
float x = foo[1,1]; // set x to 1st row 1st column value foo[3,1] = x; // copy x to the␣
↪3rd row 1st column value
float[4,4,4] foo; // a rank-3 tensor
```

For graphics applications, its common to

56.3.5 3.5 Maps

56.3.6 3.6 Sets

56.3.7 3.7 String Type.

Strings conceptually contain a sequence of char type. However, the string type is currently independant of any sequence type. Strings in Mu are immutable; you cannot change the value of a string. Instead you make new strings by either concatenating existing strings or by using the formatting operator (%).

`strings` can be constructed as a constant from a sequence of characters in double or triple quotes. (See String Constants for more details).

```
string s = "Hello World";
string y = """A long string with
possible newlines embedded "quotes" and other
difficult characters""";
```

There are a few basic operations on strings:

hash size split substr

In addition, any value can be cast to a string (or rather a string can be constructed from any value).

3.7.1 Formatting

Strings can be formatted using the % operator. The left-hand-side of the operator is a format template string and the right side is a single value or a tuple of values that are substituted into the template much like C's printf function.

For example:

```
"%0.2f" % math.pi  "3.14"
"%d" % 123  "123"
"%s and %d" % ("foo", 123)  "foo and 123"
```

3.7.1.1 Formatting Directives

56.3.8 3.8 Regular Expressions and Syntax

56.3.9 3.9 Types Defined by Applications or Native Modules

Some applications or native modules may provide an opaque type. These are types that appear to have no structure from Mu, but have operators and functions which can operate on them. There is no way to define an opaque type in Mu, but a similar effect can be had by defining an enumeration as a union.

An example of an opaque type can be found in the system module. This module contains analogues to POSIX functions and types in Mu. One of the declared types FILE represents a standard C library file descriptor. This type cannot be deconstructed but can be operated on by fopen, fclose, fread, and fwrite among others. Opaque types often represent external resources like FILE.

56.4 4. User Defined Types: Structs, Classes, and Unions

56.4.1 4.1 Records

56.4.2 4.2 Object Oriented Features: Classes

56.4.3 4.3 Tagged Union Type

Mu has a type-safe data structure called a *union* which makes it possible to deal with values from more than one type. The implementation of a union in Mu is different than languages like C and C++ which give unrestricted access to values.

Unions declarations have the following syntax:

```
union: union-name { cnstr1 [ type1 ] [ | cnstr2 [ type2 ] ]* }
```

Here's a specific declaration:

```
union: Foo { A int | B float | C string }
```

The example declares a union called Foo which has three constructors: A, B, and C. Each constructor is separated by the | character. You could read the declaration as "Foo is an int which we'll call A *or* a float which we'll call B *or* a string which we'll call C". So, the constructor A when called as a function takes an int, B takes a float, and C takes a string.

From a theoretical perspective, a union can be thought of as a type which includes the values of other types plus a *tag* for each value. In order to retrieve the value of a union, the tag must be supplied which matches the tag on the value. In other words, to get a value from a union you need to know what its tag is. This type of union is called a *tagged union* as opposed to the C-like *untagged union* where only the value is stored.

In Mu, the union includes multiple *constuctors* (which are similar to class constructors) which are used as tags. Each constructor in the union must have a unique name. The constructor has a type associated with it; values from that type are stored in the union with the constructor tag. To make a union object you must call one of its constructors.

Continuing the above example of Foo, we can use the constructors to make instances of Foo like this:

```
Foo x = Foo.A(123);
Foo y = Foo.B(3.141529);
Foo z = Foo.C("hello");
```

Or to make it easier to get at the constructors use the type:

```
use Foo;
Foo x = A(123);
Foo y = B(3.141529);
Foo z = C("hello");
```

In any case, x, y, and z are all of type Foo. The arguments and overloading of the con- structors *are the same as the underlying type's* . So, for example the string has constructors like these:

```
string(1)  "1"
string(1.234)  "1.234"
```

which means the Foo.C() can be called similarily because its underlying type is string:

```
Foo x = C(1);  x = Foo.C("1")
Foo y = C(1.234);  y = Foo.C("1.234")
```

So why not allow casting to unions? For example this seems like it would be ok:

```
Foo x = 1.123; error Can't cast to a union value
```

There are two reasons. The first is that constructor arguments for multiple constructors unions may be identical; in that case there is no obvious way to choose the proper one. The second is actually a feature of unions: you can declare multiple constructors that take the same type. So this is ok:

```
union Bar { A int | B int | C int } \
use Bar;
Bar x = A(1);

let C q = x; error *x's value was not constructed with C!*
let A w = x;  *ok! w = 1*
```

Bar has three constructors and the all take int. So the union values are really the values of the int type alone. However, the union constructor (tag) is still required to get the value. You could think of the union in this case making *flavors* of int.

4.3.1 Retrieving union values.

To retrieve a union value, you must use some form of pattern matching. There are no fields of a union as there are in the C languages. For example, to get the int out of the Foo union declared above:

```
Foo x = Foo.A(123);
int Foo.A i = x;  i = 123
```

If the pattern does not match (which means the value in the union could not have been created with the pattern constructor) an exception will be thrown:

```
Foo x = Foo.A(123); \
let Foo.B f = x; error throws: x will only match constructor A
```

This form of pattern matching used for union *deconstruction* is only used in cases where the type of the union's value is known ahead of time (so that the pattern will not fail). When type is not known, the case statement or expression is used instead. The case pattern syntax allows the use of union constructors like let.

```
Foo x = Foo.A(123); \
case (x)
```

(continues on next page)

(continued from previous page)

```
{
   A i -> { print("x's value is the int %d\n" % i); }
   B f -> { print("x's value is the float %f\n" % f); }
   C s -> { print("x's value is the string %s\n" % s); }
}

x's value is the int 123
```

Note that we didn't have a use statement in the last example: in a case statement where the value being matched against is a union, the union's type is automatically being *used* inside of it. So we don't have to prefix the constructor pattern with the union name.

Enumerations as Union Constructors

There is a special case of a union constructor which has the void type. To declare such a constructor simply omit its type. While the union may have no retrievable value of that constructor, it will still be *tagged* as such. Using this mechansim enumerated types can be created where the constructor **is** the value.

```
 union: Weekday { Monday | Tuesday | Wednesday | Thrusday | Friday } \
```

These constructors take no arguments, and there is an exception to the usual syntax for functions in the case of these constructors: no parenthesis are needed to call them.

```
 Weekday yesterday = Weekday.Monday; \

use Weekday;
Weekday today = Tuesday;
```

You cannot use let to match against enumerated unions. Only the case statement (or expression) can be used:

```
case (today) \
{
   Monday -> { ... }
   Tuesday -> { ... }
   Wednesday -> { ... }
   Thursday -> { ... }
   Friday -> { ... }
}
```

56.5 5. Functions and the Function Type.

Functions play a ubiquitous role in Mu. Constructors, operators, iteration constructs, and more are implemented as functions which can be overriden, passed as objects, or modified.

56.5.1 5.1 Function Declaration

Mu functions are declared using either the function: keyword (aka :) or the operator: keyword. When binding a function to a symbol, the form is:

```
function: identifier signature body
```

The *identifier* can be any legal identifier (with some restrictions). The *signature* portion of the declaration looks like this:

```
( return-type ; type0 arg0, type1 arg1, type2 arg2, ... )
```

This object, a signature, indicates the return type and all of the arguments to the function. The first part of the signature, the *return-type* must always be present. The second part, the argument list, can be empty indicating that there are no arguments to the function. The last portion the *body* is a code block. Here's an example function that computes factorial:

```
function: factorial(int; int x)
{
return x == if 1 then 1 else x * factorial(x-1);
}
```

56.5.2 5.2 Operator Declaration

Mu allows operator declaration and overloading. In Mu operators are just functions with syntactic sugar. The operator precedence cannot be changed, but any operator can be overloaded. The operator declaration syntax is similar to the function declaration syntax:

```
operator: operator-token signature body
```

The *operator-token* is a special sequence of characters that describes the operator. In most cases this is the same as the operator in use. For example, the operator ^ is not defined for floating point numbers. But you can make it into a power operator:

```
operator: ^ (float; float base, int power)
{
   float answer = base;

   for (int i=0; i < power - 1; i++)
   {
     answer *= base;
   }

```

(continues on next page)

(continued from previous page)

```
   return answer;
}
```

Some operators are overloaded. In order to distinguish between the overloaded versions, the *operator-token* is different the literal operator token. For example the postfix increment

operator var++ is different than the prefix increment operator ++var. Each has a different special token as show below.

This is the current list of special operator tokens:

```
_++ postfix increment
_-- postfix decrement
++_ prefix increment
--_ prefix decrement
?: conditional expression (takes three arguments)
```

56.5.3 5.3 Function Overloading

Functions may be overloaded. In other words, multiple declarations of the same function may be made as long as their arguments differ in type or number. For example:

```
function: area (float; triangle t) { ... }
function: area (float; circle t) { ... }
function: area (float; rectangle t) { ... }
function: area (float; shape t) { ... }
```

When the area function is applied to a circle, Mu will choose the appropriate version of the area function. In the above example, assuming that triangle, circle, and rectangle are all derived from the type shape, you can apply the area function to any other object that is derived from shape as well (Mu will choose the last area function above). When a direct match for an overloaded function is not found, Mu will attempt to use casting rules to make the function call.

56.5.4 5.4 Default values

Functions may have default values as long as every parameter after a parameter with a default value also has a default value. (This is the same rule that C++ has concerning default parameter values.)

```
function: root (float; float a, float b = 2)
{
return math.pow(a, 1.0 / b);
}
```

The function "root" can be called either like this:

```
 root(2.0, 2.0);
```

or this:

```
root(2.0);
```

either way will return the value of math.pow(2, 0.5). It is an error to declare a function like hits:

```
function: root (float; float a = 2, float b)
{
// ...
}
```

In this case parameter “b” follows parameter “a” which has a default value. Because “a” was declared with a default value, “b” must also be declared with a default value.

56.5.5 5.5 Returning from a Function.

There are two ways to return a value from a function. The first way is identical to C/C++: use the return statement:

```
return return-expression
```

The *return-expression* must be the same as the function return type or something that can be cast to it automatically. The return statement may appear anywhere inside the function. There can be multiple returns from a function.

Alternately, since blocks of code are also expressions, you can omit the return statement.

```
function: add (int; int a, int b)
{
   a + b;
}
```

This is not a syntax error because the last statement of the code block returns an int and therefor becomes the return value of the function. This is unlike C/C++ which require the return statement exist unless the function returns void.

56.5.6 5.6 Unnamed (Anonymous) Functions

An anonymous function can be created by omitting the function name in a function defi- nition. For example, the good old add function as an anonymous function looks like this when fed to the interpreter:

```
mu> function: (int; int a, int b) { a + b; }
(int;int,int)  lambda (int; int a, int b) (+ lambda.a lambda.b)
```

The result is a function object. You can assign this value to a variable if you like and call the function through it.

```
mu> global let x = function: (int; int a, int b) { a + b }
(int;int,int)&  lambda (int; int a, int b) (+ lambda.a lambda.b)
mu> x(1,2)
 int  3
```

Unambiguous function objects all have operator() defined for them. So you can call the function either through a variable or an expression that returns a function.

56.6 6. Constants and Initialization

56.6.1 6.1 Integral Type Constants

hex octal decimal int64 v int

56.6.2 6.2 Floating Point Constants

engineering notation

56.6.3 6.3 String Constant Syntax

single Quotes. unicode escapes.

56.6.4 6.4 String Constant Syntax

Double quotes. Unicode escapes. Control escapes. Triple quotes. String constant juxtapo- sition. Handling of newlines in strings.

56.6.5 6.5 USer and Built-in Type Constants

The reference types which have constructors can be initialized using curly brackets when assigned to a type annotated variable. This applies to all types not just collections. However, this syntax is most often used with collections.

Using arrays as an example:

```
float[4] foo = {1.0, 2.0, 3.0, 4.0};
```

Multidimensional arrays:

```
float[2,2] foo = {1.0, 2.0, 3.0, 4.0};
```

also have the form of single dimensional arrays. However arrays of arrays:

```
float[2][2] foo = { {1.0, 2.0}, {3.0, 4.0} };
```

use nested brackets. Similarily dynamic arrays can be initialized:

```
float[] foo = {1, 2, 4, 5, 6, 7};
```

trailing commas are accepted. You may also use non-constant values in an array con- struction:

```
float a = 5.0;
float[] foo = {a, a*2, a*4};
```

6.5.1 Constants

In addition, you can make a constant by putting the type in front of the braces. For example if there is a function bar() that takes a dynamic array of ints, you can supply a constant to it like this:

```
bar( int[] {1, 2, 3, 4} );
```

6.5.2 Use with Patterns

The initializer syntax will not work with patterns because the type of the expression is underconstrained:

```
let a = {1, 2, 3 }; // error
```

The left-hand-side of assignment is a pattern, and the right hand side has an unknown type. In this case it could be a list, a dynamic array, a fixed array or a struct or class. So the type of a is ill-defined. If in this case we meant a to be of type int[] for example, we could use the constant syntax and do the following:

```
let a = int[] {1, 2, 3}; // ok
```

This is well defined.

56.6.6 6.6 Converting Basic Constants

Mu recognizes some basic constant suffixes. More can be added. Suffixes are defined in the suffix module as normal functions. For example the function suffix.f is declared like this:

```
module: suffix
{
   function: f (float; int i) { i; } function: f (float; int64 i) { i; }
}
```

In use, the suffix would appear *after* a integral numeric constant such as 123f. The f suffix function is called by the parser to make a constant float out of the token 123. Of course you could also just write it 123.0 which would have the same affect.

A more useful example would be a regular expression suffix:

```
module: suffix
{
   function: r (regex; string s) { s; }
}
```

The function **r** could be used to create a list of regular expressions like so:

```
["foo.*"r, "bar.*"r, "[0-9]+"r]
```

As opposed to this:

```
[regex] {"foo.*", "bar.*", "[0-9]+" }
```

Or as a way to diambiguate overloaded functions:

```
\: foo (void; regex a) { ... }
\: foo (void; string a) { ... }

foo("[0-9]+"r); // calls first version
foo("[0-9]+"); // calls second version
```

For regular expressions, this has the added benefit of forcing the compilation of the regular expression during parsing instead of at runtime.

A more (possibly sinister) usage of suffixes is to represent mathematical or physical constants (or units) using them:

```
module: suffix
{
   function: pi (float; float x) { x * math.constants.pi; }
   function: pi (float; int x) { x * math.constants.pi; }
   function: i (complex float; float x) { complex float(0,x); }
}

let c = 6 + 12i; // alternate complex number constant form!
let twoPI = 2 pi; // questionable use of suffix!
```

56.6.7 6.7 Anonymous Function Constants

An anonymous function can be created by omitting the function name in a function defini- tion.

For example, the good old add function as an anonymous function looks like this when fed to the interpreter:

```
mu> function: (int; int a, int b) { a + b; }
(int;int,int) => lambda (int; int a, int b) (+ lambda.a lambda.b)
```

The result is a function object. You can assign this value to a variable if you like and call the function through it.

```
mu> global let x = function: (int; int a, int b) { a + b }
(int;int,int)& => lambda (int; int a, int b) (+ lambda.a lambda.b)
mu> x(1,2)
int => 3
```

Unambiguous function objects all have operator() defined for them. So you can call the function either through a variable or an expression that returns a function.

56.7 7. Polymorphic and Parameterized Types

56.7.1 7.1 Type Variables

56.7.2 7.2 Typeclasses: Families of Types.

56.8 8. Variables

Mu types have two kinds of semantics: reference and value. Types with value semantics are always copied when assigned or passed to a function as an argument. Basic number types all have value semantics.

56.8.1 8.1 Reference Type Construction Semantics

Reference types are allocated by the programmer or the runtime environment. For the programmer, this is done by calling one of the constructors for the type. For example, you can create a string object like this:

A variable that holds a reference types is actually better defined as a variant type; the value can be a member of the storage type or it can be nil.

```
string foo = string(10);
string bar = string();
```

which creates the string "10" from the integer 10 for the variable foo and the default empty string for the variable bar. When a variable is declared without an initializer, Mu will supply the default initializer or nil if there is none:

```
string baz; // these are the same
string baz = string(); //
```

If you explicitly want to set an object type variable to nil, you should do that when intializing it:

```
string foo = nil; // no object is constructed
```

This is not true of arrays however:

```
string[] bar; // creates string[] object
bar.resize(1);
bar[0].size(); // whoops! that element is nil!
bar[0] = string(); // no longer nil
```

Because objects are passed by reference, setting one variable to another can have inter- esting results:

```
string foo = "ABC";
string bar = foo;
bar += "DEF";
print( foo + "\n" );
```

The output of the above example is “ABCDEF”. Because the variables foo and bar are references to the same underlying string object, mutating operations through the bar variable will be visible through the foo variable. You can test for this condition using the operator eq:

```
if (foo eq bar) { print("they’re the same!\n"); }
```

If you want to prevent this behavior, you need to make a copy:

```
string foo = "ABC";
string bar = string(foo);
bar += "DEF";
print( foo + "\n" );
```

In this case the string “ABC” will be printed because foo is referencing a different object than bar.

56.8.2 8.2 Kinds of Variables.

There are three kinds of “variables” in a Mu: global, local, and fields.

Global variables can be declared in any scope by preceding the variable declaration with the global keyword. There is a single location in the Mu process which represents a global variable and its lifetime is the same as the process.

The scope of the global variable’s symbol declaration determines how and where the global variable can be accessed just like any other symbol. For example to make a global variable that is also globally accessable requires declaring in the top-most scope or a scope accessable from the top-most scope. For example all of the following are global variables that are accessable from any part of a Mu program:

```
global int x;
module: amodule { global int y; }

function: foo (void;) { print(x); } // ok
function: bar (void;) { print(amodule.y); } // ok
```

However, if the variable is declared in a function scope, then it is only accessable by parts of the code which can access symbols at the same scope:

```
function: foo (void;) { global int x; }
print(x); // error! can’t access it

function: bar (void;)
{
  global int x;
  function: foo (void;) {
  print(x); } // ok print(x); // ok
}
```

Local variables exist on the program stack and exist only while the declaration scope is active. In the case of a function, this is while the function is executing. The variable has a unique copy for each function invocation just like a function’s parameters.

Local variables are accessable on in the scope in which they are declared:

```
int x;
print(x); // ok
function: foo (void;) { print(x); } // error function: bar (void;)
{
  int x;
  print(x); // ok
}
```

Fields are variables that are part of a larger object. For example, the two dimensional vector type has three field variables called "x" and "y". These variables are addressable using the scope (dot) notation:

```
vector float[2] v;
v.x = 1;
v.y = 2;
```

The lifetime of a field is the same as the lifetime of the object (its scope).

56.8.3 8.3 Variable Declarations

Mu is a statically typed language. It currently does very primitive inferencing and therefor requires type annotation for variables in many cases. Variable symbols are assigned a type which is immutable – this is called its storage type. The variable will only ever hold values of its storage type.

The annotated variable declaration statement has the one of following forms:

```
 type variable [, variable ...]
type variable = expression [, variable = expression, ...]
```

In usage this looks like this:

```
int x; int y = 1;
int z = 1, q = 10;
```

56.9 9. Pattern Matching

Besides the usual imperative variable assignment syntax, the let statement can be used to match patterns. There are two benefits to using let with pattern matching: multiple nested assignments can be made simultaneously and the types of the variables can be figured out by the parser so you don't have to annotate them.

Similarily, the case statement and expression can match patterns and values while as- signing symbols to values without type annotation.

The ability of the parser to figure out types of symbols with annotation is called type inference. This is a common feature of functional languages in the ML family (O'CaML and Haslkell for example). Mu currently uses a very restricted type inference algorithm.

56.9.1 9.1 Assigning Variables with Patterns

The let statement defines symbols in the current and nested scopes. Symbols assigned using

let are immutable (the values cannot be changed). The general form of the statement is:

```
let let-pattern1 = expression1 \
[, let-pattern2 = expression2, ...] ;
```

The *let-pattern* can be any of the following:

- A single symbol which is assigned the value of its expression. This is the simplest usage of let and will never result in an exception being thrown. For example let x = 0 assigns the type int to the symbol x and causes the value 0 to be bound to it.
- A tuple destructor pattern which binds the pattern symbols to each of the elements of a tuple expression. You can only use the tuple pattern if the bound expression is a tuple type. In addition the number of elements in the pattern must match the number of elements in the expression tuple type.

```
 let (a, b, c) = (1, 2, 3);
     a = 1, b = 2, c = 3
```

- A list destructor pattern which bind the pattern symbols to each of the elements of a list expression. Like the tuple pattern, the list pattern requires that the number of symbols in the pattern be equal to the number of elements in the list value at runtime. If the number of elements does not match at runtime or the parser can figure out that the number does not match a parse time, an exception will be thrown.

```
let x = [1, 2, 3]; // without list pattern \
     x = [1, 2, 3]
let [a, b, c] = [1, 2, 3];
     a = 1, b = 2, c = 3
let [d, e] = [6, 7, 8];
    error num symbols != num elements
```

- A cons pattern which pulls apart the head and tail of a list. This pattern, like the list destructor pattern, requires a list expression. Two symbols are supplied: the head and tail symbol:

```
let h : t = [1, 2, 3, 4]; \
     h = 1, t = [2, 3, 4]
let h0 : h1 : t = [1, 2, 3];
     h0 = 1, h1 = 2, t = [3]
```

- The structure destructor pattern which pulls apart structure fields. This pattern can also be used on tuples and lists, but cannot be used on arrays.

```
struct: Foo { int a; float b; } \
let {a, b} = Foo(1, 2.0);
     a = 1, b = 2.0
```

A union type constructor pattern. The expression supplied must be of the union type. If the constructor does not match the value constructor type, an exception will be thrown:

```
 union: Bar { A int | B float } let Bar.A {x} = Bar.A(1);
     x = 1

let Bar.B {x} = Bar.A(1)
    error Expression is of type Bar.A
```

- Finally, the symbol _ (underscore) can be bound to any expression. The expression is evaluated at runtime (if it is not pure), but the value is ignored.

Each of these patterns can be combined with the others and nested to arbitrary depths. This makes it easy to pull values from inside nested structs, tuples, and lists easily.

```
struct: Foo { int a; [int] b; } \
let ([a, b, c], d, {e, f : g}) = ([1,2,3], 4, Foo(5,[6,7,8]));
     a  =  1, b  =  2, c  =  3, d  =  4, e  =  5, f =  6, g  = [7, 8]

let (a, b, _) = (1, 2, [3,4,5,6]);
     a = 1, b = 2
```

56.9.2 9.2 Pattern Matching to Control Flow

56.9.3 9.3 Case an an Expression

56.10 10. Flow Control

Mu has the usual cadre of flow control statements for a C-family language plus a few additions.

All of the flow control statements are actually functions and they all return a value which depends on the statements they execute. Usually the return value is only useful in the context of a function definition where the last statement in the function is one of the flow control constructs.

56.10.1 10.1 Conditional Execution, the if Statement

The `if` statement has two forms: one with an else clause and one without. The test expression for an `if` statement must be of type bool. The general forms are:

```
if ( bool-expression ) if-true-statement
if ( bool-expression ) if-true-statement else if-false-statement
```

The return value of the if construct function is the return value of the *if-true-statement* or the *if-false-statement* . When there is an *if-false-statement* present, it must return the same type as the *if-true-statement* .

Because the integral and float types in Mu do not cast to type bool by default, many C/C++ idioms involving the if statement do not work:

```
int x = 1;
if (x) doit(); // error, x is not bool
if (x == 1) doit(); // ok
```

Some more examples:

```
if (x == 1) \
{
  doit();
}
else
{
  stopit();
}
```

Also note that the C++ ability to declare variables in the test expression are not valid in Mu:

```
 if (bool foo = testfunc()) doit(); // error
```

Since the test expression must be of type bool, the C++ism would not be very useful in Mu.

10.6 `case` Statements.

The general form is:

```
case ( value-expression )
{
  case-pattern0 -> { statement-block0 }
  case-pattern1 -> { statement-block1 }
  case-pattern2 -> { statement-block2 }
  ...
}
```

Unlike the `if` statement, the case statement can match over any value type that can appear in a pattern.

The first matching pattern will cause the corresponding statement to be executed. A simple example matching values might be:

```
string s = ...; case (s)

{
  "one" -> { print(1); }
  "two" -> { print(2); }
  "three" -> { print(3); }
  x -> { print("Don't know what %s is\n" % x); }
}
```

56.10.2 10.3 Conditional Expressions; A Variation on if

The condition expressions in Mu looks similar to the conditional statements.

10.3.1 if-then-else Expression

In the case of the `if` conditional expression the keywords then and else are required. No parenthesis or brackets are required:

```
if test-expression then if-true-expression else if-false-expression
```

The *if-true-expression* is only evaluated if *test-expression* evualuates to true. Otherwise, the *if-false-expression* is evaluated. In either case, the value and type of the conditional expression is the same as the evaluated expression.

The type of the *if-true-expression* and the *if-false-expression* must be the same or case to the type. If the types differ, the parser will attempt to find the least lossy type cast that will make them match.

The conditional expression can be used anywhere any other expression can be used, but because of its precedence the entire expression may require parenthesis':

```
let x = if somecondition() then 0.0 else 3.1415; \
print(if x == 3.1415 then "its pi" else "its not pi");
let y = x + (if x > 0.0 then 1.0 else -1.0);
```

10.3.2 case Expression

```
case test-expression of pattern1 -> expression1
                        pattern2  -> expression2
                        pattern3  -> expression3
                        ...
```

56.10.3 10.4 Fixed Number of Iterations Looping.

The repeat statement has the general form:

```
repeat ( int-expression ) statement
```

The *int-expression* is evaluated once. *Statement* is then evaluated that many times.

```
// outputs "hello hello hello " \
repeat (3) print("hello ");
```

56.10.4 10.5 Simple Looping

The `while` and `do-while` constructs have the form:

```
while ( test-expression ) statement
do statement while ( test-expression ) ;
```

The *test-expression* must be of type bool. It is evaluted either before *statement* (or after in the case with do-while). If *test-expression* is true, then *statement* will be evaluated.

In the cast of do-while, the *statement* is always evaluated once since the test expression is evaluated after.

56.10.5 10.6 Generalized for Loop

The for construct in Mu is similar to the C/C++ statement of the same name:

```
 for ( declaration-expr; test-expr; tail-expr ) statement
```

The *declaration-expr* , *test-expr* , and *tail-expr* are all optional. If the *test-expr* does not exist, it is as if the *test-expr* where true (the loop never terminates).

Test-expr must be of type bool.

As in C++, the *declaration-expr* can declare one or more variables:

```
// outputs: "0 1 2 "
for (int i = 0; i < 3; i++) print(i + " ");
```

In this case the variable i is in the scope of statement and does not appear outside the loop:

```
// outputs: "0 1 2 "
for (int i = 0; i < 3; i++) print(i + " ");

print(i + "\n"); // error "i" not defined here.
```

56.10.6 10.7 Operating on Every Collection Element

The `for_each` construct has the general form:

```
 for_each ( identifier ; collection ) statement
```

Currently, *collection* can only be a dynamic or fixed array or a list. The *identifier* is bound to each element of *collection* then the *statement* is evaluated.

```
// Outputs: "1 2 3 " \
int[] nums = {1, 2, 3};
for_each (x; nums) print(x + " ");
```

56.10.7 10.8 Iteration on Collections over Indices

The `for_index` construct has the general form:

```
for_index ( identifier [, idenitifer, ...] ; collection ) statement
```

Currently, *collection* can be a dynamic or fixed array. The *identifier* is bound to each element of *collection* then the *statement* is evaluated. The number of identifiers must be the dimension of the collection.

```
// Outputs: "1 2 3 " \
int[] nums = {1, 2, 3};
for_index (i; nums) print(nums[i] + " ");
```

For multidimensional arrays:

```
// Transpose a matrix \
float[4,4] M = ...;
float[4,4] T;
for_index (i, j; M) T[i,j] = M[j,i];
```

56.10.8 10.9 Break and Continue: Short Circuiting Loops.

All of the looping constructs can be "short-circuited" using `break` and `continue` . These function just like their C/C++ conterparts. break and continue will terminate the inner most loop.

```
while (true)
{
  string x = doit();
  if (x == "stop it") break;
}
```

In the example above, the `while` loop will never terminate if `doit()` does not return "stop it".

If the loop as a *test-expression* , then continue can be used to cause the flow of control to skip the rest of the *statement* being iterated. In the case of a for loop, this will cause execution of the *tail-expr* before the *test-expr* .

```
// outputs: "1 2 "
for (int i=0; i < 10; i++)
{
  if (i > 2) continue;
  print(i + " ");
}
```

56.10.9 10.10 The throw statement

The `throw` statement, which raises and exception, has two forms:

```
throw throw-expression
throw
```

throw-expression can be of any reference type. So you cannot throw an `int` or `float` for example because these types are value types.

The second form is applicable only inside a the catch clause of a try-catch statement. In that context, it rethrows the last thrown value.

56.10.10 10.11 The try-catch statement

The `try-catch` form is:

```
try
{
  *try-statements*
}
catch ( *catch-expression-1* )
{
  *catch-statements*
}
catch ( *catch-expression-2* )
{
  *catch-statements*
}
...
catch ( *catch-expression-N* )
{
  *catch-statements*
}
catch (...)
{
  *default-catch-statements*
}
```

The try section and one of the catch sections are mandatory. The use of a default catch statement — one whose *catch-expression* is ... — is optional.

The statements in *try-statements* are evaluated. If during the course of evaluation an exception is raised by a throw statement, control will be returned to the try-catch state- ment. At that point a type match is attempted between the object thrown and each of the catch clauses of the try-catch statement. The first catch clause that matches has its *catch-statements* evaluated. The default catch statement — which has ... as its *catch- expression* will catch any type.

An example best explains it:

```
try \
{
  throw "no good!";
```

(continues on next page)

(continued from previous page)

```
}
catch (string s)
{
  print("caught " + s + "\n");
}
```

In this case the value "no good!" will be caught by the catch clause. The *catch-expression* in this case declares a varaible which is then assigned the thrown value. In the case of the default catch, you do not have access to the value:

```
try
{
  throw "no good!";
}
catch (...)
{
  print("caught something!");
}
```

There is no way to identify the caught type, but clean up can occur. If you need to rethrow the value you can use throw with no arguments:

```
try
{
  throw "no good!";
}
catch (...)
{
  print("caught something! rethrowing...");
  throw;
}
```

This works for any type of catch clause. In addition, you can throw a completely different object if you need to:

```
try
{
  throw "no good!";
}
catch (...)
{
  print("caught something!");
  throw "something else";
}
```

If no catch clause matches the thrown object type, its equivalent to a default catch that simply rethrows:

```
try
{
  something_that_throws();
}
```

(continues on next page)

(continued from previous page)

```
catch (...)
{
  throw;
}
```

56.10.11 10.12 The assert() Function.

Although the `assert()` function is not strictly part of Mu's exception handling, the built-in function does provide a helpful way to debug run-time problems.

`assert ( void; bool testexpr )` [Function]

testexpr is evaluated. If the value is true, nothing happens. If the value is false, an exception is raised. The exception object will contain a string representation of the *testexpr* which can be output.

For example the `mu` interpreter when present with this:

```
int x = 1;
assert(x == 2);
```

will produce this:

```
 ERROR: Uncaught Exception: Assertion failed: (== x 2).
```

Currently, assert will produce a lisp-like expression indicating the failed *testexpr* . Note that if partial evaluation of the *testexpr* occurs, you may get a surprising result:

```
mu> assert(1 == 2); \
ERROR: Uncaught Exception: Assertion failed: false.
```

In the above case the *testexpr* was partially evaluated to false early in the compilation process.

56.11 11. Namespace (Scoping) Rules

56.11.1 11.1 How Symbols are Assigned to Namespaces

When a symbol is declared (like a variable or a function) it is declared in an existing *namespace* . A namespace is itself a symbol. For example functions, types, and modules are all namespaces. The namespaces form a hierarchy; the root is called the *global namespace* . The global namespace has no name, but is pointed to by the global symbol root (which is of course in the global namespace!). Normally you don't need to access the global namespace by name.

Since every namespace is in the global namespace, we can form absolute or relative paths to symbols. This makes it possible to disambiguate two symbols with the same name, but that live in different namespaces:

```
module: Foo
{
  global int i;
```

(continues on next page)

(continued from previous page)

```
}
function: bar (void;)
{
  int i = 10;
  print("%d\n" % (i + Foo.i));
}
```

In this example, in function bar we need to add two symbols named i. In order to indicate which one we are refering to the path to the global variable Foo.i is specifically given.

56.11.2 11.2 Declaration Scopes and Rules

Each of the following is a namespace in which symbols can be declared:

- Modules. Modules are namespaces that can be accessed from any other namespace. Nested modules can access functions, variables, and types declared in the current mod- ule namespace or parent module namespaces without using a path to the symbol.
- Functions. Local variables, functions, modules, etc, that are declared inside a function body are only visible in the scope of that function or its child namespaces. It is not possible to reference a symbol inside a function (including its parameters) from outside of the function.
- Types. Fields of types are accessible to outside namespaces though the dot notation of objects. Global variables, functions, and types declared in the scope of a type are accessible from any namespace. Nested types follow the same scoping rules as nested modules (and can be intermixed with modules as well). This includes both record-like types (struct and class) as well as the union type.

56.11.3 11.3 Loading a Module

The `require` statement makes sure that a named module has been loaded. Code following a `require` statement is guaranteed to find symbols refered to in the specified module.

```
require module_name
```

56.11.4 11.4 Using a namespace

The use statement makes any accessible namespace visible *to the current scope* .

```
use *namespace*
```

Once the current scope ends, the namespace referenced by use is no longer visible. This is a convenience to make code less verbose:

```
module: Foo
{
  function: add (int; int a, int b) { a + b; }
}
```

(continues on next page)

(continued from previous page)

```
use Foo;
add(1,2);  3
```

In this case, the symbols in module Foo become visible to the current scope. So the function add can be directly refered to without calling it Foo.add. This also works for types:

```
class: Bar
{
  class: Thing { ... }
}

use Bar;
Thing x = ...;
```

Here the type Thing is directly visible becase Bar is being used. This also applies to any functions declared in Bar.

11.4.1 Using a Module Implies require

The use statement not only makes a namespace visible, but can find namespaces on the filesystem before doing so. In other words, use can also perform the same function as `require` :

```
use io;
fstream file = fstream("x.tif");
```

Here the io module may be loaded if it was not already required by some other part of the program. use will attempt to load a module if it cannot find any namespace with the same name as the argument. When applied to modules, use can be thought of as doing two things:

```
require *module_name*
use *module_name*
```

56.12 12. Symbol Aliasing

Aliasing allows you to assign an identifier as a stand-in for a symbol or constant expression. The scope an alias is the enclosing scope. An alias is a first class symbol, and can be referenced through the "." notation. Aliases are made using the binary infix ":=" operator. This operator creates an alias out of the name on the left hand side from the symbol or constant expression on the right hand side.

```
 alias_symbol := existing_symbol_or_expression;
```

56.12.1 12.1 Importing Symbols from Other Namespaces

Aliasing is primarily a syntactic convenience. The most obivious use of aliasing is to import symbols from another namespace. For example, if you are using the "sin" function from the math module a lot, but do not which to use any other symbol in the math modules, you could do this:

```
require math;
sin := math.sin;
```

This effectively imports only the sin function into the current scope. Similarily if there is a module which is buryed deep within other namespaces, you can pull the module name into the current scope like this:

```
require some.very.far.away.module;
module := some.very.far.away.module;
module.call_some_function();
```

Aliases can also be assigned to other aliases. So for example:

```
require math;
sin := math.sin;
cos := sin;
tan := cos;

sin(123.321) == tan(123.321); // eek! that's true!
```

This example "imports" the sin function into the current namespace and then maliciously calls it "cos" and "tan" as well. Chaos ensues.

56.12.2 12.2 Function Aliasing

You can alias function names. There are a couple instances where this becomes useful. The first is when a function name or a path to a function name becomes unweildy:

```
f := doSomeIncrediblyHeinousThingToAFloatingPointNumber;
float x = f(123.0);
```

In addition, you can use function aliasing to change a member function into a normal function:

```
append := float[].push_back;
float[] array;
append(array, 123.0);
```

In the above example, the push back() member function is aliased to the name "append". The member function can then be called as a normal function. Note that the aliasing does not resolve virtual functions, it grabs the exact function specified on the right hand side. In addition, when a function name is overloaded, the alias represents all the overloaded functions. In effect, the alias is overloaded the same way the function name is. This is particularly evident in type aliasing; the alias not only represents a type name, but all constuctors for the type as well.

56.12.3 12.3 Type Aliasing

Perhaps the most usefull form of aliasing is type aliasing. This is essentially the same as the C language "typedef" statement. In Mu, to create a type alias you assign a type name to an identifier:

```
Scary := float[][100][50,123][][];
```

you can then use the "Scary" symbol in place of its alias:

```
Scary foo = Scary();
```

This can be particularily useful in cases where the meaning of array types becomes messy. Here's a very contrived example

```
hpoint := float[4]; // a point is a homogeneous 4d object
cv := vector hpoint; // a cv is a vector point
Patch := cv[4,4]; // a Patch is a 4x4 array of cvs
Surface := Patch[,]; // a Surface is resizable 2D array of Patches
Model := Surface[]; // A Model is a collection of Surfaces
Scene := Model[]; // A Scene is a collection of Models
```

When compiling, Mu will substitute symbols for their aliases recursively until a type is found. In the above case, the Scene alias expands out to:

```
Scene := (vector float[4])[4,4][,][][];
```

The standard modules use type aliasing. The math module declares some type aliases for vector types:

```
math.vec4f := vector float[4];
math.vec3f := vector float[3];
math.vec2f := vector float[2];
```

56.12.4 12.4 Variable Aliasing

Finally, a less useful form of aliasing is varible name aliasing. Again, the syntax is simple assignment to an identifier:

```
float foo = 123.321;
bar := foo;
print(bar + "\n");
```

Note the similarity to the following:

```
float foo = 123.321;
float bar = foo;
print(bar + "\n");
```

Both examples produce the same output, but syntactically two very different things are happening. In the first example, an alias to the variable foo is created called bar. Bar is not a new float variable; it is a second name for the variable foo. In the second example, an actual location in memory is created called bar and that second variable is assigned the value of foo.

56.12.5 12.5 Symbolic Constants

The symbol aliasing syntax can also be used to declare a symbolic constant. This is roughly equivalent to a macro in C. The symbolic constant will always reduce to a constant expres- sion when used elsewhere.

```
pi := 3.14;
```

The symbol `pi` is not a variable and so cannot have its value changed.

56.13 13. Separate Parse and Compilation Modules

56.13.1 13.1 Module Definition

56.13.2 13.2 File System Locations

56.13.3 13.3 Module as a Unit of Compilation

56.13.4 13.4 Different Flavors of Module

56.13.5 13.5 Loading Modules at Runtime

56.14 14. Documenting Source Code

Mu has built-in syntax for annotating source code symbols like functions, variables, and type definitions. When parsed and compiled the documentation is assigned to a symbol and can be retrieved at runtime along with other symbol information. (See Runtime Module).

Modules like autodoc can convert the documentation into various formats like plain ASCII, HTML, or a TEX dialect. (See Autodoc Module).

56.14.1 14.1 Source Code Comments

Mu uses C++ comment syntax

A double slash // comments to the end of the line.

A slash followed by a star /* begins a comment that may include newlines. The comment is terminated by */.

56.14.2 14.2 Compiled Documentation

A documentation statement starts with the documentation: keyword followed by a string constant:

```
documenation: "documentation string goes here";
```

The parser will cache the documentation string until a symbol is defined in the same scope as the string. At the point, the parser will assign the documentation string to that symbol. If the scope changes, the documentation string may become orphaned.

An alternative more specific syntax makes it possible to specify the name of the next symbol to attach the string to:

```
documentation: foo "documentation string goes here";
```

In this case the next symbol defined as foo in the same scope as the documentation will be assigned the string. This makes it possible to string a number of documentation statements together before a compound definition (like a function) and document each symbol involved in the definition. This can be especially useful for documenting function parameters. For example:

```
 documentation: rotate
"""The rotate function transforms a vector about the origin returning the transformed␣
↪vector.""";

documentation: axis
"""The axis about which to rotate""";

documentation: radians
"""The amount to rotate about the axis in radians""";

documentation: v
"""The vector to rotate""";

\: rotate (vector float[3];
vector float[3] v, vector float[3] axis, vector float[3] radians)
{
..
}
```

56.14.3 14.3 Storage of Compiled Documentation

56.14.4 14.4 Documenation Syntax

56.15 15. Memory Management

Objects in Mu are automatically allocated and destroyed by the runtime environment. Objects created by the program, stack objects, and temporary objects in expressions are all handled in the same way. The Mu runtime environment uses a stanrdard mark-sweep garbage collection algorithm to find objects that are no longer being used and queues them up for finalization. When the objects are finalized, destructors are run and the object is reclaimed.

56.15.1 15.1 Allocation

There are three ways that objects are allocated: variable initialization, a constructor call, or as a temporary object during expression evaluation. When declaring a variable, the initialization either occurs directly or indirectly:

```
string foo = string(10); // foo = "10"
string bar; // bar = ""
```

By default, lack of an initialization expression causes Mu to supply the default construc- tor. In the above example, the "bar" variable is initialized like this:

```
 string bar = string();
```

Object variables are never set to nil unless you explicitly do so:

```
 string baz = nil;
```

The variable baz points to nothing. If you attempt to call a member function on baz, an exception will be thrown:

```
 baz.size(); // throws! baz == nil
```

56.15.2 15.2 Deallocation

Mu uses garbage collection to reclaim unused memory. There are never dangling references in a Mu program. When a certain amount of time has passed, or a certain number of objects have been allocated, the runtime environment may invoke the garbage collector. Obviously, this has consequences on program design. You cannot rely on objects being finalized at a particular point in a program.

The garbage collector can be tuned from the runtime module:

```
require runtime;
runtime.collect(); // invoke garbage collector manually
runtime.set_collection_threshold(1000); // set object overhead parameter
```

56.16 16. Closures and Partial Evaluation

56.16.1 16.1 Closures

Closures serve two purposes in Mu. One is to wrap variable references in nested functions. The other is to wrap some function arguments to mimic partial evaluation.

When nested functions are used,

```
function: foo (int; int a, int b)
{
  function: bar (int; int c) { a * c; }
```

(continues on next page)

(continued from previous page)

```
  bar(3.14) + b;
}
```

The foo function above will return `a * 3.14 + b` . The function bar is referring to a variable a in its scope. In essense, bar really has two arguments: c and another argument. Whenever bar is called, the extra argument's value is understood to be the value of the variable a. This is essentially syntactic sugar.

In this case, the closure is created whenever the function with hidden arguments is called. At the call point a closure object (which is a type of function object) is created that stores some of the parameter values. The closure object appears to be a function itself albeit with few arguments. The closure can be used anywhere a normal function object can be used.

In the second case, a closure can be directly created. For example let's say we defined a function that prints a message:

```
function: show_message (void; string msg)
{
  print("The message is: " + msg + "\n");
}
```

Now let's say there is a UI button object which will call a function with no arguments and no return value (type of (void;)). Since the show_message function has an argument of type string, it cannot be passed to the button:

```
button b = ...;
b.set_callback(show_message); // error wrong type
```

However, let's say we just want a message to be printed when the button is pressed. We could do this:

```
function: adapter (void;)
{
  show_message("my message");
}

b.set_callback(adapter); // ok
```

But we had to write an extra function. We could alternately use an anonymous function like this:

```
 b.set_callback(\: (void;) { show_message("my_message"); });
```

but that's not too much better. Finally, we can create a closure around the show_message function by only partially applying it:

```
 b.set_callback(closure(show_message, "my message")); // ok
```

In this case, the `closure()` function generated a closure object that appears as a function of type (void;).

56.16.2 16.2 Constant Expressions

Mu evaluates expressions very aggressively. If the compiler/interpreter can find that an expression is constant, it will evaluate it immediately. This may occur before an expression is even finished parsing.

This is not unlike C/C++ compilers evaluating constant expressions at compile time.

In addition, all functions are tagged with information about possible side effects or lack thereof. Because of this, even functions in modules can be evaluated early in order to fold constant values. Here's an example:

```
float f = math.acos(math.cos(math.pi) * -1.0);
```

the interpreter reports that this evaluates to the runtime expression:

```
float f = 0.0;
```

During parsing the interpreter determined that math.cos and math.acos are side effect free and therefor a constant argument will produce a constant return value. This is one form of partial evaluation.

56.16.3 16.3 Explicit Partial Application

Functions may be partially evaluated (applied) explicitly by using the empty argument syntax. This only applies to functions which take more than one argument.

The result of a partial function evaluation is a new function which will have the same return type as the partially evaluated function but a subset of its arguments. The following interactive session provides a basic example:

```
mu> \: add (int; int a, int b) { a + b }
mu> add(1,); // NOTE: missing arg
(int;int) => lambda (int; int) (+ 1 lambda.a)
mu> add(1,)(10);
float => 11.0f
mu> add(1,10)
float => 11.0f
```

Notice that when you call the function add with all its arguments it does what you expect: returns an `int` . When you call with *some* of its arguments it returns a new function.

56.16.4 16.4 Explicit Partial Evaluation

56.17 17. Phases

Mu has three phases: parse, compilation, and runtime evaluation.

56.17.1 17.1 Parse Phase

56.17.2 17.2 Compilation Phase

56.17.3 17.3 Runtime Phase

56.18 18. Muc: Mu Compiler

56.18.1 18.1 Invocation

56.18.2 18.2 Options

56.18.3 18.3 Muc Target

56.18.4 18.4 C++ Target

56.19 19. Standard Library

56.19.1 19.1 Built-in Functions

Printing Values String Formating size

56.19.2 19.2 Runtime Module

56.19.3 19.3 Autodoc Module

56.19.4 19.4 Math Modules

56.19.5 19.5 I/O Module

serialize

56.19.6 19.6 OpenGL Related Modules

56.19.7 19.7 POSIX Functions

56.19.8 19.8 Image I/O, Storage, and Operations

56.19.9 19.9 OpenCV Functions and Data Types

56.19.10 19.10 Read/Write GTO Files

56.20 20. Mu Compared to Similar Languages

56.20.1 20.1 C++

56.20.2 20.2 Python

56.20.3 20.3 JavaScript

56.20.4 20.4 ML Family of Languages

56.21 21. Example Usage

56.21.1 21.1 Embedding Mu in a C++ Application

56.21.2 21.2 Using Mu as a Shading Language

56.21.3 21.3 Using Mu to Control Particle Dynamics

56.21.4 21.4 Mu by Itself

56.22 Appendix A Reference

56.22.1 Properties

(Index is nonexistent)

56.22.2 Functions

56.22.3 A

assert

56.22.4 C

cons

56.22.5 H

head

56.22.6 T

tail

56.22.7 Types

(Index is nonexistent)

CHAPTER

FIFTYSEVEN

ALIGN START FRAMES PACKAGE (TO SLIDE SOURCES IN TIME SO THAT THEY CAN BE COMPARED)

This is an unsupported example that will probably be replaced with more complete tools in RV.

This package adds an "Align Start Frames" menu item to the the Edit menu. This action offsets the start frame of all the loaded movies and sequences so that they match the start frame of the first source (the first loaded sequence or movie).

As you may be aware, RV preserves the time information represented by frame numbers. This has the advantage of placing sequences properly in time according to frame numbers. So in RV if you have a version of a shot foo.12-120#.exr and another version with different handles or cut points, foo_v2.18-122#.exr adn you load these into RV to do a wipe compare, then the frames will line up correctly in time. However, if you load a movie file (a quicktime) then it will start at frame 1 regardless of the source frames it represents. This package will let you compare a movie with the frames it came from by sliding all sources in time to line up on the same frame.

Similar results can also be accomplished on the command line using the range offset flag:

```
rv [ foo.mov -ro 12 ] bar.12-120#.exr
```

CHAPTER

FIFTYEIGHT

RV/MAYA INTEGRATION (V1.4)

58.1 Requirements

The Maya integration package has only been tested with Maya 2012 and Maya 2014.

58.2 Installation

The first step on any platform is to activate the integration package. That is, go to the *Packages* tab of the RV Preferences dialog, and click the Load button next to “Maya Tools”, then restart RV.

58.2.1 Linux and Windows

On Linux and Windows, the only remaining task is to fill in the Application and Flags to run RV. In Maya open the Preferences, and go to the *Applications* section (titled “External Applications: Settings”). In the entry marked *Application Path for Viewing Sequences* enter the complete path to the *RVPUSH* executable. For example:

Linux

```
/usr/local/bin/rv/bin/rvpush
```

Windows

```
C:\Program Files (x86)\RV\bin\rvpush.exe
```

Then, in the entry marked *Optional Flags* , type this:

```
-tag playblast merge %f
```

58.2.2 OSX (Maya 2012 or Maya 2013)

On OSX, using Maya 2012 or Maya 2013, after turning on the Maya integration as described above and restarting RV, you need to install a tiny MEL script to handle the Maya side of the playblasting. In RV, go to the Maya menu, and select *Install Maya Support File.* (The file is installed in `Library/Preferences/Autodesk/maya/scripts` .)

Then, in Maya open the Preferences, and go to the *Applications* section (titled “External Applications: Settings”). In the entry marked *Application Path for Viewing Sequences* enter the Following:

```
playblastWithRV
```

Then, in the entry marked *Optional Flags* , type this:

```
%f %r
```

Done!

58.2.3 OSX (Maya 2014)

On OSX, using Maya 2014, after turning on the Maya integration as described above, and quitting RV, in Maya open the Preferences, go to the *Applications* section (titled "External Applications: Settings"). In the section marked *Sequence Viewing Applications* there are three applications, each with two entries, one for the application, and one for *Optional Flags* . In each application entry (assuming you've installed RV in the usual location), enter the following:

```
unset QT_MAC_NO_NATIVE_MENUBAR; /Applications/RV64.app/Contents/MacOS/rvpush
```

Then, in each entry marked *Optional Flags* , type this:

```
-tag playblast merge [ %f -fps %r ]
```

Done!

58.3 Build a Session in Open RV

The main point to remember about playblasting to RV is that if you leave RV open, each successive playblast will be merged into the RV Session, so you can easily compare the current render with others in the session.

Open the Session Manager (from the *Tools* menu, or by hitting the *x* key) to see a list your playblasts and view them in different ways. Each playblast will have a name provided by Maya, plus a timestamp that RV adds to help you track them. If you want to rename a playblast to something more meaningful, just select it in the Session Manager, and select the *Edit View Info* button from the right-click menu, or hit the button with the *I* on it.

At any point, you can save your Session to a Session File, so that you can easily pick up where you left off. When you return, render one playblast to get RV going, then *File→Merge* to bring in the Session File from the previous session.

58.4 Organizing and Comparing your Renders

By default, each new playblast that Maya adds to your session will become the current view. Note that you can easily switch back and forth between this Render and the previous one with the *shift-left/right-arrow* keys.

The "default" views (see the Session Manager) also let you easily see all your playblasts in a Sequence, in a Layout, or arranged in a Stack (note that you can use the "()" keys to easily cycle the stack, or you can use drag-and-drop in the Inputs section of the Session Manger to easily rearrange a group view.

If you'd like to "stick" to the view you've chosen, instead of switching to each new playblast as it arrives, toggle the *View Latest Playblast* option off on the *Maya* menu in RV.

58.5 Creating New Views

All the usual options in the Session Manager for creating and managing new views are available for managing your playblasts, but there are also a couple of hand shortcuts on the *Maya* menu.

Wipe Selected Playblasts

Select two or more playblasts in the Session Manager and hit this item to arrange them in a Stack, with Wipes mode activated, so that you can grab the edges of the images and slide them over the lower images. Remember that the "()" keys will cycle the stack order, and you can also drag and drop in the Inputs section of the Session Manager to reorder the stack.

Tile Selected Playblasts

Select two or more playblasts in the Session Manager and hit this item to arrange them in a Tiled Layout.

58.6 Rendering into Context

Don't forget that you can also bring in supplementary media to compare or use as reference for your animation. So for example you might bring in takes of the shots on either side of the one you're animating, and assemble a 3-shot sequence with one of your playblasts in the middle. But as you work, you'd like that middle shot to be replaced by newer playblasts. We call that "rendering into context".

In general, to prepare to render into context, after you setup your views, you'd

1. Turn off the *View Latest Playblast* option in the *Maya* menu.
2. Select (in the Session Manager) the previously-rendered playblast you want to swap out for new ones
3. Pick the *Mark Selected as Target* item on the *Maya* menu.

Henceforth, future playblasts will be swapped into the slot occupied by the one you just marked. Possible "render into context" workflows include:

Your shot in the cut

Load the shots before and after yours and make a Sequence view. Note that you can trim shots by adjusting the in/out points on the timeline in the Source view.

Wipe between the latest playblast and a previous take, or reference footage

Add one or more sources (or use a previous playblast). Make a Stack view. Order the Stack to your liking by dragging and dropping in the Inputs section of the Session Manager. Turn on Wipes (Tools menu).

Tile newest playblast with one or more others, reference footage, etc

Add one or more sources (or use a previous playblast). Make a Layout view. Select Tile/Column/Row (or even arrange by hand with the Manual mode) from the *Layout* menu. Arrange the Layout to your liking by dragging and dropping in the Inputs section of the Session Manager.

Last updated 2014-04-08 12:29:03 PDT

CHAPTER

FIFTYNINE

NUKE STYLE FILE SYNTAX IN OPEN RV

You can use nuke's printf-esque %d syntax in RV and we've extended the existing shake-esque syntax so you can mix and match.

> **Note:** previous versions of RV required using -ns in order to use nuke-like syntax. This is no longer necessary.

Given some files called foo.0001.exr, foo.0002.exr, foo.0003.exr, ... foo.0100.exr:

From the shell you can refer to the entire sequence using any of the following:

```
shell> rv foo.#.exr
shell> rv foo.@@@@.exr
shell> rv foo.%04d.exr
```

If you're current directory contains the sequence this will also work (in a unix shell):

```
shell> rv foo.*.exr
```

To constrain the sequence to frames 1 through 50 any of the following will work:

```
shell> rv foo.1-50#.exr
shell> rv foo.%04d.exr 1-50
shell> rv foo.1-50%04d.exr
shell> rv foo.#.exr 1-50
```

Its also possible to use multiple ranges, but only the shake-like method will work for that currently:

```
shell> rv foo.1-10,20-50,60-100#.exr
```

Any of these should also work inside brackets [ ].

CHAPTER

SIXTY

NUKE/OPEN RV INTEGRATION

60.1 Introduction

Rather than just attach a “flipbook” to Nuke, the goal of this integration effort is to provide compositors with a unified framework in which RV’s core media functionality (playback, browsing, arranging, editing, etc) is always instantly available to augment and enhance Nuke’s own capabilities.

Key features include:

- Checkpointing: Save a rendered frame with a copy of the current nuke script
- Rendering: Save a rendered sequence with a copy of the current nuke script
- Background rendering in Nuke 6.2 **and** 6.1
- Live update of RV during renders, showing the latest frame rendered
- Rendered frames visible in RV as soon as they are written
- Rendered frames from canceled renders are visible
- Render directly into a slap comp or sequence in RV
- Full checkpoints: copies of entire ranges of frames, for comparison
- Visual browsing of checkpoints and renders
- Visual comparison (wipes, tiled) of checkpoints and renders
- Restoring the script to the state of any checkpoint or render
- Read and Write nodes in the script dynamically mirrored as sources in RV
- Read/Write node path, frame range, color space dynamically synced to RV
- Node selection dynamically synced to the View in RV
- Frame changes in Nuke dynamically synced to RV frame
- RV Sources can be used to create the corresponding Nuke Read node
- All Render/Checkpoint context retained in session file on disk
- Support for `%V` -style stereoscopic Reads, Writes, renders, and checkpoints.

60.2 Note to Users

Thanks for trying out the software; the integration toolset is in active development, and we'd very much appreciate any bug reports, feature requests, or other comments.

Before you send us bug reports and feature requests, however, you might want to check over the list of known issues and planned work in *this appendix* .

60.3 Updating an Existing Installation

If this Package is updated within a new Open RV distribution, you might need to update the Python code on the "Nuke side". To do so, just follow the Installation instructions below.

If the RV and Nuke components of the integration package are mis-matched, you'll get an error dialog when you start RV from Nuke.

60.4 Installation

60.4.1 Personal Installation

1. Start RV and go to the *Packages* tab of the *Preferences* dialog
2. Find *Nuke Integration* in the Package list and click the *Load* toggle next to it.
3. Restart RV
4. Click the *Nuke* item on the *Tools* menu
5. From the *Nuke* menu, select the *Install Nuke Support Files* item and follow the directions.

To confirm that the Nuke support files are properly installed, start Nuke. You should see an *RV* menu on the main menubar, and if you select *RV/Preferences…* , you should get the appropriate dialog.

That's it for installation!

60.4.2 Site-wide Installation

In what follows we suppose that you've installed RV in `/usr/local/RV` , and that you keep your Nuke scripts in subdirectories of `/usr/local/nuke/scripts` . If you do otherwise please adjust the paths below appropriately.

1. Make a subdir in your Nuke scripts area for the rvnuke support files:

   ```
   % mkdir /usr/local/nuke/scripts/rvnuke
   ```

2. Copy the Nuke support files into place

   ```
   % cp /usr/local/rv/plugins/SupportFiles/rvnuke/* /usr/local/nuke/scripts/rvnuke
   ```

3. Edit the `init.py` file in `/usr/local/nuke/scripts` to include this line:

   ```
   nuke.pluginAddPath('./rvnuke')
   ```

Done!

60.5 Getting Started

60.5.1 Rv Preferences

In order to launch RV from Nuke, Nuke needs to know where the RV executable is. To set this, start Nuke and select the *RV/Preferences…* menu item. Navigate to the RV executable you want to use with Nuke and hit *OK* .

This setting is stored and used across all future Nuke sessions.

You can also specify any additional default command line arguments for RV in the *RV Preferences* dialog.

If you have a RAID or other fast storage device you may want to configure the RV/Nuke integration to use a directory on this device as the base for all Session directories (see below). If so set the "Default Session Dir Base" preference accordingly.

60.5.2 Rv Project Settings

There are several settings that the integration uses that may be different for different Nuke projects. Once you have a script loaded, select the *RV/Project Settings…* menu item, and then the *RV* tab of the Project Settings.

The table below lists all the RV Project Settings, with explanations, but the most important is the "Session Directory." This directory is where all media, script versions, and other information is stored for this Nuke script/project. It **must** be unique for each project.

Session Directory

The root directory for all media, scripts and other information related to this project. It will be created if it does not exist. Since media will be stored under this directory, you may want to put it on a device with fast IO. This name **must** be uniqe across all projects.

You can set "Default Session Dir Base" in the RV Preferences (see above) so that by default all Session directories are created on your fast IO device.

Render File Format

The format of all media files created by rendering and checkpointing.

Nuke Node Selection → RV Current View

If this box is checked, every time you select a node in nuke, if RV is connected, the current RV view node will be set to the corresponding view. This lets you quickly view or play media, either input media associated with a Read node, or rendered media associated with any node that has been checkpointed or rendered.

Nuke Frame → RV Frame

If this box is checked, frame changes in Nuke will force the corresponding frame change in RV.

Nuke Read Node Changes → RV Sources

If this box is checked, the total set of Read nodes in the project will be dynamically synced to RV. That is, for every Read node in the project, there will be a corresponding Source in RV with the same media, available for playback on demand. Adding or Deleteing a Read node in Nuke will trigger the corresponding action in RV. Changes to Read node file path, frame range, and color space will also be reflected in RV.

60.5.3 Quick Start Summary

You must set the RV executable path using the *RV/Preferences..* menu item before you use RV with Nuke at all, and whenever you start work on a new project/script, use *RV/Project Settings…* to make sure that the *Session Directory* is set to something reasonable before you start RV from that script for the first time. See above for details.

60.5.4 Open RV Toolbar

Note that all the items on the RV menu are also available on the RV toolbar, which you can find in the Panes submenu.

60.6 Read/Write Nodes

Once you've set the RV path and Session Dir as described above, and have an interesting Nuke script loaded, try starting up RV with the *RV/Start RV* menu item. Assuming you have the *Sync Read Changes* setting active, as soon as RV starts you should see all the Read nodes in the script reflected as media Sources in RV.

If you don't see the Session Manager, try hitting the *x* to bring it up. In the Session Manager, You'll see a Folder called "Read Nodes" with a Source for each Read node in the script. Each source is labeled with the name of the corresponding Read node, and a timestamp for when it was last modified.

> **Note:** The Session Manager behavior at RV start-up can be set to "aways shown", "always hidden" or "remember previous state" using the "wrench" menu on the Session Manager.

You can double-click on each Source to play just that one, or on the "Read Nodes" folder to see them all.

Back in Nuke, note that if you edit the Path, Frame Range, or Color Space attributes of a Read node, the changes are reflected in the corresponding Source in RV.

If the *Sync Selection* setting is active, as you select various Read nodes in Nuke, the RV current view switches to the corresponding Source.

Also, if the *Sync Frame* setting is active, frame changes in the Nuke viewer will be reflected in RV.

Note that if you don't want all Read Nodes to be synced automatically, you can still sync some (or all) of them when you want to with the appropriate items on the *RV* menu.

Pretty much all the above applies to Write nodes as well.

60.7 Checkpoints and Renders

As with Read nodes, Checkpoints and Renders are representations in RV of particular nodes in Nuke. So the Frame and Selection syncing described in the Read Nodes section applies to Checkpoints and Renders as well.

Unlike Read nodes, the media associated with Checkpoints and Renders are generated from the Nuke script and so reflect the state of the script at the time of rendering.

60.7.1 Checkpoints

The point of a Checkpoint is to to visually label a particular point in your projects development, so that you can easily return to that point if you want to. When you've made some changes in your script, and reach a point where you want to go in another direction, or try something out, or work on a different aspect of the project, that's a good time to "bookmark" your work with a Checkpoint.

To make a Checkpoint, select a node that visually reflects the state of the script and select *RV/Create Checkpoint* . You'll see a new Source appear in RV, in a Folder named for the node you selected, with a single rendered frame from that node.

As you work on a particular aspect of your project, you may want to make many Checkpoints of a particular node, so that you can easily compare the visual effect of different parameter settings. They'll all be collected in a single folder in the Session Manager, and as with Read nodes, you can double click on a single one to view it, or double click on the folder itself to see them all.

60.7.2 Rendering

A Render is similar to a Checkpoint, but involves rendering a sequence of frames, instead of just one. To render, select the node of interest, then select *RV/Render to RV* . You'll get a dialog with some parameters:

Output Node

The name of the node to be rendered.

Use Selected

If checked, the output node will always be equal to whatever node is selected when the dialog is shown. If unchecked, the output node will "stick" and not be affected by the selection.

First Frame

The first frame in the sequence to be rendered.

Last Frame

The last frame in the sequence to be rendered.

Since Renders can occupy significant disk space, successive renders of the same node overwrite any pre-existing render. But each render also automatically generates a single-frame Checkpoint of the same Nuke state. Also, deleting a Render or Checkpoint in the Session Manager (with the Trash Can button), also removes the corresponding media from disk.

During a Render, RV updates dynamically to show you all the frames rendered so far. If the render is canceled, you still see in RV any frames that completed before the cancel. RV Sources from renders go into the same Folder as Checkpoints from the same node.

60.7.3 Full Checkpoints

A Full Checkpoint is just like a regular checkpoint except that an entire sequence of frames is saved. To create a Full Checkpoint, select a Render in the RV Session Manager and then select *Create Full Checkpoint* from the *Nuke* menu in RV.

60.8 Working with Media in Open RV

Relevant here is the chapter on the *Session Manager* and the section on *navigation*

60.8.1 Folders

The Nuke integration makes use of Folders to organize your media. You'll have a folder for all your Read nodes, a folder of checkpoints and renders for each rendered node, and a catch-all folder called "Other" to collect the rest. All folders are viewable and make for a handy "browsing" interface.

60.8.2 Comparisons

You can easily Compare two or more renders or checkpoints (or any views, actually). Just select the views of interest in the Session Managerand select on the comparison items on RV's *Nuke* menu: *Nuke/Wipe Selected Views* or *Nuke/Tile Selected Views* .

60.9 Modifying the Nuke Project from Open RV

60.9.1 Restoring Checkpoints

Any Checkpoint (or Render) can provide a source from which the Nuke project can be restored to the state it was in when the Checkpoint's media was rendered. To restore a Checkpoint, select it in the RV Session Manager, and choose *Nuke/Restore Checkpoint* . After a confirmation dialog, the Nuke script will be restored.

The navigation techniques referenced above combine with checkpoint restoration to produce some nice workflows (I think). For example:

1. After lots of rendering and checkpointing of node *FinalMerge* , double-click on the *Renders of FinalMege* folder to see a layout of all the checkpoints and renders.
2. Bring up the Image Info widget to mouse around and see the names and timestamps of all the views in the layout.
3. Double click on one if the tiles to examine that checkpoint more closely.
4. Decide to restore this checkpoint, it's alread selected, so just hit *Nuke/Restore Checkpoint*

Also note that the Restore operation is undo-able, from the Nuke *Edit* menu.

60.9.2 Adding Read Nodes

Of course you can still view media that's unconnected to the Nuke project in a connected RV. So you can for example browse an element library. Once you have media that you'd like to include in your project, just select the Sources in the Session Manager and choose *Nuke/Create Nuke Read Node* . The corresponding Read node will be created in Nuke. Actually you can create any number at once by just selecting however many you want.

CHAPTER

SIXTYONE

OTIO READER PACKAGE OVERVIEW

The RV package plugin is based on the OTIO project.

The OTIO Reader is an RV package that allows RV to import and export OTIO files. The package is installed and loaded by default with support for all OTIO schemas part of OTIO version 0.15. So users can now import and export OTIO files through any one of the various RV import/export paths.

The rest of this section will detail how to customize the import and export processes, and how to support custom OTIO schemas or metadata.

61.1 Imported and exported nodes

When importing, the resulting RV node graph will be added to the current view node. When exporting, the current view node will be the root of the exported OTIO. Since not all RV nodes are supported as the root of an OTIO file, the export option is only available when the view node is an RVSequenceGroup, RVStackGroup, or RVSourceGroup.

61.2 Package contents

The RV package is installed in the usual Python package installation location: `<Installation directory>/Plugins/Python`.

The package also uses a number of files located in `Plugins/SupportFiles/otio_reader` and detailed below.

- `manifest.json`: Provides examples of schemas and hooks used in the import process.
- `cdlExportHook.py`: An example of exporting an RVLinearize to a custom CDL effect in OTIO.
- `cdlHook.py`: An example of importing a custom CDL effect in OTIO into an RVLinearize node.
- `cdlSchema.py` An example schema of a CDL effect.
- `clipHook.py`: An example of a hook called before importing a clip.
- `customTransitionHook.py`: An example of importing a custom transition in OTIO into RV.
- `effectHook.py`: A helper file for adding and setting RV properties from OTIO.
- `sourcePostExportHook.py`: A hook called after an RVSourceNodeGroup has been exported to a `Clip`. This can be used to add custom effects for other nodes within the same source group. The RVLinearize node is provided as an example.
- `retimeExportHook.py`: A hook for exporting an RVRetime node to OTIO schemas LinearTimeWarp or FreezeFrame.
- `timeWarpHook.py`: A hook for importing OTIO's LinearTimeWarp and FreezeFrame schemas.

61.3 Using the OTIO hook system

The RV package is using the existing OTIO hook system to execute its hooks, so any existing `manifest.json` in the `OTIO_PLUGIN_MANIFEST_PATH` environment variable can be modified to work with RV's hook system, including RV's own `manifest.json` file.

The `OTIO_PLUGIN_MANIFEST_PATH` environment variable can be modified to work with RV's hook system, including RV's own manifest.json.

There are three types of OTIO hooks that will be called by the RV package:

- pre- and post-hooks for both import and export
- custom schema hooks for import
- custom transition hook for import
- custom RV node hooks for export

61.3.1 Pre and post-hooks

The pre- and post-hooks are called before each known OTIO schema during import or RV node during export. For import, they are named `pre_hook_[schema_name]` and `post_hook_[schema_name]` and will be called just before and just after processing the schema. For export, they are similarly named `pre_export_hook_[node_name]` and `post_export_hook_[node_name]`. For example, a hook can be added named `pre_hook_Clip` and it will be called just before the import process has created an RV source based on the Clip. These hooks are useful for handling custom metadata in schemas. The provided `clipHook.py` is an example of this.

61.3.2 Custom schema hooks

During import, the custom schema hooks are called whenever the `otio_reader` encounters a schema it cannot handle. These hooks are named `[schema_name]_to_rv`. For example, `CDL_to_rv` will be called whenever a schema named CDL is found. The provided `cdlHook.py` is an example of this. This is most commonly used for custom effects and other schemas that are not provided by OTIO.

The custom transition hook is similar to the custom schema hooks, except instead of being based on the schema name, it is called whenever the `transition_type` field of the Transition schema is set to Custom. The `customTransitionHook.py` is an example of this.

61.3.3 Custom transition hook

The custom transition hook is similar to the custom schema hooks, except instead of being based on the schema name, it is called whenever the `transition_type` field of the Transition schema is set to Custom. The `customTransitionHook.py` is an example of this.

61.3.4 Custom Open RV node hooks

During export, the custom RV node hooks are called whenever the `otio_reader` encounters a node it cannot handle. These hooks are named `export_[node_name]`. The provided `cdlExportHook.py` is an example of this. This is commonly used for effects in RV that do not have equivalent OTIO schemas.

61.3.5 Import Hook file parameters

The `in_timeline` parameter in the OTIO hook functions will be set to the OTIO schema being processed, not the full timeline. For example, in `cdlHook.py` the `in_timeline` is the CDL effect and in the `pre_hook_Clip` it is the Clip.

The `argument_map` will contain the context that can be helpful when creating RV nodes. For example, the following keys can currently be present, depending on which hook is being called:

- `transition`: `RV transition` output node
- `stack`: `RVStackGroup` output node
- `sequence`: RVSequenceGroup output node
- `source`: `RVSource` output node
- `source_group`: `RVSourceGroup` output node
- `track_kind`: `OTIO track_kind` property of the OTIO Track currently being processed

The `in_timeline parameter` in the OTIO hook functions will be set to the OTIO schema being processed—not the full timeline. For example, in `cdlHook.py` the `in_timeline` is the CDL effect and in the `pre_hook_Clip` it is the Clip.

The `argument_map` will contain the context that can be helpful when creating RV nodes. For example, the following keys can currently be present, depending on which hook is being called:

- `transition`: `RV transition` output node
- `stack`: `RVStackGroup` output node
- `sequence`: `RVSequenceGroup` output node
- `source`: `RVSource` output node
- `source_group`: `RVSourceGroup` output node
- `track_kind`: `OTIO track_kind` property of the OTIO Track currently being processed
- `global_start_time`: The `global_start_time` of the OTIO Timeline.

61.3.6 Import Hook file return values

When RV nodes are being created—such as effects or transitions—the return value should be set to the name of the created node. The `otio_reader` will add the node as input at the current location in RV's node graph. It will also have its metadata added as a new property on the node named `otio.metadata`.

For pre- and post- hooks, no return value is expected. However, the pre-hooks can optionally return False, which will quit processing the current schema and all its children.

61.3.7 Export Hook file parameters

The `in_timeline` parameter in the OTIO hook functions will be set to the OTIO timeline that has been constructed up to that point. The `argument_map` will contain an `rv_node_name` key with its value set to the name of the RV node being processed.

Depending on the context in which the hook is called, the `argument_map` parameter can make the following keys available:

- `in_frame`: The `edl.in` value for clips and gaps
- `out_frame`: The `edl.out` value for clips and gaps
- `cut_in_frame`: The `edl.frame` value for clips and gaps
- `pre_item`: The OTIO item before a transition
- `post_item`: The OTIO item after a transition

61.3.8 Export Hook file return values

If an effect returned, the plugin will add it to the effect list of the OTIO Clip or Gap. The OTIO Clip or Gap is created from the RVSourceGroup to which the effect is attached.

For pre- and post- hooks, no return value is expected. However, the pre-hook nodes can optionally return `False`, which will quit processing the current node and all of its inputs.

61.4 Media Multi-Reference Support

If you're using OTIO version 0.15 or later, you have access to the Media Multi-Reference (MMR) feature.

With MMR, known and supported in RV as *Multiple Media Representations*, a clip can reference high-resolution media and proxies. In RV, this allows the user to easily switch between the different representations of the same media.

To accommodate these workflows, a single media reference contained within a clip becomes a dictionary of media references. Below is what the difference looks like when serialized.

OTIO Representation without MMR

```
"media_reference" : {
   "OTIO_SCHEMA": "ImageSequenceReference.1",
   "metadata": {},
   "name": "",
   "available_range": null,
   "available_image_bounds": null,
   "target_url_base": "/mnt/nvme/Projects/aProject/scene01",
   "name_prefix": "opening.",
   "name_suffix": ".exr",
   "start_frame": 1,
   "frame_step": 1,
   "rate": 1.0,
   "frame_zero_padding": 4,
   "missing_frame_policy": "error"
}
```

OTIO Representation with MMR

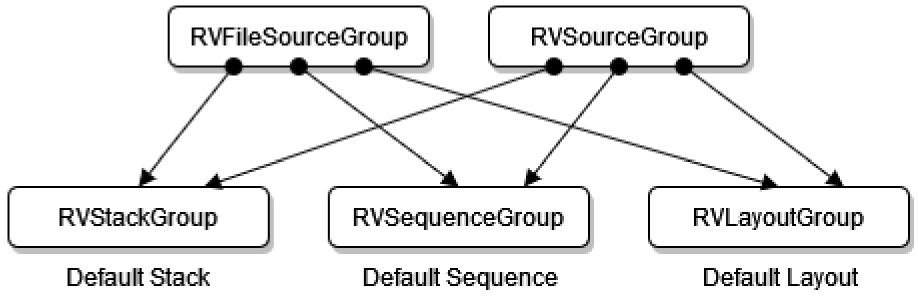
```
"media_references": {
   "Frames": {
       "OTIO_SCHEMA": "ImageSequenceReference.1",
       "metadata": {},
       "name": "",
       "available_range": null,
       "available_image_bounds": null,
       "target_url_base": "/mnt/nvme/Projects/aProject/scene01",
       "name_prefix": "opening.",
       "name_suffix": ".exr",
       "start_frame": 1,
       "frame_step": 1,
       "rate": 1.0,
       "frame_zero_padding": 4,
       "missing_frame_policy": "error"
   },
   "Movie": {
       "OTIO_SCHEMA": "ExternalReference.1",
       "metadata": {},
       "name": "",
       "available_range": {
           "OTIO_SCHEMA": "TimeRange.1",
           "duration": {
               "OTIO_SCHEMA": "RationalTime.1",
               "rate": 24.0,
               "value": 1300.0
           },
           "start_time": {
               "OTIO_SCHEMA": "RationalTime.1",
               "rate": 24.0,
               "value": 1.0
           }
       },
       "available_image_bounds": null,
       "target_url": "/mnt/nvme/Projects/aProject/scene01/opening.mov"
   },
   "Streaming": {
       "OTIO_SCHEMA": "ExternalReference.1",
       "metadata": {},
       "name": "",
       "available_range": {
           "OTIO_SCHEMA": "TimeRange.1",
           "duration": {
               "OTIO_SCHEMA": "RationalTime.1",
               "rate": 24.0,
               "value": 1300.0
           },
           "start_time": {
               "OTIO_SCHEMA": "RationalTime.1",
               "rate": 24.0,
               "value": 1.0
           }
       },
```

(continues on next page)

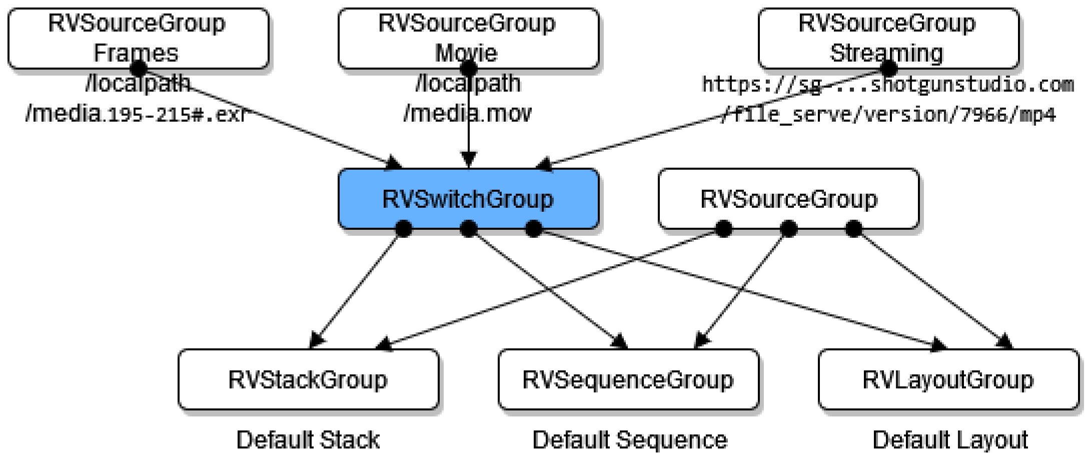
(continued from previous page)

```
        "available_image_bounds": {
            "OTIO_SCHEMA": "Box2d.1",
            "min": {
                "OTIO_SCHEMA": "V2d.1",
                "x": -1.1764705882352942,
                "y": -0.5
            },
            "max": {
                "OTIO_SCHEMA": "V2d.1",
                "x": 1.1764705882352942,
                "y": 0.5
            }
        },
        "target_url": "https://acme.shotguncloud.com/file_serve/version/23088/mp4"
    }
},
"active_media_reference_key": "Streaming"
```

RV Representation without MMR



RV Representation with MMR



When you import an OTIO file in RV, you always end up with an RV SwitchGroup where `active_media_reference` sets the active RVSourceGroup.

When you export from RV an OTIO file, every SwitchGroup node outputs as a Clip with its `active_media_reference_key` set to the currently active RVSourceGroup.

Open RV and its companion tools, RVIO and RVLS, have been created to support digital artists, directors, supervisors, and production crews who need reliable, flexible, high-performance tools to review image sequences, movie files, and audio. RV is clean and simple in appearance and has been designed to let users load, play, inspect, navigate and edit image sequences and audio as simply and directly as possible. RV's advanced features do not clutter its appearance but are available through a rich command-line interface, extensive hot keys and key-chords, and smart drag/drop targets. RV can be extensively customized for integration into proprietary pipelines. The RV Reference Manual has information about RV customization.