

---

# Ardour Documentation

nicolas

Jun 05, 2019



# CONTENTS

<b>1</b>	<b>34 - Importing and Exporting Session Data</b>	<b>145</b>
<b>2</b>	<b>34.1 - Import Dialog</b>	<b>147</b>
2.1	The Soundfile Information Box . . . . .	147
2.2	Auditioner . . . . .	148
2.3	Importing options . . . . .	148
<b>3</b>	<b>34.2 - Supported File Formats</b>	<b>149</b>
<b>4</b>	<b>34.3 - Adding Pre-existing Material</b>	<b>151</b>
<b>5</b>	<b>34.4 - Copying Versus Linking</b>	<b>153</b>
5.1	Copying . . . . .	153
5.2	Linking . . . . .	153
<b>6</b>	<b>34.5 - Searching for Files Using Tags</b>	<b>155</b>
6.1	Creating and adding tags . . . . .	155
6.2	Searching for files by tag . . . . .	155
<b>7</b>	<b>34.6 - Stem Exports</b>	<b>157</b>
<b>8</b>	<b>35 - File and Session Management and Compatibility</b>	<b>159</b>
<b>9</b>	<b>35.1 - Session Templates</b>	<b>161</b>
9.1	Creating a Session Template . . . . .	161
9.2	Using a Session Template . . . . .	161
9.3	Managing Templates . . . . .	161
<b>10</b>	<b>35.2 - Snapshots</b>	<b>163</b>
10.1	Switching to a Snapshot . . . . .	163
10.2	Starting Ardour With a Snapshot . . . . .	163
<b>11</b>	<b>35.3 - Cleaning Up Sessions</b>	<b>165</b>
11.1	Bring all media into session folder . . . . .	165
11.2	Reset Peak Files . . . . .	165
11.3	Clean-up Unused Sources... . . . . .	165
11.4	Flush Wastebasket . . . . .	166
<b>12</b>	<b>35.4 - Interchange with other DAWs</b>	<b>167</b>
12.1	Transferring an Ardour session from / to another DAW . . . . .	167
12.2	Importing ProTools® files . . . . .	167
12.3	Using AATranslator . . . . .	167

<b>13 Part V - Playback &amp; Recording</b>	<b>169</b>
<b>14 36 - Playing Back Track Material</b>	<b>171</b>
<b>15 36.1 - Controlling Playback</b>	<b>173</b>
15.1 Positioning the Playhead . . . . .	173
15.2 Looping the Transport . . . . .	173
<b>16 36.2 - Using Key Bindings</b>	<b>175</b>
<b>17 37 - Audio Recording</b>	<b>177</b>
<b>18 37.1 - Monitoring</b>	<b>179</b>
<b>19 37.2 - Latency Considerations</b>	<b>181</b>
<b>20 37.3 - Monitor Signal Flow</b>	<b>183</b>
20.1 External Monitoring . . . . .	183
20.2 JACK-Based Hardware Monitoring . . . . .	183
20.3 Software Monitoring . . . . .	183
<b>21 38 - Punch Recording Modes</b>	<b>187</b>
<b>22 38.1 - Working With Markers</b>	<b>189</b>
<b>23 38.2 - Creating Location Markers</b>	<b>191</b>
<b>24 38.3 - Creating Range Markers</b>	<b>193</b>
24.1 Creating a Range on the timeline . . . . .	193
24.2 Editing a Range . . . . .	193
<b>25 38.4 - Moving Markers</b>	<b>195</b>
25.1 Single marker . . . . .	195
25.2 Multiple markers . . . . .	195
25.3 Both ends of a range marker . . . . .	195
<b>26 38.5 - The Loop Range</b>	<b>197</b>
<b>27 38.6 - Marker Context Menu</b>	<b>199</b>
<b>28 38.7 - Punch Range</b>	<b>201</b>
<b>29 Part VI - Editing</b>	<b>203</b>
<b>30 39 - Navigating the Editor</b>	<b>205</b>
30.1 Scrolling . . . . .	205
30.2 Zooming . . . . .	206
30.3 Height of the tracks . . . . .	206
<b>31 40 - Editing Basics</b>	<b>209</b>
<b>32 40.1 - Working With Regions</b>	<b>211</b>
32.1 Working With Regions . . . . .	211
<b>33 40.2 - Region Naming</b>	<b>213</b>
33.1 Whole File Region Names . . . . .	213
33.2 Normal Region Names . . . . .	213

33.3	Copied Region Names . . . . .	213
33.4	Renaming Regions . . . . .	214
<b>34</b>	<b>40.3 - Corresponding Regions Selection</b>	<b>215</b>
34.1	How Ardour Decides Which Regions are “Corresponding” . . . . .	215
34.2	Overlap Correspondence . . . . .	215
<b>35</b>	<b>40.4 - Region Context Menu</b>	<b>217</b>
<b>36</b>	<b>40.5 - Common Region Edit Operations</b>	<b>219</b>
<b>37</b>	<b>40.6 - Copy Regions</b>	<b>221</b>
37.1	Copy a Single Region . . . . .	221
37.2	Copy Multiple Regions . . . . .	221
37.3	Fixed-Time Copying . . . . .	221
<b>38</b>	<b>40.7 - Move Regions With the Mouse</b>	<b>223</b>
38.1	Move Multiple Regions . . . . .	223
38.2	Fixed-Time Motion . . . . .	223
<b>39</b>	<b>40.8 - Align (Spot) Regions</b>	<b>225</b>
<b>40</b>	<b>41 - Edit Mode and Tools</b>	<b>227</b>
<b>41</b>	<b>41.1 - Editing Clocks</b>	<b>229</b>
41.1	Clock Modes . . . . .	229
41.2	Changing clock values with the keyboard . . . . .	229
41.3	Avoiding the mouse entirely . . . . .	230
41.4	Entering Partial Times . . . . .	230
41.5	Entering Delta Times . . . . .	230
41.6	Changing clock values with the mouse . . . . .	230
<b>42</b>	<b>41.2 - Which Regions Are Affected?</b>	<b>231</b>
<b>43</b>	<b>42 - Making Selections</b>	<b>233</b>
43.1	Track Selection . . . . .	233
43.2	Region Selection . . . . .	233
43.3	Deselecting a Region . . . . .	233
43.4	Selecting Multiple Regions in a Track . . . . .	233
43.5	Selecting All Regions in a Track . . . . .	234
43.6	Selecting Multiple Regions Across Different Tracks . . . . .	234
43.7	Selecting a Region From the Region List . . . . .	234
<b>44</b>	<b>43 - Editing Regions and Selections</b>	<b>235</b>
<b>45</b>	<b>43.1 - Trimming Regions</b>	<b>237</b>
45.1	Drag-Trimming With the Mouse . . . . .	237
45.2	Other Trimming operations . . . . .	237
<b>46</b>	<b>43.2 - Push/Pull Trimming</b>	<b>239</b>
<b>47</b>	<b>43.3 - Stretching</b>	<b>241</b>
<b>48</b>	<b>43.4 - Separate Under</b>	<b>245</b>
48.1	Separate Under . . . . .	245
48.2	Separate Using Range . . . . .	245

<b>49 43.5 - Strip Silence from Audio Regions</b>	<b>247</b>
<b>50 43.6 - Insert/Remove Time</b>	<b>249</b>
50.1 Insert Time . . . . .	249
50.2 Remove Time . . . . .	250
<b>51 43.7 - Region Properties</b>	<b>251</b>
51.1 Region properties . . . . .	251
<b>52 44 - Fades and Crossfades</b>	<b>255</b>
52.1 Region Fades . . . . .	255
52.2 Crossfades . . . . .	255
52.3 Fade Shapes . . . . .	256
<b>53 45 - Gain Envelopes</b>	<b>259</b>
53.1 Manipulating Gain Envelopes . . . . .	259
<b>54 46 - Playlists</b>	<b>261</b>
<b>55 46.1 - Understanding Playlists</b>	<b>263</b>
55.1 Tracks are not Playlists . . . . .	263
55.2 Playlists are Cheap . . . . .	263
<b>56 46.2 - Playlist Operations</b>	<b>265</b>
56.1 Renaming Playlists . . . . .	265
56.2 Sharing Playlists . . . . .	265
<b>57 46.3 - Playlist Usecases</b>	<b>267</b>
57.1 Using Playlists for Parallel Processing . . . . .	267
57.2 Using Playlists for “Takes” . . . . .	267
57.3 Using Playlists for Multi-Language Productions . . . . .	267
<b>58 47 - Rhythm Ferret</b>	<b>269</b>
58.1 Accessing the Rhythm Ferret . . . . .	270
58.2 The “Mode” selection . . . . .	270
58.3 The Percussive Onset mode . . . . .	270
58.4 The Note Onset Mode . . . . .	271
58.5 Manual adjustment . . . . .	271
<b>59 Part VII - MIDI</b>	<b>273</b>
<b>60 48 - MIDI Editing</b>	<b>275</b>
60.1 Key features of Ardour MIDI handling . . . . .	275
60.2 Notable Differences . . . . .	275
<b>61 49 - Creating MIDI Tracks</b>	<b>277</b>
<b>62 50 - Creating MIDI Regions</b>	<b>279</b>
<b>63 51 - Adding New Notes</b>	<b>281</b>
63.1 Adding new notes . . . . .	281
<b>64 52 - Changing Note Properties</b>	<b>283</b>
<b>65 53 - Handling Overlapping Notes</b>	<b>285</b>
<b>66 54 - Note Cut, Copy and Paste</b>	<b>287</b>

<b>67 55 - Note Selection</b>	<b>289</b>
67.1 Selecting/Navigating note-by-note . . . . .	289
67.2 Selecting notes with the mouse . . . . .	289
67.3 Listening to Selected Notes . . . . .	289
<b>68 56 - Quantizing MIDI</b>	<b>291</b>
<b>69 57 - Step Entry</b>	<b>293</b>
<b>70 58 - Patch Change</b>	<b>295</b>
70.1 Inserting Patch Changes . . . . .	295
70.2 Modifying Patch Changes . . . . .	295
70.3 Moving Patch Changes . . . . .	295
70.4 Removing Patch Changes . . . . .	296
70.5 Names for Patch Numbers: MIDNAM files . . . . .	296
<b>71 59 - Independent and Dependent MIDI Region Copies</b>	<b>297</b>
71.1 Changing dependent/independent copying for the entire session . . . . .	297
71.2 Making an existing copy of a MIDI region independent . . . . .	297
<b>72 60 - Transposing MIDI</b>	<b>299</b>
<b>73 61 - Automating MIDI-Pitch bending and aftertouch</b>	<b>301</b>
<b>74 62 - Transforming MIDI-Mathematical operations</b>	<b>303</b>
<b>75 63 - MIDI List Editor</b>	<b>305</b>
<b>76 64 - MIDI Tracer</b>	<b>307</b>
<b>77 65 - MIDI Recording</b>	<b>309</b>
<b>78 66 - MIDI Scene Automation</b>	<b>311</b>
78.1 Recording Scene Changes . . . . .	311
78.2 Manually Creating Scene Changes . . . . .	311
78.3 Playing back Scene Changes . . . . .	311
78.4 Editing Scene Changes . . . . .	312
78.5 Disabling Scene Changes . . . . .	312
<b>79 Part VIII - Arranging</b>	<b>313</b>
<b>80 67 - Time, Tempo and Meter</b>	<b>315</b>
<b>81 67.1 - Tempo and Meter</b>	<b>317</b>
81.1 Tempo . . . . .	317
81.2 Meter . . . . .	318
<b>82 67.2 - Techniques for Working with Tempo and Meter</b>	<b>319</b>
82.1 Matching a recorded tempo with a tempo ramp . . . . .	319
82.2 Other use cases . . . . .	323
<b>83 Part IX - Mixing</b>	<b>325</b>
<b>84 68 - Basic Mixing</b>	<b>327</b>
<b>85 68.1 - Metering in Ardour</b>	<b>329</b>
85.1 Introduction . . . . .	329

85.2	Meter Types . . . . .	329
85.3	Ardour Specifics . . . . .	331
85.4	Overview of meter types . . . . .	332
<b>86</b>	<b>68.2 - Signal Routing</b>	<b>335</b>
<b>87</b>	<b>68.3 - Aux Sends</b>	<b>337</b>
87.1	Adding a new aux bus . . . . .	337
87.2	Adding a send to an aux bus . . . . .	337
87.3	Adding a new aux bus and sending a Track Group to it . . . . .	338
87.4	Altering Send Levels . . . . .	338
87.5	Disabling Sends . . . . .	338
87.6	Send Panning . . . . .	338
<b>88</b>	<b>68.4 - Comparing Aux Sends and Subgroups</b>	<b>339</b>
<b>89</b>	<b>68.5 - External Sends</b>	<b>341</b>
89.1	Adding an External Send . . . . .	341
89.2	Removing an External Send . . . . .	341
89.3	Altering Send Levels . . . . .	341
89.4	Disabling Sends . . . . .	341
89.5	Editing Send Routing . . . . .	342
<b>90</b>	<b>68.6 - Inserts</b>	<b>343</b>
<b>91</b>	<b>68.7 - Subgrouping</b>	<b>345</b>
<b>92</b>	<b>68.8 - Patchbay</b>	<b>347</b>
92.1	Variants on the Patchbay . . . . .	349
92.2	Other patchbay features . . . . .	349
<b>93</b>	<b>68.9 - Track/Bus Signal Flow</b>	<b>351</b>
93.1	Overview . . . . .	351
93.2	Strict I/O . . . . .	351
93.3	Customizing the Signal Flow: The Pin Connection window . . . . .	353
<b>94</b>	<b>68.10 - Sidechaining</b>	<b>355</b>
94.1	A simple example: Sidechain compression . . . . .	355
94.2	MIDI Sidechaining . . . . .	355
94.3	Pre-processing the sidechained signal . . . . .	356
<b>95</b>	<b>68.11 - Muting and Soloing</b>	<b>357</b>
95.1	Without a monitor bus . . . . .	357
95.2	With a monitor bus . . . . .	357
95.3	Other solo options . . . . .	358
<b>96</b>	<b>68.12 - Panning</b>	<b>361</b>
96.1	Types of Panners . . . . .	361
<b>97</b>	<b>68.13 - Mono Panner</b>	<b>363</b>
97.1	Mono Panner User Interface . . . . .	363
97.2	Using the mouse . . . . .	363
97.3	Keyboard bindings . . . . .	363
97.4	Using the scroll wheel/touch scroll . . . . .	364
<b>98</b>	<b>68.14 - Balance Control</b>	<b>365</b>

<b>99 68.15 - Stereo Panner</b>	<b>367</b>
99.1 Stereo Panner User Interface . . . . .	367
99.2 Stereo panning caveats . . . . .	369
<b>100 68.16 - VBAP Panner</b>	<b>373</b>
100.1 Basic concepts . . . . .	373
100.2 Speaker layout . . . . .	373
100.3 N:M panning . . . . .	375
<b>101 69 - Plugin and Hardware Inserts</b>	<b>377</b>
<b>102 69.1 - Working With Plugins</b>	<b>379</b>
102.1 Adding/Removing/Copying Plugins . . . . .	379
<b>103 69.2 - Processor Box</b>	<b>381</b>
103.1 Adding Processors . . . . .	382
103.2 To Reorder (Move) Processors . . . . .	382
103.3 To Enable/Disable a Processor . . . . .	382
103.4 Selecting Processors . . . . .	383
103.5 Removing Processors . . . . .	383
<b>104 69.3 - Plugin Manager</b>	<b>385</b>
104.1 Plugin Display Status . . . . .	385
104.2 Filtering Listed Plugins . . . . .	386
104.3 Inserting Plugins in the Processor Box . . . . .	386
<b>105 69.4 - Managing Plugin Presets</b>	<b>387</b>
105.1 What Is a Plugin Preset? . . . . .	387
105.2 The Preset Selector . . . . .	387
105.3 Loading a New Preset . . . . .	387
105.4 Creating a Preset . . . . .	387
105.5 Saving a Preset . . . . .	388
105.6 Deleting a preset . . . . .	388
<b>106 69.5 - Working with Ardour-built Plugin Editors</b>	<b>389</b>
106.1 Generic Plugin Editor . . . . .	389
106.2 Analysis Graph . . . . .	389
106.3 MIDI instruments specificities . . . . .	389
<b>107 69.6 - Plugins Bundled With Ardour</b>	<b>391</b>
<b>108 69.7 - Getting More Plugins</b>	<b>393</b>
108.1 Plugins by Standard: . . . . .	393
108.2 How to install plugins? . . . . .	394
<b>109 69.8 - Using Windows VST Plugins on Linux</b>	<b>395</b>
109.1 A Plea To Plugin Manufacturers . . . . .	395
<b>110 70 - Automation</b>	<b>397</b>
<b>111 70.1 - Automation Nomenclature</b>	<b>399</b>
<b>112 70.2 - Automation Modes</b>	<b>401</b>
<b>113 70.3 - Automation Lanes</b>	<b>403</b>

<b>114</b>	<b>70.4 - Automation Curves</b>	<b>405</b>
<b>115</b>	<b>70.5 - Controlling a Track with Automation</b>	<b>407</b>
115.1	Recording an Automation Curve Using Write Mode . . . . .	407
115.2	Recording an Automation Curve Using Touch Mode . . . . .	408
115.3	Drawing an Automation Curve Using the Mouse . . . . .	408
115.4	Controlling the Track . . . . .	408
<b>116</b>	<b>71 - Mixdown</b>	<b>409</b>
<b>117</b>	<b>71.1 - Export Dialog</b>	<b>411</b>
117.1	File Format . . . . .	411
117.2	The location . . . . .	412
117.3	Analyze exported audio . . . . .	413
117.4	Time Span . . . . .	414
117.5	Channels . . . . .	415
117.6	Stem Export . . . . .	415
<b>118</b>	<b>71.2 - Export Format Profiles</b>	<b>417</b>
118.1	Export Format Profiles . . . . .	417
<b>119</b>	<b>Part X - Video</b>	<b>421</b>
<b>120</b>	<b>72 - Video Timeline and Monitoring</b>	<b>423</b>
<b>121</b>	<b>73 - Video Timeline Setup</b>	<b>425</b>
121.1	Single Machine . . . . .	425
121.2	Studio Setup . . . . .	425
<b>122</b>	<b>74 - Transcoding, Formats &amp; Codecs</b>	<b>427</b>
122.1	Ardour specific issues . . . . .	427
<b>123</b>	<b>75 - Workflow &amp; Operations</b>	<b>429</b>
123.1	Overview of Operations . . . . .	429
123.2	Adding a video . . . . .	429
123.3	Working with A/V . . . . .	432
123.4	Exporting Video . . . . .	432
<b>124</b>	<b>Part XI - Control Surfaces</b>	<b>435</b>
<b>125</b>	<b>76 - Controlling Ardour with OSC</b>	<b>437</b>
<b>126</b>	<b>76.1 - OSC: Controlling Ardour with OSC</b>	<b>439</b>
126.1	Connecting to Ardour via OSC . . . . .	439
126.2	Control Surface Set Up . . . . .	439
126.3	Querying Ardour for information . . . . .	441
126.4	List of OSC messages . . . . .	442
<b>127</b>	<b>76.2 - OSC: Using the Setup Dialog</b>	<b>449</b>
127.1	Dialog settings . . . . .	449
<b>128</b>	<b>76.3 - OSC: Querying Ardour</b>	<b>457</b>
128.1	A list of strips . . . . .	457
128.2	A list of sends . . . . .	458
128.3	A list if tracks that send audio to a bus . . . . .	458
128.4	A list of plug-ins for strip . . . . .	459

128.5 A list of a plug-in's parameters . . . . .	459
<b>12976.4 - OSC: Feedback</b>	<b>461</b>
129.1 List of OSC feedback messages . . . . .	462
<b>13076.5 - OSC: Feedback and Strip-types Values</b>	<b>467</b>
130.1 strip_types . . . . .	467
130.2 feedback . . . . .	468
<b>13176.6 - OSC: Jog Modes</b>	<b>469</b>
131.1 Jog Modes . . . . .	469
131.2 Scrub . . . . .	469
<b>13276.7 - OSC: Automation</b>	<b>471</b>
<b>13376.8 - OSC: Personal Monitoring Control</b>	<b>473</b>
133.1 Setup . . . . .	473
133.2 The OSC commands and feedback for personal monitoring . . . . .	473
<b>13476.9 - OSC: Parameter Types</b>	<b>477</b>
<b>13576.10 - OSC: Selection and Expansion Considerations</b>	<b>479</b>
<b>13676.11 - OSC control for Ardour 4.7 and Prior</b>	<b>481</b>
136.1 Connecting to Ardour via OSC . . . . .	481
136.2 List of OSC messages . . . . .	482
<b>13777 - Controlling Ardour with Mackie Control Devices</b>	<b>495</b>
<b>13877.1 - Devices Using Mackie/Logic Control Protocol</b>	<b>497</b>
138.1 Enabling Mackie Control in Ardour . . . . .	497
138.2 Devices using ipMIDI . . . . .	498
138.3 Connecting control surface and Ardour MIDI ports . . . . .	498
138.4 Customizing your control surface . . . . .	498
138.5 Preparing your device for use with Ardour . . . . .	498
<b>13977.2 - SSL Nucleus</b>	<b>499</b>
139.1 Pre-configuring the Nucleus . . . . .	499
139.2 Connecting the Nucleus . . . . .	499
139.3 Nucleus Design Discussion . . . . .	500
<b>14077.3 - Behringer Devices in Mackie/Logic Control Mode</b>	<b>503</b>
140.1 Behringer BCF-2000 Faders Controller . . . . .	503
140.2 Behringer X-Touch . . . . .	510
140.3 Behringer X-Touch Compact . . . . .	510
140.4 Behringer X-Touch Mini . . . . .	510
<b>14177.4 - What to do if your Device is not Listed</b>	<b>513</b>
<b>14277.5 - Working With Extenders</b>	<b>515</b>
<b>14378 - Generic MIDI</b>	<b>517</b>
<b>14478.1 - Generic MIDI Binding Maps</b>	<b>519</b>
144.1 Creating new MIDI maps . . . . .	520
144.2 A Complete (though muddled) Example . . . . .	523

<b>14578.2 - Generic MIDI Learn</b>	<b>525</b>
145.1 Philosophy . . . . .	525
145.2 Basics . . . . .	525
145.3 Cancelling a Learned MIDI Binding . . . . .	525
145.4 Avoiding work in the future . . . . .	525
<b>14678.3 - Generic MIDI and Encoders</b>	<b>527</b>
<b>14779 - Using the PreSonus FaderPort</b>	<b>529</b>
147.1 Connecting the Faderport . . . . .	529
147.2 Using the Faderport . . . . .	530
<b>14880 - Using the PreSonus FaderPort 8</b>	<b>535</b>
148.1 Connecting the FaderPort 8 . . . . .	535
148.2 Using the FaderPort 8 . . . . .	535
<b>14981 - Using the Ableton Push 2</b>	<b>547</b>
149.1 Connecting the Push 2 . . . . .	548
149.2 Push 2 Configuration . . . . .	548
149.3 Basic Concepts . . . . .	548
149.4 Musical Performance . . . . .	549
149.5 Global Mix . . . . .	549
149.6 Track Mix . . . . .	550
149.7 Scale Selection . . . . .	551
149.8 Specific Button/Knob Functions . . . . .	551
<b>150Part XII - Scripting</b>	<b>553</b>
<b>15182 - Lua Scripting</b>	<b>555</b>
151.1 Preface . . . . .	555
151.2 Overview . . . . .	555
151.3 Integration . . . . .	556
151.4 Managing Scripts . . . . .	556
151.5 Script Layout . . . . .	557
151.6 Accessing Ardour Objects . . . . .	560
151.7 Concepts . . . . .	560
151.8 Current State . . . . .	561
151.9 Examples . . . . .	561
<b>15283 - Lua Bindings Class Reference</b>	<b>565</b>
152.1 ARDOUR:Amp . . . . .	567
152.2 ARDOUR:AudioBackend . . . . .	568
152.3 ARDOUR:AudioBackendInfo . . . . .	571
152.4 ARDOUR:AudioBuffer . . . . .	571
152.5 ARDOUR:AudioEngine . . . . .	572
152.6 ARDOUR:AudioPlaylist . . . . .	574
152.7 ARDOUR:AudioPort . . . . .	576
152.8 ARDOUR:AudioRange . . . . .	577
152.9 ARDOUR:AudioRangeList . . . . .	578
152.10 ARDOUR:AudioRegion . . . . .	579
152.11 ARDOUR: AudioSource . . . . .	583
152.12 ARDOUR: AudioTrack . . . . .	585
152.13 ARDOUR: AudioTrackList . . . . .	592
152.14 ARDOUR: Automatable . . . . .	593
152.15 ARDOUR: AutomatableSequence . . . . .	594

152.16 ARDOUR:AutomationControl . . . . .	594
152.17 ARDOUR:AutomationList . . . . .	596
152.18 ARDOUR:BackendVector . . . . .	598
152.19 ARDOUR:BeatsFramesConverter . . . . .	599
152.20 ARDOUR:BufferSet . . . . .	599
152.21 ARDOUR:ChanCount . . . . .	599
152.22 ARDOUR:ChanMapping . . . . .	601
152.23 ARDOUR:ControlList . . . . .	601
152.24 ARDOUR:ControlListPtr . . . . .	602
152.25 ARDOUR.DSP . . . . .	603
152.26 ARDOUR:DSP:Biquad . . . . .	604
152.27 ARDOUR:DSP:DspShm . . . . .	605
152.28 ARDOUR:DSP:FFTSpectrum . . . . .	606
152.29 ARDOUR:DSP:LowPass . . . . .	607
152.30 ARDOUR:DataType . . . . .	608
152.31 ARDOUR:Delivery . . . . .	609
152.32 ARDOUR:DeviceStatus . . . . .	611
152.33 ARDOUR:DeviceStatusVector . . . . .	611
152.34 ARDOUR:DoubleBeatsFramesConverter . . . . .	612
152.35 ARDOUR:EventList . . . . .	612
152.36 ARDOUR:FileSource . . . . .	613
152.37 ARDOUR:FluidSynth . . . . .	614
152.38 ARDOUR:GainControl . . . . .	615
152.39 ARDOUR:IO . . . . .	617
152.40 ARDOUR:IOProcessor . . . . .	619
152.41 ARDOUR:InterThreadInfo . . . . .	620
152.42 ARDOUR:Location . . . . .	621
152.43 ARDOUR:LocationList . . . . .	623
152.44 ARDOUR:Locations . . . . .	623
152.45 ARDOUR.LuaAPI . . . . .	624
152.46 ARDOUR:LuaAPI:Vamp . . . . .	627
152.47 ARDOUR:LuaOSC:Address . . . . .	629
152.48 ARDOUR:LuaProc . . . . .	629
152.49 ARDOUR:LuaTableRef . . . . .	631
152.50 ARDOUR:Meter . . . . .	632
152.51 ARDOUR:MeterSection . . . . .	632
152.52 ARDOUR:MetricSection . . . . .	633
152.53 ARDOUR:MidiBuffer . . . . .	633
152.54 ARDOUR:MidiModel . . . . .	634
152.55 ARDOUR:MidiModel:DiffCommand . . . . .	634
152.56 ARDOUR:MidiModel:NoteDiffCommand . . . . .	635
152.57 ARDOUR:MidiPlaylist . . . . .	636
152.58 ARDOUR:MidiPort . . . . .	639
152.59 ARDOUR:MidiRegion . . . . .	640
152.60 ARDOUR:MidiSource . . . . .	644
152.61 ARDOUR:MidiTrack . . . . .	646
152.62 ARDOUR:MidiTrackList . . . . .	653
152.63 ARDOUR:MonitorProcessor . . . . .	654
152.64 ARDOUR:MusicFrame . . . . .	657
152.65 ARDOUR:MuteControl . . . . .	657
152.66 ARDOUR:NotePtrList . . . . .	659
152.67 ARDOUR:OwnedPropertyList . . . . .	660
152.68 ARDOUR:PannerShell . . . . .	660
152.69 ARDOUR:ParameterDescriptor . . . . .	661

152.70 ARDOUR:PeakMeter . . . . .	662
152.71 ARDOUR:PhaseControl . . . . .	664
152.72 ARDOUR:Playlist . . . . .	666
152.73 ARDOUR:Plugin . . . . .	668
152.74 ARDOUR:Plugin:IOPortDescription . . . . .	670
152.75 ARDOUR:PluginControl . . . . .	670
152.76 ARDOUR:PluginInfo . . . . .	672
152.77 ARDOUR:PluginInsert . . . . .	673
152.78 ARDOUR:Port . . . . .	675
152.79 ARDOUR:PortEngine . . . . .	676
152.80 ARDOUR:PortList . . . . .	676
152.81 ARDOUR:PortManager . . . . .	677
152.82 ARDOUR:PortSet . . . . .	678
152.83 ARDOUR:PresentationInfo . . . . .	679
152.84 ARDOUR:PresetRecord . . . . .	680
152.85 ARDOUR:PresetVector . . . . .	680
152.86 ARDOUR:Processor . . . . .	681
152.87 ARDOUR:ProcessorList . . . . .	682
152.88 ARDOUR:ProcessorVector . . . . .	683
152.89 ARDOUR:Progress . . . . .	684
152.90 ARDOUR:Properties:BoolProperty . . . . .	684
152.91 ARDOUR:Properties:FloatProperty . . . . .	684
152.92 ARDOUR:Properties:FrameposProperty . . . . .	684
152.93 ARDOUR:PropertyChange . . . . .	685
152.94 ARDOUR:PropertyList . . . . .	685
152.95 ARDOUR:RCConfiguration . . . . .	685
152.96 ARDOUR:ReadOnlyControl . . . . .	711
152.97 ARDOUR:Readable . . . . .	712
152.98 ARDOUR:Region . . . . .	712
152.99 ARDOUR:RegionFactory . . . . .	716
152.100ARDOUR:RegionList . . . . .	716
152.101ARDOUR:RegionListPtr . . . . .	717
152.102ARDOUR:RegionMap . . . . .	718
152.103ARDOUR:RegionVector . . . . .	718
152.104ARDOUR:Route . . . . .	719
152.105ARDOUR:Route:ProcessorStreams . . . . .	725
152.106ARDOUR:RouteGroup . . . . .	725
152.107ARDOUR:RouteGroupList . . . . .	728
152.108ARDOUR:RouteList . . . . .	728
152.109ARDOUR:RouteListPtr . . . . .	729
152.110ARDOUR:Session . . . . .	729
152.111ARDOUR:SessionConfiguration . . . . .	735
152.112ARDOUR:SessionObject . . . . .	744
152.113ARDOUR:SessionObjectPtr . . . . .	745
152.114ARDOUR:SideChain . . . . .	745
152.115ARDOUR:Slavable . . . . .	747
152.116ARDOUR:SlavableAutomationControl . . . . .	747
152.117ARDOUR:SoloControl . . . . .	749
152.118ARDOUR:SoloIsolateControl . . . . .	751
152.119ARDOUR:SoloSafeControl . . . . .	753
152.120ARDOUR:Source . . . . .	755
152.121ARDOUR:SourceList . . . . .	757
152.122ARDOUR:Stripable . . . . .	758
152.123ARDOUR:StripableList . . . . .	761

152.124ARDOUR:Tempo	761
152.125ARDOUR:TempoMap	762
152.126ARDOUR:TempoSection	763
152.127ARDOUR:Track	763
152.128ARDOUR:UnknownProcessor	770
152.129ARDOUR:VCA	772
152.130ARDOUR:VCAList	775
152.131ARDOUR:VCAManager	776
152.132ARDOUR:WeakAudioSourceList	777
152.133ARDOUR:WeakRouteList	777
152.134ARDOUR:WeakSourceList	778
152.135ArdourUI	779
152.136ArdourUI:ArdourMarker	779
152.137ArdourUI:ArdourMarkerList	779
152.138ArdourUI:AxisView	780
152.139ArdourUI:Editor	780
152.140ArdourUI:MarkerSelection	786
152.141ArdourUI:RegionSelection	787
152.142ArdourUI:RegionView	788
152.143ArdourUI:RouteTimeAxisView	788
152.144ArdourUI:RouteUI	788
152.145ArdourUI:Selectable	789
152.146ArdourUI:Selection	789
152.147ArdourUI:SelectionList	790
152.148ArdourUI:StripableTimeAxisView	791
152.149ArdourUI:TimeAxisView	791
152.150ArdourUI:TimeAxisViewItem	791
152.151ArdourUI:TimeSelection	791
152.152ArdourUI:TrackSelection	792
152.153ArdourUI:TrackViewList	793
152.154ArdourUI:TrackViewStdList	794
152.155C:ByteArray	795
152.156C:DoubleVector	795
152.157C:FloatArray	796
152.158C:FloatArrayVector	796
152.159C:FloatVector	797
152.160C:Int64List	798
152.161C:IntArray	798
152.162C:StringList	799
152.163C:StringVector	800
152.164Cairo:Context	800
152.165Cairo:ImageSurface	809
152.166Cairo:PangoLayout	810
152.167Evoral:Beats	812
152.168Evoral:Control	812
152.169Evoral:ControlEvent	813
152.170Evoral:ControlList	813
152.171Evoral:ControlSet	815
152.172Evoral:Event	815
152.173Evoral:NotePtr	815
152.174Evoral:Parameter	816
152.175Evoral:ParameterDescriptor	817
152.176Evoral:Range	817
152.177Evoral:Sequence	818

152.178LuaDialog:Dialog . . . . .	818
152.179LuaDialog:Message . . . . .	818
152.180LuaSignal:Set . . . . .	818
152.181PBD . . . . .	819
152.182PBD:Command . . . . .	819
152.183PBD:Configuration . . . . .	820
152.184PBD:Controllable . . . . .	820
152.185PBD:ID . . . . .	821
152.186PBD:IdVector . . . . .	821
152.187PBD:RingBuffer8 . . . . .	822
152.188PBD:RingBufferF . . . . .	823
152.189PBD:RingBufferI . . . . .	823
152.190PBD:Stateful . . . . .	824
152.191PBD:StatefulDestructible . . . . .	824
152.192PBD:StatefulDestructiblePtr . . . . .	825
152.193PBD:StatefulDiffCommand . . . . .	825
152.194PBD:StatefulPtr . . . . .	826
152.195PBD:XMLNode . . . . .	827
152.196Timecode:BBT_TIME . . . . .	827
152.197Timecode:Time . . . . .	827
152.198Vamp:Plugin . . . . .	828
152.199Vamp:Plugin:Feature . . . . .	832
152.200Vamp:Plugin:FeatureList . . . . .	832
152.201Vamp:Plugin:FeatureSet . . . . .	833
152.202Vamp:Plugin:OutputDescriptor . . . . .	834
152.203Vamp:Plugin:OutputList . . . . .	835
152.204Vamp:PluginBase . . . . .	836
152.205Vamp:PluginBase:ParameterDescriptor . . . . .	838
152.206Vamp:PluginBase:ParameterList . . . . .	839
152.207Vamp:RealTime . . . . .	839
152.208os . . . . .	840
152.209enum/Constants . . . . .	840
152.210class Index . . . . .	855
<b>153Part XIII - Appendix</b>	<b>863</b>
<b>1544 - List of Menu Actions</b>	<b>865</b>
154.1 Menu actions . . . . .	865
<b>1555 - Ardour Monitor Modes</b>	<b>881</b>
<b>1566 - Files and Directories Ardour Knows About</b>	<b>883</b>
156.1 Configuration Directory . . . . .	883
156.2 Plugins . . . . .	884
156.3 Project Directory . . . . .	885
<b>1577 - MIDI notes reference</b>	<b>887</b>

## The Ardour Manual

*Introduction to Ardour 1 - Welcome to Ardour*

- 1.1 - About Ardour's documentation
- 1.2 - Ardour Overview
- 1.3 - Why is it called Ardour?
- 1.4 - Why Write a DAW for Linux?
- 1.5 - Isn't This a Really Complicated Program?
- 1.6 - Creating Music with Ardour
- 1.7 - Additional Resources

2 - Ardour Basics

- 2.1 - Starting Ardour
- 2.2 - Understanding Basic Concepts and Terminology
- 2.3 - Using the Mouse
- 2.4 - Basic GUI Operations

3 - Keyboard and Mouse Shortcuts

- 3.1 - Default Keyboard Bindings
- 3.2 - Mnemonic Bindings for Linux
- 3.3 - Mnemonic Bindings for OS X

Ardour Configuration 4 - Ardour Systems

- 4.1 - The Right Computer System for Digital Audio
- 4.2 - The Right Mouse

5 - System Specific Setup

- 5.1 - Ubuntu Linux
- 5.2 - Microsoft Windows
- 5.3 - KDE Plasma 5

6 - I/O Setup

- 6.1 - Connecting Audio and MIDI Devices
- 6.2 - Using More Than One Audio Device
- 6.3 - Monitor Setup in Ardour

7 - Synchronization

- 7.1 - On Clock and Time
- 7.2 - Latency and Latency-Compensation
- 7.3 - Timecode Generators and Slaves
- 7.4 - Overview of all Timecode related settings

8 - Preferences 9 - Session Properties

- 9.1 - Timecode Tab

- 9.2 - *Sync Tab*
- 9.3 - *Fades Tab*
- 9.4 - *Media Tab*
- 9.5 - *Locations Tab*
- 9.6 - *Filenames Tab*
- 9.7 - *Monitoring Tab*
- 9.8 - *Meterbridge Tab*
- 9.9 - *Session Misc Tab*

*10 - Configuring MIDI*

- 10.1 - *MIDI on Linux*
- 10.2 - *MIDI on OS X*

*Ardour's Interface 11 - Ardour's Interface Overview 12 - Main Menu*

- 12.1 - *The Session Menu*
- 12.2 - *The Transport Menu*
- 12.3 - *The Edit Menu*
- 12.4 - *The Region Menu*
- 12.5 - *The Track Menu*
- 12.6 - *The View Menu*
- 12.7 - *The Window Menu*
- 12.8 - *The Help Menu*

*13 - Status Bar 14 - Transport Bar 15 - Transport Clocks 16 - Selection and Punch Clocks 17 - Mini-Timeline*

*18 - Other Toolbar Items 19 - Toolbox 20 - Controls*

- 20.1 - *Zoom Controls*
- 20.2 - *Grid Controls*
- 20.3 - *Edit Point Control*
- 20.4 - *Nudge Controls*

*21 - Ruler 22 - Summary 23 - Editor Lists*

- 23.1 - *The Region List*
- 23.2 - *The Tracks and Busses List*
- 23.3 - *The Snapshot List*
- 23.4 - *The Track and Bus Group List*
- 23.5 - *The Ranges and Marks Lists*

*24 - Favorite Plugins Window 25 - Strips list 26 - Groups list 27 - Mixer Strips*

- 27.1 - *Audio/MIDI Mixer Strips*
- 27.2 - *Audio/MIDI Busses Mixer Strips*
- 27.3 - *VCA Mixer Strips*

- 27.4 - Master Bus Strip

## 28 - Editor Tracks

- 28.1 - Audio Track Controls
- 28.2 - MIDI Track Controls
- 28.3 - Bus Controls

## 29 - Track and Bus Groups 30 - Monitor Section Sessions & Tracks 31 - Sessions

- 31.1 - What's in a Session?
- 31.2 - Where Are Sessions Stored?
- 31.3 - New/Open Session Dialog
- 31.4 - Renaming a Session
- 31.5 - Session Metadata
- 31.6 - Backup and Sharing of Sessions

## 32 - Tracks

- 32.1 - Track Types
- 32.2 - Adding Tracks, Busses and VCAs
- 32.3 - Controlling Track Ordering
- 32.4 - Track Context Menu

## 33 - Controlling Track Appearance

- 33.1 - Layering Display
- 33.2 - Track Color
- 33.3 - Track Height
- 33.4 - Waveform display

## 34 - Importing and Exporting Session Data

- 34.1 - Import Dialog
- 34.2 - Supported File Formats
- 34.3 - Adding Pre-existing Material
- 34.4 - Copying Versus Linking
- 34.5 - Searching for Files Using Tags
- 34.6 - Stem Exports

## 35 - File and Session Management and Compatibility

- 35.1 - Session Templates
- 35.2 - Snapshots
- 35.3 - Cleaning Up Sessions
- 35.4 - Interchange with other DAWs

## Playback & Recording 36 - Playing Back Track Material

- 36.1 - Controlling Playback

- 36.2 - *Using Key Bindings*

*37 - Audio Recording*

- 37.1 - *Monitoring*
- 37.2 - *Latency Considerations*
- 37.3 - *Monitor Signal Flow*

*38 - Punch Recording Modes*

- 38.1 - *Working With Markers*
- 38.2 - *Creating Location Markers*
- 38.3 - *Creating Range Markers*
- 38.4 - *Moving Markers*
- 38.5 - *The Loop Range*
- 38.6 - *Marker Context Menu*
- 38.7 - *Punch Range*

*Editing 39 - Navigating the Editor 40 - Editing Basics*

- 40.1 - *Working With Regions*
- 40.2 - *Region Naming*
- 40.3 - *Corresponding Regions Selection*
- 40.4 - *Region Context Menu*
- 40.5 - *Common Region Edit Operations*
- 40.6 - *Copy Regions*
- 40.7 - *Move Regions With the Mouse*
- 40.8 - *Align (Spot) Regions*

*41 - Edit Mode and Tools*

- 41.1 - *Editing Clocks*
- 41.2 - *Which Regions Are Affected?*

*42 - Making Selections 43 - Editing Regions and Selections*

- 43.1 - *Trimming Regions*
- 43.2 - *Push/Pull Trimming*
- 43.3 - *Stretching*
- 43.4 - *Separate Under*
- 43.5 - *Strip Silence from Audio Regions*
- 43.6 - *Insert/Remove Time*
- 43.7 - *Region Properties*

*44 - Fades and Crossfades 45 - Gain Envelopes 46 - Playlists*

- 46.1 - *Understanding Playlists*
- 46.2 - *Playlist Operations*

- 46.3 - Playlist Usecases

#### 47 - Rhythm Ferret MIDI

- 48 - MIDI Editing
- 49 - Creating MIDI Tracks
- 50 - Creating MIDI Regions
- 51 - Adding New Notes
- 52 - Changing Note Properties
- 53 - Handling Overlapping Notes
- 54 - Note Cut, Copy and Paste
- 55 - Note Selection
- 56 - Quantizing MIDI
- 57 - Step Entry
- 58 - Patch Change
- 59 - Independent and Dependent MIDI Region Copies
- 60 - Transposing MIDI
- 61 - Automating MIDI-Pitch bending and aftertouch
- 62 - Transforming MIDI-Mathematical operations
- 63 - MIDI List Editor
- 64 - MIDI Tracer
- 65 - MIDI Recording
- 66 - MIDI Scene Automation

#### Arranging 67 - Time, Tempo and Meter

- 67.1 - Tempo and Meter
- 67.2 - Techniques for Working with Tempo and Meter

#### Mixing 68 - Basic Mixing

- 68.1 - Metering in Ardour
- 68.2 - Signal Routing
- 68.3 - Aux Sends
- 68.4 - Comparing Aux Sends and Subgroups
- 68.5 - External Sends
- 68.6 - Inserts
- 68.7 - Subgrouping
- 68.8 - Patchbay
- 68.9 - Track/Bus Signal Flow
- 68.10 - Sidechaining
- 68.11 - Muting and Soloing

- [68.12 - Panning](#)
- [68.13 - Mono Panner](#)
- [68.14 - Balance Control](#)
- [68.15 - Stereo Panner](#)
- [68.16 - VBAP Panner](#)

### [69 - Plugin and Hardware Inserts](#)

- [69.1 - Working With Plugins](#)
- [69.2 - Processor Box](#)
- [69.3 - Plugin Manager](#)
- [69.4 - Managing Plugin Presets](#)
- [69.5 - Working with Ardour-built Plugin Editors](#)
- [69.6 - Plugins Bundled With Ardour](#)
- [69.7 - Getting More Plugins](#)
- [69.8 - Using Windows VST Plugins on Linux](#)

### [70 - Automation](#)

- [70.1 - Automation Nomenclature](#)
- [70.2 - Automation Modes](#)
- [70.3 - Automation Lanes](#)
- [70.4 - Automation Curves](#)
- [70.5 - Controlling a Track with Automation](#)

### [71 - Mixdown](#)

- [71.1 - Export Dialog](#)
- [71.2 - Export Format Profiles](#)

### [Video](#)

- [72 - Video Timeline and Monitoring](#)
- [73 - Video Timeline Setup](#)
- [74 - Transcoding, Formats & Codecs](#)
- [75 - Workflow & Operations](#)

### [Control Surfaces](#) [76 - Controlling Ardour with OSC](#)

- [76.1 - OSC: Controlling Ardour with OSC](#)
- [76.2 - OSC: Using the Setup Dialog](#)
- [76.3 - OSC: Querying Ardour](#)
- [76.4 - OSC: Feedback](#)
- [76.5 - OSC: Feedback and Strip-types Values](#)
- [76.6 - OSC: Jog Modes](#)
- [76.7 - OSC: Automation](#)

- 76.8 - OSC: Personal Monitoring Control
- 76.9 - OSC: Parameter Types
- 76.10 - OSC: Selection and Expansion Considerations
- 76.11 - OSC control for Ardour 4.7 and Prior

## 77 - Controlling Ardour with Mackie Control Devices

- 77.1 - Devices Using Mackie/Logic Control Protocol
- 77.2 - SSL Nucleus
- 77.3 - Behringer Devices in Mackie/Logic Control Mode
- 77.4 - What to do if your Device is not Listed
- 77.5 - Working With Extenders

## 78 - Generic MIDI

- 78.1 - Generic MIDI Binding Maps
- 78.2 - Generic MIDI Learn
- 78.3 - Generic MIDI and Encoders

## 79 - Using the PreSonus FaderPort 80 - Using the PreSonus FaderPort 8 81 - Using the Ableton Push 2 Scripting

- 82 - Lua Scripting
- 83 - Lua Bindings Class Reference

## Appendix

- 84 - List of Menu Actions
- 85 - Ardour Monitor Modes
- 86 - Files and Directories Ardour Knows About
- 87 - MIDI notes reference

## Part I - Introduction to Ardour

### 1 - Welcome to Ardour

#### 1.1 - About Ardour's documentation

##### Conventions Used In This Manual

This section covers some of the typographical and language conventions used in this manual.

##### Keyboards and Modifiers

Keyboard bindings are shown like this: s or x.

x means “press the key, keep it pressed and then also press the x key”.

Combinations such as e may be seen, which means “hold down the key *and* the key, and then, while keeping them both down, press the e key”.

Different platforms have different conventions for which modifier key (Control or Command) to use as the primary or most common modifier. When viewing this manual from a machine identifying itself as running OS X, Cmd will be seen where appropriate (for instance in the first example above). On other machines Ctrl will be seen instead.

### **Mouse Buttons**

*Mouse buttons* are referred to as Left, Middle and Right. Ardour can use additional buttons, but they have no default behaviour in the program.

### **Mouse Click Modifiers**

Many editing functions are performed by clicking the mouse while holding a modifier key, for example Left.

### **Mouse Wheel**

Some GUI elements can optionally be controlled with the mouse wheel when the pointer is hovering over them. The notation for mouse wheel action is .

### **Context-click**

The term context-click is used to indicate a Right-click on a particular element of the graphical user interface. Although right-click is the common, default way to do this, there are other ways to accomplish the same thing—this term refers to any of them, and the result is always that a menu specific to the item clicked on will be displayed.

### **“The Pointer”**

When the manual refers to the “pointer”, it means the on-screen representation of the mouse position or the location of a touch action if touch interface is being used.

### **Other User Input**

Ardour supports hardware controllers, such as banks of faders, knobs, or buttons.

### **Menu Items**

Menu items are indicated like this: Top > Next > Deeper. Each “>”-separated item indicates one level of a nested menu or sub-menu.

### **OSC Messages**

OSC messages, whether sent or received, are displayed like this: /transport\_stop.

## Preference/Dialog Options

Choices in various dialogs, notably the Preferences and Properties dialog, are indicated thus:

Edit > Preferences > Audio > Some Option.

Each successive item indicates either a menu, sub-menu, or a tabbed dialog navigation. The final item is the one to choose or select.

If an option is deselected, it will look like this:

Edit > Preferences > Audio > Some other Option.

## User Input

Some dialogs or features may require the user to input data such as this. In rare cases, certain operations will be required to be performed at the command line of the operating system:

cat /proc/cpuinfo sleep 3600 ping www.google.com .. rubric:: Program Output

name program-output

Important messages from Ardour or other programs will be displayed like this.

## Notes

Important notes about things that might not otherwise be obvious are shown in this format.

## Warnings

Hairy issues that might cause things to go wrong, lose data, impair sound quality, or eat your proverbial goldfish, are displayed in this way.

## 1.2 - Ardour Overview

Ardour is a professional digital workstation for working with audio and MIDI.

### Ardour is meant for...

#### Audio Engineers

Ardour's core user group: people who want to record, edit, mix and master audio and MIDI projects. When you need complete control over your tools, when the limitations of other designs get in the way, when you plan to spend hours or days working on a session, Ardour is there to make things work the way you want them to.

#### Musicians

Being the best tool to record talented performers on actual instruments has always been a top priority for Ardour. Rather than being focused on electronic and pop music idioms, Ardour steps out of the way to encourage the creative process to remain where it always has been: a musician playing a carefully designed and well built instrument.

## Soundtrack Editors

Sample accurate sync and shared transport control with video playback tools allows Ardour to provide a fast and natural environment for creating and editing soundtracks for film and video projects.

## Composers

Arrange audio and MIDI using the same tools and same workflow. Use external hardware synthesizers or software instruments as sound sources. From sound design to electro-acoustic composition to dense multitrack MIDI editing, Ardour can help.

## Ardour features...

### Audio and MIDI Multi-Track Recording and Editing

Any number of tracks and busses. Non-linear editing. Non-destructive (and destructive!) recording. Any bit depth, any sample rate. Dozens of file formats.

### Plugins with Full Sample Accurate Automation

AudioUnit, LV2, LinuxVST and LADSPA formats. FX plugins. Software instruments. MIDI processors. Automate any parameters. Physically manipulate them via control surfaces. Distribute processing across as many (or as few) cores as you want.

### Transport Sync and External Control Surfaces

Best-in-industry sync to MIDI timecode and LTC. Send and receive MIDI Machine Control. Sync with JACK transport and MIDI clock. Dedicated Mackie Control protocol support, pre-defined mappings for many MIDI controllers plus dynamic MIDI learn. Use OSC to drive almost any operation in Ardour.

### Powerful Anywhere-to-Anywhere Signal Routing

Complex signal flows are simple and elegant. Inputs and outputs connect to your hardware and/or other applications. Use sends, inserts and returns freely. Connections can be one-to-many, many-to-one or many-to-many. Tap signal flows at any point. If you can't connect in the way you want with Ardour, it probably can't be done.

### Video Timeline

Import a single video and optionally extract the soundtrack from it. Display a frame-by-frame (thumbnail) timeline of the video. Use a Video-monitor window, or full-screen display, of the imported video in sync with any of the available ardour timecode sources. Lock audio-regions to the video: Move audio-regions with the video at video-frame granularity. Export the video, cut start/end, add blank frames and/or mux it with the soundtrack of the current-session.

### 1.3 - Why is it called Ardour?

The name “Ardour” came from considerations of how to pronounce the acronym HDR. The most obvious attempt sounds like a vowel-less “harder” and it then was then a short step to an unrelated but slightly homophonic word:

- ardour *n* 1: a feeling of strong eagerness (usually in favor of a person or cause); “they were imbued with a revolutionary ardor”; “he felt a kind of religious zeal” [syn: ardor, elan, zeal]
- 2: intense feeling of love [syn: ardor]
- 3: feelings of great warmth and intensity; “he spoke with great ardor” [syn: ardor, fervor, fervour, fervency, fire, fervidness]

Given the work required to develop Ardour, and the personality of its primary author, the name seemed appropriate even without the vague relationship to HDR.

Years later, another interpretation of “Ardour” appeared, this time based on listening to non-native English speakers attempt to pronounce the word. Rather than “Ardour”, it became “Our DAW”, which seemed poetically fitting for a Digital Audio Workstation whose source code and design belongs to a group of collaborators.

### 1.4 - Why Write a DAW for Linux?

There are already a number of excellent digital audio workstations. To mention just a few: ProTools, Nuendo, Samplitude, Digital Performer, Logic, Cubase (SX), Sonar, along with several less well known systems such as SADIE, SAWStudio and others. Each of these programs has its strengths and weaknesses, although over the last few years most of them have converged on a very similar set of core features. However, each of them suffers from two problems when seen from the perspective of Ardour’s development group:

- they do not run natively on Linux
- they are not available in source code form, making modifications, improvements, bugfixes by technically inclined users or their friends or consultants impossible.

It is fairly understandable that most existing proprietary DAWs do not run on Linux, given the rather small (but growing) share of the desktop market that Linux has. However, when surveying the landscape of “popular operating systems”, we find:

- older versions of Windows: plagued by abysmal stability and appalling security
- newer versions of Windows seem stable but still suffer from security problems
- OS X: a nice piece of engineering that is excellent for audio work but only runs on proprietary hardware and still lacks the flexibility and adaptability of Linux.

Security matters today, and will matter more in the future as more and more live or semi-live network based collaborations take place.

Let’s contrast this with Linux, an operating system which:

- can stay up for months (or even years) without issues
- is endlessly configurable down to the tiniest detail
- is not owned by any single corporate entity, ensuring its life and direction are not intertwined with that of a company (for a contrary example, consider BeOS)
- is fast and efficient
- runs on almost any computing platform ever created, including old “slow” systems and new “tiny” systems (e.g. Raspberry Pi)

- is one of the most secure operating systems “out of the box”

More than anything, however, Ardour’s primary author uses Linux and wanted a DAW that ran there.

Having written a DAW for Linux, it turned out to be relatively easy to port Ardour to OS X, mostly because of the excellent work done by the JACK OS X group that ported JACK to OS X.

### **1.5 - Isn't This a Really Complicated Program?**

There is no point in pretending that Ardour is a simple, easy to use program. The development group has worked hard to try to make simple things reasonably easy, common tasks quick, and hard and/or uncommon things possible. There is no doubt that there is more to do in this area, as well as polishing the user interface to improve its intuitiveness and work flow characteristics.

At the same time, multi-track, multi-channel, non-linear, non-destructive audio editing is a far from simple process. Doing it right requires not only a good ear, but a solid appreciation of basic audio concepts and a robust mental model/metaphor of what one is doing. Ardour is not a simple “audio recorder”—it can certainly be used to record stereo (or even mono) material in a single track, but the program has been designed around much richer capabilities than this.

Some people complain that Ardour is not “intuitive” to use—its lead developer has [some thoughts on that](#).

### **1.6 - Creating Music with Ardour**

Ardour can be used in many different ways, from extremely simple to extremely complex. Many projects can be handled using the following kind of workflow:

#### **Stage 1: Creating The Project**

The first step is to create a new session, or open an existing one. A session consists of a folder containing a session file that defines all the information about the session. All media files used by the session are usually stored within the session folder.

More details on sessions can be found in *Sessions* chapter.

#### **Stage 2: Creating and Importing Audio and MIDI Data**

Once a session has been created, it will be necessary to add some audio and/or MIDI material to it—which can be done in one of 3 ways:

- Record incoming audio or MIDI data, either via audio or MIDI hardware connected to the computer, or from other applications
- Create new MIDI data using the mouse and/or various dialogs
- Import existing media files into the session

MIDI recordings consist of performance data (“play note X at time T”) rather than actual sound. As a result, they are more flexible than actual audio, since the precise sound that they will generate when played depends on where the MIDI data is sent to. Two different synthesizers may produce very different sounds in response to the same incoming MIDI data.

Audio recordings can be made from external instruments with electrical outputs (keyboards, guitars, etc.), or via microphones or other sound capturing equipment.

Ardour can use the JACK Audio Connection Kit for all audio and MIDI I/O, making recording audio/MIDI from other applications fundamentally identical to recording audio/MIDI from audio/MIDI hardware.

### Stage 3: Editing and Arranging

Once there is material within the session, it can be arranged in time. This is done in one of the two main windows of Ardour: the Editor window.

Audio/MIDI data appears in chunks called regions, which are arranged into horizontal lanes called tracks. Tracks are stacked vertically in the Editor window. Regions can be copied, shortened, moved, and deleted without changing the actual data stored in the session at all—Ardour is a non-destructive editor. (Almost) nothing done while editing will ever modify the files stored on disk (with the exception of the session file itself).

Many transformations can be done to the contents of regions, again without altering anything on disk. It is possible to alter, move, delete and remove silence from audio regions, for example.

MIDI regions can also be copied, moved, shortened, or deleted without altering the MIDI files, though any edit like adding, suppressing or moving *notes* inside a region results in a modification of the underlying MIDI file.

### Stage 4: Mixing and Adding Effects

Once the arrangement of the session mostly complete, the next step is the mixing phase. Mixing is a broad term to cover the way the audio signals that the session generates during playback are processed and added together into a final result that is actually heard. It can involve altering the relative levels of various parts of the session, adding effects that improve or transform certain elements, and others that bring the sound of the whole session to a new level.

Ardour allows automation of changes to any mixing parameters (such as volume, panning, and effects controls)—it will record the changes made over time, using a mouse or keyboard or some external control device, and can play back those changes later. This is very useful because often the settings needed will vary in one part of a session compared to another—rather than using a single setting for the volume of a track, it may need increases followed by decreases (for example, to track the changing volume of a singer). Using automation can make all of this relatively easy.

### Stage 5: Export

Once the arrangement and mix of the session is finalized, a single audio file that contains a ready-to-listen to version of the work is usually desired. Ardour allows the exporting of audio files in a variety of formats (simultaneously in some cases). This exported file would typically be used in creating a CD, or be the basis for digital distribution of the work.

Of course it is sometimes desirable to export material that isn't finished yet—for example, to give a copy to another party to mix on their own system. Ardour allows exporting as much of a session as desired, at any time, in any supported format.

## 1.7 - Additional Resources

In addition to this documentation, there are a variety of other resources:

- the [Ardour release notes](#)
- the [Ardour Forums](#)
- information about [Ardour Support](#) via mailing lists and IRC (chat)

The IRC channels in particular are where most of the day-to-day development and debugging is done, and there are plenty of experienced users to help if problems are encountered when using Ardour.

Please be prepared to hang around for a few hours, the chat is usually busiest from 19:00 UTC to 04:00 UTC. It is best to keep one's IRC client window open if possible, so that a belated answer can be seen.

## 2 - Ardour Basics

### 2.1 - Starting Ardour

There are several ways of starting Ardour, which may vary depending on which platform it is being used on:

- by double-clicking the Ardour icon in the platform's file manager (e.g. Nautilus on Linux, Finder on OS X)
- by double-clicking on an Ardour session file in the platform's file manager
- on Linux, Ardour can also be started via the command line (see below)

When Ardour is run for the very first time, a special dialog is displayed that will ask several questions about the system's setup. The questions will not be asked again, but the choices thus made can always be modified via the Edit > Preferences dialog.

If JACK is needed, in general, it is sensible to start it *before* Ardour is run. Though this is not strictly necessary, it will provide more control and options over JACK's operation. JACK can be started through the CLI of a terminal, or by using a GUI program, like [QjackCtl](#) or [Cadence](#).

If Ardour is opened without specifying an existing session, it will display the Session > New... dialog and the Audio/MIDI Setup dialog. See *New/Open Session Dialog* for a description of those dialogs.

#### Starting Ardour From the Command Line (Linux)

Like (almost) any other program on Linux, Ardour can be started on the command line. Type the following command in a terminal window:

Ardour5 To start Ardour with an existing session, use:

Ardour5 */path/to/session* Replace /path/to/session with the actual path of the session. Either the session folder or any session file inside the folder can be specified, including snapshots.

To start Ardour with a new, named session, use:

Ardour5 -N */path/to/session ..* rubric:: 2.2 - Understanding Basic Concepts and Terminology

```
name understanding-basic-concepts-and-terminology
class clear
```

In order to fully grasp the terms used in Ardour (and this manual), it is necessary to understand what things like sessions, tracks, busses, regions and so on—as used in Ardour—are.

## Sessions

An Ardour session is a container for an entire project. A session may contain an arbitrary number of tracks and busses consisting of audio and MIDI data, along with information on processing those tracks, a mix of levels, and everything else related to the project. A session might typically contain a song, an entire album, or a complete live recording.

Ardour sessions are kept in directories; these directories contain one or more session files, some or all of the audio and MIDI data, and a number of other state files that Ardour requires. The session file describes the structure of the session, and holds automation data and other details.

Ardour's session file is written in XML format, which is advantageous as it is *somewhat* human-readable and human-editable in a crisis. Sound files are stored in one of a number of optional formats, and MIDI files as SMF.

It is also possible for Ardour sessions to reference sound and MIDI files outside the session directory, to conserve disk space and avoid unnecessary copying if the data is available elsewhere on the disk.

Ardour has a single current session at all times; if Ardour is started without specifying one, it will offer to load or create one.

More details can be found in the *Sessions* chapter.

## Tracks

A track is a concept common to most DAWs, and also used in Ardour. Tracks can record audio or MIDI data to disk, and then replay it with processing. They also allow the audio or MIDI data to be edited in a variety of different ways.

In a typical pop production, one track might be used for the kick drum, another for the snare, more perhaps for the drum overheads and others for bass, guitars and vocals.

Ardour can record to any number of tracks at one time, and then play those tracks back. On playback, a track's recordings may be processed by any number of plugins, panned, and/or its level altered to achieve a suitable mix.

A track's type is really only related to the type of data that it stores on disk. It is possible, for example, to have a MIDI track with a synthesizer plugin which converts MIDI to audio. Even though the track remains MIDI (in the sense that its on-disk recordings are MIDI), its output may be audio-only.

More details can be found in the *Tracks* chapter.

## Busses

Busses are another common concept in both DAWs and hardware mixers. They are similar in many ways to tracks; they process audio or MIDI, and can run processing plugins. The only difference is that their input is obtained from other tracks or busses, rather than from disk.

A bus might typically be used to collect together the outputs of related tracks. Consider, for example, a three track recording of a drum kit; given kick, snare and overhead tracks, it may be helpful to connect the output of each to a bus called "drums", so that the drum kit's level can be set as a unit, and processing (such as equalization or compression) can be applied to the mix of all the tracks. Such busses are also called groups.

## Regions

A track may contain many segments of audio or MIDI. Ardour contains these segments in things called regions, which are self-contained snippets of audio or MIDI data. Any recording pass, for example, generates a region on each track that is enabled for recording. Regions can be subjected to many editing operations; they may be moved around, split, trimmed, copied, and so on.

More details can be found at [Working With Regions](#).

## Playlists

The details of what exactly each track should play back is described by a playlist. A playlist is simply a list of regions; each track always has an active playlist, and can have other playlists which can be switched in and out as required.

More details can be found in the *Playlists* chapter.

## Plugins

Ardour allows processing audio and MIDI using any number of plugins. These are external pieces of code, commonly seen as VST plugins on Windows or AU plugins on Mac OS X. Ardour supports the following plugin standards:

LADSPA	the first major plugin standard for Linux. Many LADSPA plugins are available, mostly free and open-source.
LV2	the successor to LADSPA. Lots of plugins have been ported from LADSPA to LV2, and also many new plugins written.
VST	Ardour supports VST plugins that have been compiled for Linux.
AU	Mac OS X versions of Ardour support AudioUnit plugins.

Ardour has some support for running Windows VST plugins on Linux, but this is rather complicated, extremely difficult for the Ardour developers to debug, and generally unreliable, as it requires running a large amount of Windows code in an emulated environment. If it is at all possible, it is strongly advisable to use native LADSPA, LV2 or Linux VST plugins on Linux, or AU on Mac OS X.

More details can be found at [Working With Plugins](#).

## 2.3 - Using the Mouse

### Clicking

Throughout this manual, the term click refers to the act of pressing and releasing the Left mouse button. This action is used to select objects, activate buttons, turn choices on and off, pop up menus and so forth. On touch surfaces, it also corresponds to a single, one-finger tap on the GUI.

### Right Clicking

The term right-click refers to the act of pressing and releasing the Right mouse button. This action is used to pop up context menus (hence the term “context click”, which will also be seen). It is also used by default in combination with the shift key to delete objects within the editor window.

Some mice designed for use with Mac OS X may have only one button. By convention, pressing and holding the Control key while clicking is interpreted as a right-click by many applications.

### Middle Clicking

A middle-click refers to the act of pressing and releasing the Middle mouse button. Not all mice have a middle click button (see the *Mouse* chapter for details). Sometimes the scroll wheel acts as a clickable middle button. This action is used for time-constrained region copying and mapping MIDI bindings.

Internally, your operating system may identify the mouse buttons as Button1, Button2, and Button3, respectively. It may be possible to invert the order of buttons to accommodate left-handed users, or to re-assign them arbitrarily. This manual assumes the canonical order.

### Double Clicking

A double click refers to two rapid press/release cycles on the leftmost mouse button. The time interval between the two actions that determines whether this is seen as two clicks or one double click is controlled by your system preferences, not by Ardour.

### Dragging

A drag primarily refers to the act of pressing the leftmost mouse button, moving the mouse with the button held down, and then releasing the button. On touch surfaces, this term also corresponds to a single one-finger touch-move-release action.

Ardour also uses the middle mouse button for certain kinds of drags, which will be referred to as a middle-drag.

### Modifiers

There are many actions in Ardour that can be carried out using a mouse button in combination with a modifier key. When the manual refers to Left, it means that you should first press the key, carry out a left click while it is held down, and then finally release the key.

Available modifiers depend on your platform:

#### Linux Modifiers

- Ctrl (Control)
- Shift
- Alt
- Win (Super/Windows)

#### OS X Modifiers

- Cmd (Command, “windmill”)
- Ctrl (Control)
- Alt (Option)
- Shift

### Scroll Wheel

Ardour can make good use of a scroll wheel on the mouse (assuming it has one), which can be utilized for a variety of purposes. Scroll wheels generate vertical scroll events, (ScrollUp) and (ScrollDown). Some also emit horizontal events, (ScrollLeft) and (ScrollRight).

When appropriate, Ardour will differentiate between these two different scroll axes. Otherwise it will interpret ScrollDown and ScrollLeft as equivalent and similarly interpret ScrollUp and ScrollRight as equivalent.

Typically, scroll wheel input is used to adjust continuous controls such as faders and knobs, or to scroll vertically or horizontally inside a window.

## 2.4 - Basic GUI Operations

By default, Ardour will show helpful tooltips about the purpose and use of each GUI element if the pointer is positioned over it and hovered there for a short while. These little pop-up messages can be a good way to discover the purpose of many aspects of the GUI.

Pop-ups can also be distracting for experienced users, who may wish to disable them via Edit > Preferences > GUI > Show tooltip if mouse hovers over a control.

### Selection Techniques

Ardour follows the conventions used by most other computer software (including other DAWs) for selecting objects in the GUI.

#### Selecting individual objects

Clicking on an object (sometimes on a particular part of its on-screen representation) will select the object, and deselect other similar objects.

#### Selecting multiple (similar) objects

A left-click on an object toggles its selected status, so using left on a series of objects will select (or deselect) each one of them. A completely arbitrary set of selections can be constructed with this technique.

#### Selecting a range of objects

In cases where the idea of “select all objects between this one and that one” makes sense, select one object and then left-click on another to select both of them as well as all objects in between.

#### Time range selection

To select a time range in the Editor, Left-click and drag the mouse. A Left drag then lets you create other ranges and a left-click extends a range to cover a wider area.

#### Selection Undo

The set of objects (including time range) that are selected at any one time is known as the selection. Each time an object is selected or deselected, the new selection is stored in an undo/redo stack. This stack is cleared each time the content of the timeline changes.

If a complex selection has been built up and then accidentally cleared it, choosing Edit > Undo Selection Change will restore the previous selection. If a selection is undone and a return to the state before the undo is desired, choosing Edit > Redo Selection Change will take the selection back to where it was before Edit > Undo Selection Change was chosen.

## Cut and Paste Operations

The clipboard is a holder for various kinds of objects (regions, control events, plugins) that is used during cut-and-paste operations.

### Cut

A cut operation removes selected objects and places them in the clipboard. The existing contents of the clipboard are overwritten. The default key binding is x.

### Copy

A copy of the selected objects are placed in clipboard. There is no effect on the selected objects themselves. The existing contents of the clipboard are overwritten. The default key binding is c.

### Paste

The current contents of the clipboard are pasted (inserted) into the session, using the current edit point as the destination. The contents of the clipboard remain unchanged—the same item can be pasted multiple times. The default key binding is v.

## Deleting Objects

Within the Editor window (and to some extent within the Mixer window too), there are several techniques for deleting objects (regions, control points, and more).

### Using the mouse and keyboard

Select the object(s) to be deleted and then press the Del key. This does **not** put the deleted object(s) in the clipboard, so they cannot be pasted elsewhere.

### Using normal cut and paste shortcuts

Select the object(s) and then press x. This puts the deleted object(s) in the clipboard so that they can be pasted elsewhere.

### Using just the mouse

By default, Shift Right will delete the clicked-upon object. Like the Del key, this does **not** put the deleted object(s) in the clipboard.

The modifier and mouse button used for this can be controlled via Edit > Preferences > User Interaction > Delete using .... Any modifier and mouse button combination can be used.

## Undo/Redo for Editing

While editing, it sometimes happens that an unintended change is made, or a choice is made that is later decided to be wrong. All changes to the arrangement of session components (regions, control points) along the timeline can be undone (and redone if necessary).

The default keybindings are Z for Undo and R for Redo. These match the conventions of most other applications that provide undo/redo.

Changes are also saved to the session history file, so that undo/redo is possible even if the session is closed and reopened later, even if Ardour is exited in between.

The maximum number of changes that can be undone can be configured under Edit > Preferences > Misc > Undo. The maximum number of changes stored in the history file is a separate parameter, and can also be set in the same place.

In addition to the normal undo (which works only on actions that change the timeline), there is a visual undo which will revert any command that affects the display of the editor window. Its shortcut is Z. There is also an undo for selection; see “Selection Techniques” above.

## 3 - Keyboard and Mouse Shortcuts

### 3.1 - Default Keyboard Bindings

Almost every available function in Ardour can be bound to a keyboard shortcut (and those few that cannot will usually respond to an *OSC command*). Ardour comes with a rich set of default key bindings for the most commonly used functions.

These bindings strive to be mnemonic, that is, easy and intuitive to remember, and follow widely accepted conventions. As a general rule, the first letter of an operation will be used for as a shortcut, if available. This does not necessarily lead to the best ergonomics for rapid editing—there are alternative binding sets for that—but it does make it simpler for newcomers to remember some of the most useful ones, for example:

S for Region > Edit > Split or P for Transport > Playhead > Playhead to Mouse.

Existing key bindings in menus are listed on the right side of the menu items.

Almost every key binding in Ardour can be looked for and/or changed in Window > Key Bindings (which is bound to K).

Ardour will silently reassign the binding of a key combination that is already in use, possibly removing a standard keyboard shortcut without any warning. This might lead to confusion when asking for help—when the explanation is given in terms of a standard key binding—which will have a completely different effect on the system with the modified key bindings.

The conventions for using modifier keys (, , , etc.) differ among platforms, so different default bindings for each are provided.

### 3.2 - Mnemonic Bindings for Linux

A printable cheat-sheet with the mnemonic bindings for Linux is available for download in [US Letter](#) and [A4](#) paper format.

This set of bindings assumes an en\_US keyboard. However, most if not all bindings will also work on other keyboards when the AltGr key is used to compose those glyphs that are not directly accessible.

**Transport & Recording Control**

destroy last recording	Del
engage record	r
fast forward	→
loop play (the loop range)	l
rewind	←
set playhead position	p
start recording	Space
stop (keep loop/range play)	Space
stop and destroy	Space
toggle auto play	5
toggle auto return	6
toggle click (metronome)	7
toggle playhead follows edits	F
toggle playhead tracking	F
toggle roll	Space
toggle selected track rec-enable	b
toggle selected track solo status	s
transition to reverse	↓
transition to roll	↑

**Session & File Handling**

add track(s) or bus(ses)	n
export session	e
import audio files	i
open a new session	n
open a recent session	o
open an existing session	o
quit	q
save session	s
snapshot session	s
toggle selected track MIDI input	i

## Changing What's Visible

fit tracks vertically	f
move selected tracks down	↓
move selected tracks up	↑
scroll down (page)	PgDn
scroll down (step)	↓
scroll up (page)	PgUp
scroll up (step)	↑
toggle editor window mixer	e
visual undo	z
zoom height to selected region(s)	z
zoom height and time to selected region	z
zoom in	=
zoom out	•

## Window Visibility

toggle locations dialog	l
focus on main clock	÷
maximise editor space	f
switch between editor & mixer window	m
show rhythm ferret window	f
toggle big clock	b
toggle color manager	c
toggle editor window	e
toggle global audio patchbay	p
toggle global midi patchbay	p
toggle key bindings editor	k
toggle preferences dialog	o
toggle preferences dialog	p

## Editing with Edit Point

Most edit functions operate on a single Edit Point (EP). The edit point can be any of: playhead (default), the mouse or an active marker. The choice of edit point (by default) also sets the Zoom Focus.

EP to next region sync	;
EP to prev region sync	'
cycle to next grid snap mode	2
cycle to next zoom focus	1
insert from region list	i
insert time	t
move EP to playhead	
next EP w/marker	'
next EP w/o marker	'
trim back	k
trim front	j
trim region end to edit point	}
trim region start to edit point	{
trim region to end of prev region	j
trim region to start of next region	k
use previous grid unit	3
use next grid unit	4
use previous grid unit	3
use next musical grid unit	4

### Aligning with the Edit Point

Align operations move regions so that their start/end/sync point is at the edit point. Relative operations just align the first region and moves other selected regions to maintain relative positioning.

align end(s)	a
align start(s)	a
align start(s) relative	a
align sync points	a
align sync points (relative)	a
range end to next prev edge	>
range end to next region edge	>
range start to next region edge	<
range start to prev region edge	<

### Edit Point Playback

play edit range	Space
play from EP & return	Space
play selected region(s)	h

## Region Operations

duplicate region (multi)	d
duplicate region (once)	d
export selected region(s)	e
increase region gain	^
move to original position	o
mute/unmute	m
normalize	n
nudge backward	-
nudge forward	•
quantize MIDI notes	q
reduce region gain	&
reverse	r
set fade in length	/
set fade out length	\
set region sync point	v
split	s
toggle fade in active	/
toggle fade out active	\
transpose	t

## Generic Editing

copy	c
cut	x
delete	Del
paste	v
redo	r
undo	z

## Selecting

There are a few functions that refer to an Edit Range. The current edit range is defined using combinations of the possible edit points: playhead, active marker, or mouse.

all after playhead	p
all before playhead	p
all enclosed by edit range	u
all present in edit range	u
convert edit range to range	F6
invert selection	i
select all after EP	e
select all before EP	e
select all in loop range	l
select all in punch range	d
select everything	a
select next track/bus	↓
select previous track/bus	↑

### Defining Loop, Punch Range and Tempo Changes

set loop range from edit range	]
set loop range from region(s)	]
set punch range from edit range	[
set punch range from region(s)	[
set tempo (1 bar) from edit range	0
set tempo (1 bar) from region(s)	9

### 3.3 - Mnemonic Bindings for OS X

A printable cheat sheet for these bindings is available for download.

### Transport & Recording Control

destroy last recording	Del
engage record	r
fast forward	→
loop play (the loop range)	l
rewind	←
set playhead position	p
start recording	space
stop (keep loop/range play)	space
stop and destroy	space
toggle auto play	5
toggle auto return	6
toggle click (metronome)	7
toggle playhead follows edits	f
toggle playhead tracking	f
toggle roll	space
toggle track rec-enable	b
toggle track solo status	s
transition to reverse	↓
transition to roll	↑

## Session & File Handling

add track(s) or bus(ses)	n
export session	e
import audio files	i
open a new session	n
open a recent session	o
open an existing session	o
quit	q
save session	s
snapshot session	s
toggle sel. track MIDI input	i

## Changing What's Visible

fit tracks vertically	f
move selected tracks down	↓
move selected tracks up	↑
scroll down (page)	PgDn
scroll down (step)	↓
scroll up (page)	PageUp
scroll up (step)	↑
toggle editor window mixer	e
toggle last 2 zoom states	z
zoom height to selected region(s)	Control+z
zoom height and time to selected region	z
zoom in	=
zoom out	•

## Window Visibility

toggle locations dialog	l
focus on main clock	÷
maximise editor space	f
rotate editor & mixer window	m
show rhythm ferret window	f
toggle big clock	b
toggle color manager	c
toggle editor window	e
toggle global audio patchbay	p
toggle global midi patchbay	p
toggle key bindings editor	k
toggle preferences dialog	o
toggle preferences dialog	p

## Editing with Edit Point

Most edit functions operate on a single Edit Point (EP). The edit point can be any of: playhead (default), the mouse or an active marker. The choice of edit point (by default) also sets the Zoom Focus.

EP to next region sync	;
EP to prev region sync	'
cycle to next grid snap mode	2
cycle to next zoom focus	1
insert from region list	i
insert time	t
move EP to playhead	Return
next EP w/marker	^
next EP w/o marker	'
trim back	k
trim front	j
trim region end to edit point	}
trim region start to edit point	{
trim region to end of prev region	j
trim region to start of next region	k
use previous grid unit	3
use next grid unit	4
use previous grid unit	3
use next musical grid unit	4

## Aligning with the Edit Point

Align operations move regions so that their start/end/sync point is at the edit point. Relative operations just align the first region and moves other selected regions to maintain relative positioning.

align end(s)	a
align start(s)	
align start(s) relative	
align sync points	a
align sync points (relative)	a
range end to next prev edge	>
range end to next region edge	>
range start to next region edge	<
range start to prev region edge	<

## Edit Point Playback

play edit range	Space
play from EP & return	Space
play selected region(s)	h

## Region Operations

duplicate region (multi)	d
duplicate region (once)	d
export selected region(s)	
increase region gain	^
move to original position	o
mute/unmute	m
normalize	n
nudge backward	-
nudge forward	•
quantize MIDI notes	q
reduce region gain	&
reverse	r
set fade in length	/
set fade out length	\
set region sync point	v
split	s
toggle fade in active	/
toggle fade out active	\
transpose	t

## Generic Editing

copy	c
cut	x
delete	Del
paste	v
redo	r
undo	z

## Selecting

There are a few functions that refer to an Edit Range. The current edit range is defined using combinations of the possible edit points: playhead, active marker, or mouse.

all after playhead	p
all before playhead	p
all enclosed by edit range	u
all present in edit range	u
convert edit range to range	F6
invert selection	i
select all after EP	Shift+e
select all before EP	e
select all in loop range	l
select all in punch range	d
select everything	a
select next track/bus	↓
select previous track/bus	↑

## Defining Loop, Punch Range and Tempo Changes

set loop range from edit range	]
set loop range from region(s)	]
set punch range from edit range	[
set punch range from region(s)	[
set tempo (1 bar) from edit range	0
set tempo (1 bar) from region(s)	9

## Part II - Ardour Configuration

### 4 - Ardour Systems

Using a general purpose computer for the recording and playback of digital audio is not trivial. This chapter covers some of the most common pitfalls encountered on the way to creating a reliable and powerful audio workstation.

#### 4.1 - The Right Computer System for Digital Audio

It is nice to think that one could just go and buy any computer, install a bit of software on it and start using it to record and create music. This idea isn't necessarily wrong, but there are some important details that it misses. Any computer that can be bought today (since somewhere around the end of 2012) is capable of recording and processing a lot of audio data. It will come with a builtin audio interface that can accept inputs from microphones and/or electrical instruments; it will have a disk with a huge amount of space for storing audio files.

However, when recording, editing and mixing music, it is generally desirable to have very little latency between the time a sound is generated and when it can be heard. When the audio signal flows through a computer, that means that the computer has to be able to receive the signal, process it and send it back out again as quickly as possible. And this is where it becomes very important *what* computer system is being used for this task, because it is **absolutely not** the case that *any* computer can do it well.

Routing audio through a computer will always cause some delay, but if it is small, it will generally never be noticed. There are also ways to work in which the delay does not matter at all (for example, not sending the output from the computer to speakers).

The latency that is typically needed for working with digital audio is in the 1–5 ms range. For comparison, if one is sitting 1 m (3 ft) from a set of speakers, the time the sound takes to reach the ears is about 3 ms. Any modern computer can limit the delay to 100 ms; most can keep it under 50 ms. Many will be able to get down to 10 ms without too much effort. Attempting to reduce the latency on a computer that cannot physically do it will cause clicks and glitches in the audio, which is clearly undesirable.

### Hardware-related Considerations

Video interface	Poorly engineered video interfaces (and/or their device drivers) can “steal” computer resources for a long time, preventing the audio interface from keeping up with the flow of data.
Wireless interface	Poorly engineered wireless networking interfaces (and/or their device drivers) can also block the audio interface from keeping up with the flow of data.
USB ports	When using an audio interface connected via USB, and sometimes even if not, the precise configuration of the system’s USB ports can make a big difference. There are many cases where plugging the interface into one port will work, but using different USB port results in much worse performance. This has been seen even on Apple systems.
Internal USB Hubs	Ideally, all USB ports should connect directly to the main bus inside the computer. Some laptops (and possibly some desktop systems) come wired with an internal USB hub between the ports and the system bus, which can then cause problems for various kinds of external USB devices, including some models of audio interfaces. It is very difficult to discover whether this is true or not, without simply trying it out.
CPU speed control	Handling audio with low latency requires that the processor keeps running at its highest speed at all times. Many portable systems try to regulate processor speed in order to save power—for low latency audio, this should be totally disabled, either in the BIOS or at the OS level.
Excessive Interrupt Sharing	If the audio interface is forced by the computer to share an interrupt line (basically a way to tell the CPU that something needs its attention) with too many other (or wrong) devices, this can also prevent the audio interface from keeping up with the flow of data. In laptops it is generally impossible to do anything about this. In many desktop systems, it is possible at the BIOS level to reassign interrupts to work around the problem.
SMIs	SMIs are interrupts sent by the motherboard to tell the computer about the state of various hardware. They cannot safely be disabled, and they can take a relatively long time to process. It is better to have a motherboard which never sends SMIs at all—this is also a requirement for realtime stock trading systems, which have similar issues with latency.
Hyperthreading	This technology is becoming less common as actual multi-core CPUs become the norm, but it still exists and is generally not good for realtime performance. Sometimes this can be disabled in the BIOS, sometimes it cannot. A processor that uses hyperthreading will be less stable in very low latency situations than one without.
Excessive vibration	This doesn’t affect the flow of data to or from the audio interface, but it can cause the flow of data to and from disk storage to become <i>much</i> slower. If a computer going to be used in an environment with loud live sound (specifically, high bass volume), make sure it is placed so that the disk is not subjected to noticeable vibration. The vibrations will physically displace the read-write heads of disk, and the resulting errors will force a retry of the reading from the disk. Retrying over and over massively reduces the rate at which data can be read from the disk. Avoid this.

Richard Ames presents a long (28 minute) [video](#) that is very helpful if you want to understand these issues in more depth. It is a little bit Windows-centric, but the explanations to all types of computers and operating systems.

## 4.2 - The Right Mouse

Ardour is designed to work best with a three button mouse equipped with a scroll wheel. While it can be used with a two button mouse or touchpad, at least two key operations will not be (easily) available:

- time-constrained region copying
- *MIDI bindings* created by “learning” them from incoming MIDI data

It is strongly encouraged to invest in a three-button mouse. A good quality mouse (especially one with a weighted, latchable scroll wheel) will make the use of Ardour vastly more efficient. They are cheap, and time is not.

For information on how to use the mouse in Ardour, see [Using the mouse](#).

## 5 - System Specific Setup

It is unfortunate, but some OSes and/or Desktop Environments will cause problems that are beyond the capability of Ardour to address. The following covers some of the known problems and how to work around them.

### 5.1 - Ubuntu Linux

Ubuntu Linux is the most popular variety of Linux in use on desktop and laptop systems. It has the backing of a for-profit corporation (Canonical Inc.), a defined philosophy and a huge and worldwide user base. As a result, it is a common platform for people who want to use Ardour and other tools for music creation and pro-audio work.

#### High Level Recommendations for Ubuntu Users

Currently, installing pro audio applications on vanilla Ubuntu requires some configuration, in order for the user to gain realtime privilege (read below). Ubuntu Studio, which is an official flavor of Ubuntu, and thus shares the repositories with Ubuntu, has this already configured. Other distributions, such as KXStudio, and Dreamstudio are largely based on Ubuntu, and like Ubuntu Studio, has these settings pre-configured, while also containing customized versions of Ubuntu packages, which often are more up to date.

#### Installing Ardour

There may be unintended differences, and even bugs in Ubuntu native packages, as a result of a different building method. For this reason, Ardour developers highly recommend installing the official ready-to-run version of the program that can be downloaded from [ardour.org](#), as Ubuntu native packages are not supported in the official Ardour forums or other support channels.

Follow these steps to install the latest version of Ardour:

1. Download the latest release from [ardour.org](#).
2. Right-click the downloaded file and choose properties.
3. Click the Permissions tab and check the option “Allow this file to run as a program”.

4. Close the dialog and double-click the file.
5. Follow the prompts.

## Problems with the interaction between PulseAudio and JACK

### Background Info

Like many distributions, Ubuntu has decided to use PulseAudio as the default audio system. PulseAudio is a rich and capable system that provides excellent services for typical users of Linux on the desktop. However, it is not capable of the type of performance that tools like Ardour require and in particular does not offer the possibility of sending audio between applications that can make the Linux audio environment a very interesting one.

This would not be a problem if it were not for the fact that JACK will not run correctly (if at all) if it needs to use the same soundcard/audio interface that PulseAudio is using. And since, PulseAudio on Ubuntu is configured by default to always use the (typically single) audio interface on the computer, this is a bit of a problem.

The developers of JACK and PulseAudio got together in 2009 and agreed upon a mechanism by which PulseAudio and JACK could cooperate in their use of a single soundcard. Whether or not PulseAudio is running by default, when JACK starts up it sends out a request to use the soundcard. If PulseAudio is running, it will give up its use of the soundcard to allow JACK to take over (and can optionally be told to route its own audio through JACK). When JACK finishes, it sends out another message, and PulseAudio can once again use the soundcard directly.

### What is the problem?

The specific issues known at this time for all flavors of Ubuntu 12.04 and 12.10 are:

- a bug in PulseAudio that causes it not to give up the soundcard when JACK asks (LP: #1163638, fixed in Ubuntu 13.04).

### Symptoms

A message like Cannot start JACK in the output from JACK as it starts up (though see the next section for other causes of this). This output may be hidden in the Messages window of QJackCtl (aka JACK Control), so one should check there.

### How to fix

These bugs do not affect releases from 13.04, and earlier releases (12.04 and 12.10) are in the process of being fixed.

## Problems with JACK configuration

### What is the problem?

To function as intended, JACK needs to run with access to two operating system facilities called realtime scheduling and memory locking. This means that the user who starts JACK *must* be allowed access to these facilities. By default, Ubuntu does create a user group that has this permission but—it does not put new users into this group by default. Read more about why [here](#). Consequently, the user will not have permission to run JACK in the way they should.

## Symptoms

A message like Cannot lock down memory in the output from JACK as it starts up. This output may be hidden in the Messages window of QJackCtl (aka JACK Control), so one should check there.

## How to fix

Make sure the file /etc/security/limits.d/audio.conf exists. If it is named /etc/security/limits.d/audio.conf.disabled, rename it to the former. Run the command:

```
sudo usermod -a -G audio YOUR-LGIN-NAME
```

Then log out and log in again. On Ubuntu Studio the user is a member of audio group by default, but not on other official flavors.

## Reporting Issues

Given the difficulties in supporting Ubuntu and the limited time and resources of the Ardour team, the Ubuntu Studio Project has requested that issues and bug reports related to Ubuntu, Ubuntu Studio and other derivatives be directed to them.

## Contact Information for Ubuntu Studio

The Ubuntu Studio Homepage

The Ubuntu Studio Forums.

Information on the Ubuntu Studio Mailing Lists.

Information on the Ubuntu Studio IRC channel. #ubuntustudio on irc.freenode.net

## 5.2 - Microsoft Windows

Microsoft Windows is officially supported since version 5.0 (2016).

## Installing Ardour

1. Download the latest windows build from [the download page](#).
2. Run the installer and follow the prompts.

## How to help

- Hang out in #ardour on irc.freenode.net. One may ask questions there and if possible, answer questions that others have.
- Keep an eye on the [Windows forum](#) and contribute to the discussions there.
- Update this manual via pull requests on [github](#).

## 5.3 - KDE Plasma 5

Under KDE Plasma 5, plugin and various other windows will not stay on top of any main window; therefore a workaround is required.

## Workaround for ancillary windows not staying on top in KDE Plasma 5

In order to force ancillary windows in Ardour to stay on top, the following steps are necessary:

1. Launch the System Settings application.
2. Open Workspace > Window Management.
3. Select Window Rules in the left-hand sidebar. It should default to the Window matching tab.
4. Click on the New... button.
5. On the line that says Window class (application), set the combo box to Substring Match and type ardour in the text entry field.
6. In the list box that is labeled Window types:, click on the option Dialog Window, then press and hold Ctrl while clicking on the second option Utility Window.
7. Select the Arrangement & Access tab.
8. Check the box next to the Keep above option. On the same line, select Force from the combo box, then click on the Yes radio button for that line.
9. Click on the OK button to dismiss the dialog.

At this point the System Settings application can be closed.

## Background Info

According to one of the lead KDE developers, they are not willing to follow the ICCCM standard for utility windows. Apparently they are alone in this understanding, as plugin windows on Ardour under Linux work out of the box on every other WM out there.

Under KDE 4, there was a workaround in Ardour (Preferences > Theme > All floating windows are dialogs) that would “trick” KDE into forcing certain window types to be on top of their parent windows, but this no longer works under KDE Plasma 5.

## 6 - I/O Setup

### 6.1 - Connecting Audio and MIDI Devices

Normally Ardour does not care about how audio and MIDI gets into the computer—it pretty much deals only with its own inputs and outputs; it is up to the user to ensure that all external routing is sound. After all, Ardour has no way to know how signals from the outside world get to it. However, there are some things that Ardour can do to help troubleshoot problems with audio and MIDI connections—at least on the computer side.

For example, a typical setup might include a microphone that feeds a mixer that then feeds the computer. A failure can occur anywhere in that signal chain, including the cables that connect everything together. As far as Ardour is concerned, the most important connection is the one coming from the sound source to the physical audio input of the computer—in this example, the cable connecting between the mixer and the computer.

Common sense and basic troubleshooting skills are needed when problems arise, and in the above example, one would have to go through the entire signal chain to ensure that each component was working as it should.

## Common Problems

Ardour tries to set things up in a sane manner by automatically connecting the hardware inputs of the computer to its master input and the hardware outputs to the master output. If the signal coming into the hardware inputs is active, the meters on Ardour's master channel should move. If they don't, some things to check include:

- Making sure there is actually an input signal
- Making sure the input signal is getting into the computer
- Making sure that Ardour is talking to the correct sound card
- Making sure that the sound card in use by Ardour is working properly

## 6.2 - Using More Than One Audio Device

Ardour will only ever deal with a single audio device. When it is desired to use more than one audio device at the same time, there are two choices:

- Use Ardour to start JACK (which handles all audio I/O), and create a “fake” audio device which represents all the multiple devices to be used. How to do this is platform dependent and described below.
- Use a different tool to start JACK and manage all the devices.

Ardour is fundamentally designed to be a component in a pro-audio/music creation environment. Standard operating practice for such setups involves using only a single digital sample clock (something counting off the time between audio samples). This means that trying to use multiple independent soundcards is problematic, because each soundcard has its own sample clock, running independently from the others. Over time, these different clocks drift out of sync with each other, which causes glitches in the audio. This drift cannot be stopped, although in some cases the effects may be insignificant enough that they might not be noticeable.

Thus, in an ideal world, a single device with a single clock and all the inputs, outputs and other features needed should be used. Of course, there are those who like to point out that this is not an ideal world, and believe that software should make up for this.

## OS X

In CoreAudio, aggregate devices provide a method to use multiple soundcards as a single device. For example, two eight-channel devices can be aggregated so that 16 channels can be recorded in Ardour.

When using a *single* typical 3rd party audio interface (such as those from Apogee, RME, Presonus, and many others), *or* using JackPilot or a similar application to start JACK, there is no need to worry about any of this. An aggregate device only needs to be set up if any of the following conditions are true:

- Two entirely separate devices are used *and* JACK is started using Ardour
- A builtin audio device is used *and* JACK is started using Ardour
- More than two entirely separate devices are used

In the case of a builtin audio device, an aggregate device that combines “Builtin Input” and “Builtin Output” into one device needs to be created.

The precise instructions for creating an aggregate device on OS X have varied from one released to another. Please read <https://support.apple.com/en-us/HT202000>.

## Linux

Please see the instructions at <http://jackaudio.org/faq>.

### 6.3 - Monitor Setup in Ardour

Ardour has three main settings which affect how monitoring is performed. The first is Edit > Preferences > Signal Flow > Record monitoring handled by. There are two or three options here, depending on the capabilities of the hardware.

The other two settings are more complex. One is Tape machine mode, found in the same dialog, and the other is the Session > Properties > Monitoring automatically follows transport state setting.

Monitoring also depends on the state of the track's record-enable button, the session record-enable button, and on whether or not the transport is rolling.

#### Software or Hardware Monitoring Modes

If Ardour is set to external monitoring, Ardour does not do any monitoring.

#### Monitoring in Non-Tape-Machine Mode

When Tape-Machine mode is off, and a track is armed, Ardour *always* monitors the live input, except in one case: if the transport is rolling, the session is not recording, and auto-input is active, the playback from an armed track will be heard.

Unarmed tracks will play back their contents from disc, unless the transport is stopped *and* auto-input is enabled. In this case, the track monitors its live input.

#### Monitoring in Tape-Machine Mode

In Tape-Machine mode, when a track is armed, its behaviour is the same as in non-tape-machine mode.

Unarmed tracks however will always just play back their contents from disk; the live input will never be monitored.

## 7 - Synchronization

### 7.1 - On Clock and Time

Synchronization in multimedia involves two concepts which are often confused: clock (or speed) and time (location in time).

A clock determines the speed at which one or more systems operate. In the audio world this is generally referred to as [Word Clock](#). It does not carry any absolute reference to a point in time: A clock is used to keep a system's sample rate regular and accurate. Word clock is usually at the frequency of the sample rate—at 48 kHz, its period is about 20  $\mu$ s. Word Clock is the most common sample rate based clock but other clocks do exist such as Black and Burst, Tri-Level and DARS. Sample rates can be derived from these clocks as well.

Time or timecode specifies an absolute position on a timeline, such as 01:02:03:04 (expressed as Hours:Mins:Secs:Frames). It is actual *data* and not a clock *signal* per se. The granularity of timecode is Video Frames and is an order of magnitude lower than, say, Word Clock which is counted in samples. A

typical frame rate is 25 fps with a period of 40 ms. In the case of 48 kHz and 25 fps, there are 1920 audio samples per video frame.

The concepts of clock and timecode are reflected in JACK and Ardour:

JACK provides clock synchronization and is not concerned with time code (this is not entirely true, more on jack-transport later). On the software side, jackd provides sample-accurate synchronization between all JACK applications. On the hardware side, JACK uses the clock of the audio-interface. Synchronization of multiple interfaces requires hardware support to sync the clocks. If two interfaces run at different clocks the only way to align the signals is via re-sampling (SRC—Sample Rate Conversion), which is expensive in terms of CPU usage and may decrease fidelity if done incorrectly.

Timecode is used to align systems already synchronized by a clock to a common point in time, this is application specific and various standards and methods exist to do this.

To make things confusing, there are possibilities to synchronize clocks using timecode. e.g. using mechanism called jam-sync and a phase-locked loop.

An interesting point to note is that LTC (Linear Time Code) is a Manchester encoded, frequency modulated signal that carries both clock and time. It is possible to extract absolute position data and speed from it.

## 7.2 - Latency and Latency-Compensation

Latency is a system's reaction time to a given stimulus. There are many factors that contribute to the total latency of a system. In order to achieve exact time synchronization all sources of latency need to be taken into account and compensated for.

### Sources of Latency

#### Sound propagation through the air

Since sound is a mechanical perturbation in a fluid, it travels at comparatively slow speed of about 340 m/s. As a consequence, an acoustic guitar or piano has a latency of about 1–2 ms, due to the propagation time of the sound between the instrument and the player's ear.

#### Digital-to-Analog and Analog-to-Digital conversion

Electric signals travel quite fast (on the order of the speed of light), so their propagation time is negligible in this context. But the conversions between the analog and digital domain take a comparatively long time to perform, so their contribution to the total latency may be considerable on otherwise very low-latency systems. Conversion delay is usually below 1 ms.

#### Digital Signal Processing

Digital processors tend to process audio in chunks, and the size of that chunk depends on the needs of the algorithm and performance/cost considerations. This is usually the main cause of latency when using a computer and the one that can be predicted and optimized.

#### Computer I/O Architecture

A computer is a general purpose processor, not a digital audio processor. This means the audio data has to jump a lot of fences in its path from the outside to the CPU and back, contending in the process with some other parts of the system vying for the same resources (CPU time, bus bandwidth, etc.)

## The Latency chain

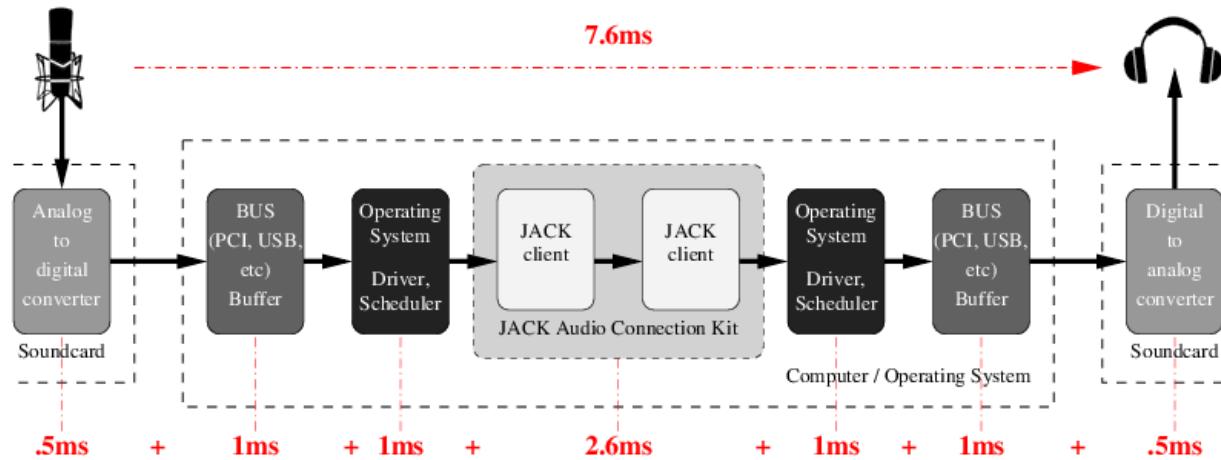


Fig. 1: Latency chain

The numbers are an example for a typical PC. With professional gear and an optimized system the total round-trip latency is usually lower. The important point is that latency is always additive and a sum of many independent factors.

Processing latency is usually divided into capture latency (the time it takes for the digitized audio to be available for digital processing, usually one audio period), and playback latency (the time it takes for In practice, the combination of both matters. It is called round-trip latency: the time necessary for a certain audio event to be captured, processed and played back.

It is important to note that processing latency in a jackd is a matter of choice. It can be lowered within the limits imposed by the hardware (audio device, CPU and bus speed) and audio driver. Lower latencies increase the load on the system because it needs to process the audio in smaller chunks which arrive much more frequently. The lower the latency, the more likely the system will fail to meet its processing deadline and the dreaded xrun (short for buffer over- or under-run) will make its appearance more often, leaving its merry trail of clicks, pops and crackles.

The digital I/O latency is usually negligible for integrated or PCI audio devices, but for USB or FireWire interfaces the bus clocking and buffering can add some milliseconds.

## Low Latency use cases

Low latency is **not** always a feature one wants to have. It comes with a couple of drawbacks: the most prominent is increased power consumption because the CPU needs to process many small chunks of audio data, it is constantly active and can not enter power-saving mode (think fan noise). Since each application that is part of the signal chain must run in every audio cycle, low-latency systems will undergo context switches between applications more often, which incur a significant overhead. This results in a much higher system load and an increased chance of xruns.

For a few applications, low latency is critical:

## Playing virtual instruments

A large delay between the pressing of the keys and the sound the instrument produces will throw off the timing of most instrumentalists (save church organists, whom we believe to be awesome latency-compensation

organic systems.)

### Software audio monitoring

If a singer is hearing her own voice through two different paths, her head bones and headphones, even small latencies can be very disturbing and manifest as a tinny, irritating sound.

### Live effects

Low latency is important when using the computer as an effect rack for inline effects such as compression or EQ. For reverbs, slightly higher latency might be tolerable, if the direct sound is not routed through the computer.

### Live mixing

Some sound engineers use a computer for mixing live performances. Basically that is a combination of the above: monitoring on stage, effects processing and EQ.

In many other cases, such as playback, recording, overdubbing, mixing, mastering, etc. latency is not important, since it can easily be compensated for.

To explain that statement: During mixing or mastering, one doesn't care if it takes 10ms or 100ms between the instant the play button is pressed and the sound coming from the speaker. The same is true when recording with a count in.

### Latency compensation

During tracking it is important that the sound that is currently being played back is internally aligned with the sound that is being recorded.

This is where latency compensation comes into play. There are two ways to compensate for latency in a DAW, read-ahead and write-behind. The DAW starts playing a bit early (relative to the playhead), so that when the sound arrives at the speakers a short time later, it is exactly aligned with the material that is being recorded. Since we know that playback has latency, the incoming audio can be delayed by the same amount to line things up again.

The second approach is prone to various implementation issues regarding timecode and transport synchronization. Ardour uses read-ahead to compensate for latency. The time displayed in the Ardour clock corresponds to the audio signal that is hearded on the speakers (and is not where Ardour reads files from disk).

As a side note, this is also one of the reasons why many projects start at timecode 01:00:00:00. When compensating for output latency the DAW will need to read data from before the start of the session, so that the audio arrives in time at the output when the timecode hits 01:00:00:00. Ardour does handle the case of 00:00:00:00 properly but not all systems/software/hardware that you may inter-operate with may behave the same.

### Latency Compensation And Clock Sync

To achieve sample accurate timecode synchronization, the latency introduced by the audio setup needs to be known and compensated for.

In order to compensate for latency, JACK or JACK applications need to know exactly how long a certain signal needs to be read-ahead or delayed:

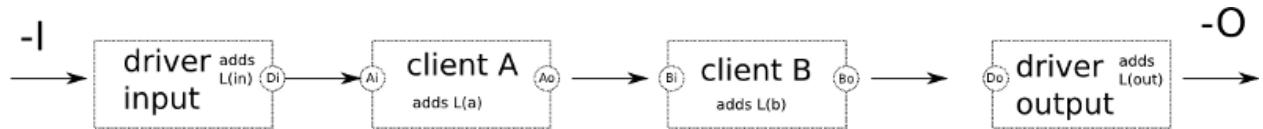


Fig. 2: Jack Latency Compensation

In the figure above, clients A and B need to be able to answer the following two questions:

- How long has it been since the data read from port Ai or Bi arrived at the edge of the JACK graph (capture)?
- How long will it be until the data written to port Ao or Bo arrives at the edge of the JACK graph (playback)?

JACK features an API that allows applications to determine the answers to above questions. However JACK can not know about the additional latency that is introduced by the computer architecture, operating system and soundcard. These values can be specified by the JACK command line parameters -I and -O and vary from system to system but are constant on each. On a general purpose computer system the only way to accurately learn about the total (additional) latency is to measure it.

### Calibrating JACK Latency

Linux DSP guru Fons Adriaensen wrote a tool called `jack_delay` to accurately measure the round-trip latency of a closed loop audio chain, with sub-sample accuracy. JACK itself includes a variant of this tool called `jack_idelay`.

`Jack_idelay` allows to measure the total latency of the system, subtracts the known latency of JACK itself and suggests values for `jackd`'s audio-backend parameters.

`jack_[io]delay` works by emitting some rather annoying tones, capturing them again after a round trip through the whole chain, and measuring the difference in phase so it can estimate with great accuracy the time taken.

The loop can be closed in a number of ways:

- Putting a speaker close to a microphone. This is rarely done, as air propagation latency is well known so there is no need to measure it.
- Connecting the output of the audio interface to its input using a patch cable. This can be an analog or a digital loop, depending on the nature of the input/output used. A digital loop will not factor in the AD/DA converter latency.

Once the loop has been closed, one must:

1. Launch `jackd` with the configuration to test.
2. Launch `jack_delay` on the command line.
3. Make the appropriate connections between the jack ports so the loop is closed.
4. Adjust the playback and capture levels in the mixer.

### 7.3 - Timecode Generators and Slaves

Ardour supports three common timecode formats: LTC, MTC, and MIDI Clock, as well as JACK-transport, a JACK-specific timecode implementation.

Ardour can generate timecode and thus act as timecode master, providing timecode information to other applications. Ardour can also be slaved to some external source in which case the playhead follows the incoming timecode.

Combining the timecode slave and generator modes, Ardour can also translate timecode. e.g create LTC timecode from incoming MTC.

### Ardour Timecode Configuration

Each Ardour session has a specific timecode frames-per-second setting which is configured in session > properties > timecode. The selected timecode affects the timecode ruler in the main window as well as the clock itself.

Note that some timecode formats do not support all of Ardour's available fps settings. MTC is limited to 24, 25, 29.97 and 30 fps.

The video pull-up modes change the effective samplerate of Ardour to allow for changing a film soundtrack from one frame rate to another. The concept is beyond the scope of this manual, but Wikipedia's entry on [Telecine](#) may be a good start.

### Ardour Timecode Generator Configuration

This is pretty straightforward: simply turn it on. The MTC and MIDI-Clock generator do not have any options. The LTC generator has a configurable output level. JACK-transport cannot be *generated*. Jack itself is always synced to its own cycle and cannot do varispeed—it will always be synced to a hardware clock or another JACK master.

The relevant settings for timecode generator can be found in Edit > Preferences > MIDI Preferences (for MTC, MC) and Edit > Preferences > Transport Preferences (for LTC).

The timecode is sent to jack-ports `ardour:MTC out`, `ardour:MIDI clock out` and `ardour:LTC-out`. Multiple generators can be active simultaneously.

Note that, as of Jan 2014, only the LTC generator supports latency compensation. This is due to the fact the Ardour MIDI ports are not yet latency compensated.

In Session > Properties, it is possible to define an offset between Ardour's internal time and the timecode sent. Currently only the LTC generator honors this offset.

Both LTC and MTC are limited to 30 fps. Using frame rates larger than that will disable the generator. In both cases also only 24, 25, 29.97df (drop-frame) and 30 fps are well defined by specifications (such as SMPTE-12M, EU and the MIDI standard).

### MTC Generator

The MTC generator has no options. Ardour sends full MTC frames whenever the transport is relocated or changes state (start/stop). MTC quarter frames are sent when the transport is rolling and the transport speed is within 93% and 107%.

### LTC Generator

The level of the LTC generator output signal can be configured in the Preferences > Transport dialog. By default it is set to -18 dBFS, which corresponds to 0dBu in an EBU calibrated system.

The LTC generator has an additional option to keep sending timecode even when the transport is stopped. This mode is intended to drive analog tape machines which unspool the tape if no LTC timecode is received.

LTC is send regardless of Ardour's transport speed. It is accurately generated even for very slow speeds (<5%) and only limited by the soundcard's sampling-rate and filter (see Gibbs phenomenon) for high speeds.

### **Ardour Slave Configuration**

The timecode source can be switched with the button just right of Ardour's main clock. By default it is set to Internal in which case Ardour will ignore any external timecode. The button allows to toggle between Internal and the configured timecode source which is chosen in Edit > Preferences > Transport.

When Ardour is chasing (synchronizing to) an external timecode source, the following cases need to be distinguished:

1. the timecode source shares the clock
2. the timecode source is independent (no wordclock sync)

and

1. the timecode source uses the same FPS setting as Ardour
2. the timecode source runs at different frames-per-second

In both cases the first option is preferred: clock sync + same FPS setting.

### **Frames-per-second**

If the frames-per-second do not match, Ardour can either re-calculate and map the frames, or the configured FPS (Session > Properties) can be changed automatically while the slave is active. The behavior is configured with the checkbox Edit > Preferences > Transport > Match session video frame rate to external timecode.

When enabled, the session video frame rate will be changed to match that of the selected external timecode source. When disabled, the session video frame rate will not be changed to match that of the selected external timecode source. Instead the frame rate indication in the main clock will flash red, and Ardour will convert between the external timecode standard and the session standard.

29.97 drop-frame timecode is another corner case. While the SMPTE 12M-1999 specifies 29.97df as 30000/1001 frames per second, not all hardware devices follow that standard. The checkbox Lock to 29.9700 fps instead of 30000/1001 allows to use a compatibility mode for those devices.

When enabled, the external timecode source is assumed to use 29.970000 fps instead of 30000/1001. SMPTE 12M-1999 specifies 29.97df as 30000/1001. The spec further mentions that drop-frame timecode has an accumulated error of -86 ms over a 24-hour period. Drop-frame timecode would compensate exactly for a NTSC color frame rate of  $30 * 0.9990$  (ie 29.970000). That is *not* the actual rate. However, some vendors use that rate—despite it being against the specs—because the variant of using exactly 29.97 fps yields zero timecode drift.

### **Clock Sync Lock**

As described in the [On Clock and Time](#) chapter, timecode and clock are independent. If the external timecode source is not in sample-sync with the audio hardware (and JACK), Ardour needs to run at varispeed to adjust for the discrepancy.

The checkbox External timecode is sync locked allows to select the behavior according to the setup. When enabled, it indicates that the selected external timecode source shares sync (Black & Burst, Wordclock, etc) with the audio interface.

In other words: if enabled, Ardour will only perform initial synchronization and keep playing at speed 1.0 instead of vari-speed adjusting to compensate for drift.

Note that vari-speed is unavailable when recording in Ardour, and all tracking happens at speed 1.0. So in order to record in sync with external timecode it must be sample-locked or it will drift over time.

## MIDI Clock

MIDI Clock is not a timecode format but tempo-based time. The absolute reference point is expressed as beats-per-minute and Bar, Beat and Tick. There is no concept of sample-locking for MIDI clock signals. Ardour will vari-speed if necessary to chase the incoming signal.

Note that the MIDI Clock source must be connected to the `ardour:MIDI clock in` port.

## LTC—Linear Timecode

The LTC slave decodes an incoming LTC signal on a JACK audio port. It will auto-detect the frame rate and start locking to the signal once two consecutive LTC frames have been received.

The incoming timecode signal needs to arrive at the `ardour:LTC-in` port. Port-connections are restored for each session and the preference dialog offers an option to select it for all sessions.

Ardour's transport is aligned to LTC-frame start/end positions according to the SMPTE 12M-1999 specification, which means that the first bit of an LTC-Frame is aligned to different Lines of a Video-Frame, depending on the TV standard used. Only for Film (24fps) does the LTC-Frame directly match the video Frame boundaries.

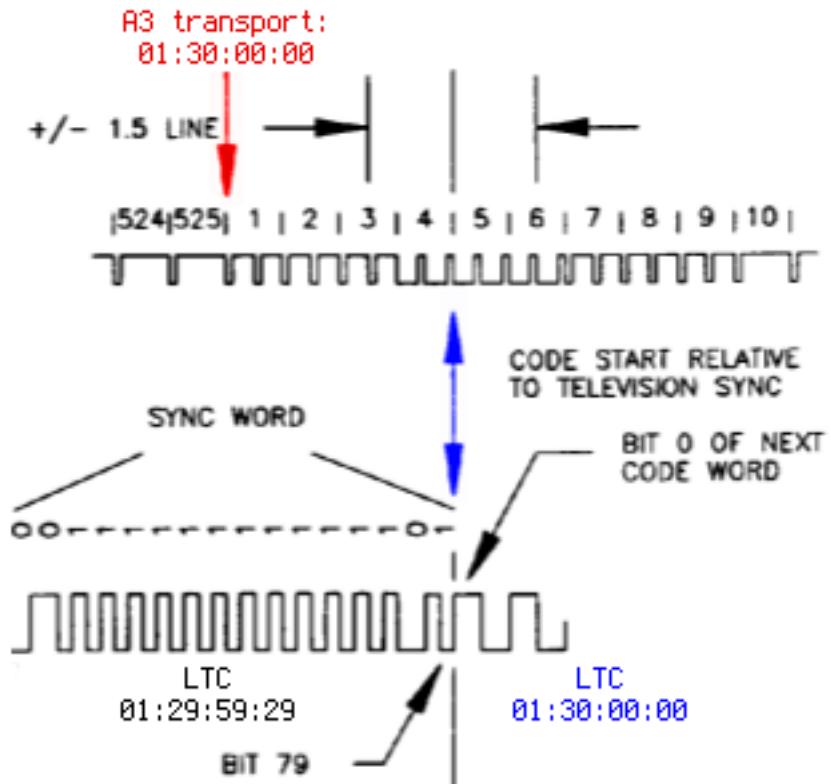


Fig. 3: LTC frame alignment for the 525/60 TV standard

Ardour supports vari-speed and backwards playback but will only follow speed changes if the sync locked option is disabled.

While Ardour is chasing LTC, the main transport clock will display the received Timecode as well as the delta between the incoming signal and Ardour's transport position.

A global offset between incoming timecode and Ardour's transport can be configured in Session > Properties.

The user-bits in the received LTC frame are ignored.

### **MTC—MIDI Timecode**

Ardour's MTC slave parses full timecode messages as well as MTC quarter-frame messages arriving on the `ardour:MTC in` port. The transport will only start rolling once a complete sequence of 8 quarter frames has been received.

Ardour supports vari-speed and backwards playback but will only follow MTC speed changes if the sync locked option is disabled.

When Ardour is chasing MTC, the main transport clock will display the received Timecode as well as the delta between the incoming signal and Ardour's transport position.

### **JACK Transport**

When slaved to jack, Ardour's transport will be identical to JACK-transport. As opposed to other slaves, Ardour can be used to control the JACK transport states (stopped/rolling). No port connections need to be made for jack-transport to work.

JACK-transport does not support vari-speed, nor offsets. Ardour does not chase the timecode but is always in perfect sample-sync with it.

JACK-transport also includes temp-based-time information in Bar:Beats:Ticks and beats-per-minute. However, only one JACK application can provide this information at a given time. The checkbox Session > Properties > JACK Time Master configures Ardour to act as translator from timecode to BBT information.

## **7.4 - Overview of all Timecode related settings**

Timecode settings are accessed from the menu in three places:

- Session > Properties > Timecode
- Edit > Preferences > Transport
- Edit > Preferences > MIDI

## Timecode Settings

Timecode frames-per-second	Configure timecode frames-per-second (23.976, 24, 24.975, 25, 29.97, 29.97 drop, 30, 30 drop, 59.94, 60). Note that all fractional framerates are actually $\text{fps} * (1000.0 / 1001.0)$ .
Pull up/down	Video pull-up modes change the effective samplerate of Ardour to allow for changing a film soundtrack from one frame rate to another. See <a href="#">Telecine</a>
Slave Timecode offset	The specified offset is added to the received timecode (MTC or LTC).
Timecode Generator offset	Specify an offset which is added to the generated timecode (so far only LTC).
JACK Time Master	Provide Bar Beat Tick and other information to JACK.

These settings are session specific.

## Transport Preferences

External timecode source	Select timecode source: JACK, LTC, MTC, MIDI Clock
Match session video frame rate to external timecode	This option controls the value of the video frame rate <i>while chasing</i> an external timecode source. When enabled, the session video frame rate will be changed to match that of the selected external timecode source. When disabled, the session video frame rate will not be changed to match that of the selected external timecode source. Instead the frame rate indication in the main clock will flash red and Ardour will convert between the external timecode standard and the session standard.
External timecode is sync locked	Indicates that the selected external timecode source shares sync (Black & Burst, Wordclock, etc) with the audio interface.
Lock to 29.9700 fps instead of 30000/1001	The external timecode source is assumed to use 29.97 fps instead of 30000/1001. SMPTE 12M-1999 specifies 29.97df as 30000/1001. The spec further mentions that drop-frame timecode has an accumulated error of -86ms over a 24-hour period. Drop-frame timecode would compensate exactly for a NTSC color frame rate of $30 * 0.9990$ (ie 29.970000). That is not the actual rate. However, some vendors use that rate—despite it being against the specs—because the variant of using exactly 29.97 fps has zero timecode drift.
LTC incoming port	Offers a session agnostic way to retain the LTC port connection.
Enable LTC generator	Does just what it says.
Send LTC while stopped	Enable to continue to send LTC information even when the transport (playhead) is not moving. This mode is intended to drive analog tape machines which unspool the tape if no LTC timecode is received.
LTC generator level	Specify the Peak Volume of the generated LTC signal in dbFS. A good value is 0 dBu (which is -18 dbFS in an EBU calibrated system).

These settings are common to all sessions.

## MIDI Preferences

Send MIDI Timecode	Enable MTC generator
Send MIDI Clock	Enable MIDI Clock generator

These settings are also common to all sessions.

## 8 - Preferences

Global preferences control general workflow and system configuration, and should apply to all sessions. They are located in Edit > Preferences and stored in Ardour's user configuration file in the user's home directory.

## General

- DSP CPU Utilization
  - Signal processing uses: sets how many cpu processors can be used to do signal processing. It can be set to use one up to all processors.
- Memory Usage
  - Waveform image cache (megabytes): sets the maximum amount of ram that can be used to store the images representing the waveforms in the editor. Past this amount, the images will be regenerated when needed, which can significantly decrease the system's performance.
- Engine
  - Try to auto-launch audio/midi engine allows Ardour to try to automatically launch the audio and MIDI system, driver and device, thus not showing the *Audio/MIDI Setup* dialog. This can save a little time if the system is always used the same way.
- Automation
  - **Thinning factor** ranges from 0 to 1000 with larger values sending fewer automation changes. Thinning is like lossy audio compression, removing data that is less likely to be noticed, although the more is removed, the more likely the loss will be noticed. The advantage to thinning is reduced CPU usage.
  - **Automation sampling interval** ranges from 1 to 1000 ms. Determines how frequently the automation input is sampled. The shorter the interval the higher the accuracy but also the higher the CPU requirements.
- Tempo
  - Allow non quarter-note pulse: by default, Ardour understands the *tempo* as the number of quarter notes in a minute. Checking this option allows to set the tempo based on any division of the note, from whole to 1/128th. This is reflected in the Edit Tempo window (accessed by double clicking a tempo marker) that shows a “Pulse” menu when this option is checked.
- GUI Lock
  - Lock timeout (seconds): locks the GUI after this many idle seconds (zero being ‘never lock’). The GUI can also be locked with Session > Lock. When locked, a dialog will display a “Click to unlock” button.

## Session

- Undo defines the behaviour of the Undo operations:
  - **Limit undo history** sets how many commands can be undone using Z or Edit > Undo. Unchecking will keep an endless memory of operations to undo, at the expense of memory.
  - **Save undo history** sets how many commands are saved so they are available to be undone after reopening the session. This can also be unchecked to keep all actions undoable, at the cost of bigger session files.
  - **Verify removal of last capture** when enabled prompts to verify removal the last recording capture when Edit > Remove Last Capture is executed.
- Session Management:
  - **Make periodic backups of the session file** will create a backup session file after changes to the timeline. The backup file is the session name followed by *.ardour.bak*. The backup can be used to recover from crashes when the session had not been explicitly saved.

- **Always copy imported files** selects, and then disables changes to, the *Copy files to session* option in the *Add Existing Media* dialog.
- **Default folder for new sessions:** defaults the folder where Ardour will create new session folders. This is used in the *Session Setup* dialog displayed by Session > New.
- **Maximum number of recent sessions:** determines how many of the last opened sessions shows in the *Recent Sessions* dialog displayed by Session > Recent.

### Translation

- Internationalization
  - Use translations sets if Ardour should use a translated version of all the messages. The default (unchecked) is English (US). When checked, and if a language file exists for the system language, this file will be used to translate Ardour.

### Editor

- General
  - Snap rubberband to grid when enabled uses the grid when *selecting regions* with a rubberband rectangle.
  - Prompt for new marker names when enabled, popup a dialog when a new *marker* is created. This allows markers to be named as they are created.
  - Allow dragging of the playhead, when enabled, allows dragging the playhead with the mouse in the **Editor** window.
  - Playhead dragging speed (%): chooses how fast the canvas scrolls when dragging the playhead outside of the visible canvas.
  - Limit zooming & summary view to X minutes beyond session extents prevents the zoom out both in the editor and the summary, to show anything past the chosen time after the end marker, restraining the vision to only useful content.
  - Zoom to mouse position when zooming with scroll wheel: by default, Ardour zooms to the *edit point*. When this option is checked, and the zoom is done with + mousewheel, the zoom will happen at the mouse cursor position regardless of the edit point chosen.
  - Zoom with vertical drag in rulers allows, when checked, to click anywhere in an empty zone of the *ruler* zone and drag up to zoom in or down to zoom out.
  - Double click zooms to selection allows by double clicking, to zoom on the selection, both on the time and tracks axes. If the selection has been done with or , then this key should still be pressed when double clicking for this to work, otherwise the first click breaks the group selection.
  - Update editor window during drags of the summary: when enabled the contents of the editor window will redraw the tracks area as the selection rectangle in the *summary* area is moved or resized.
  - Auto-scroll editor window when dragging near its edges when enabled will scroll the editor window automatically when dragging a region. This can make it easier to see where to position the region.
  - Show gain envelopes in audio regions: sets in which *modes* the gain envelope is displayed in audio regions. The gain envelope is superimposed over the region in the selected modes, and hidden otherwise for a better legibility.
- Editor Behaviour

- Move relevant automation when audio regions are moved, when enabled, causes automation data to stay with a region when the region is moved inside the playlist. When disabled, the automation is not affected by movement of regions.
- Ignore Y-axis click position when adding new automation-points allows to create new automation points at the x-position of the mouse, without using the Y-position as the value, hence creating a new automation point at its present value.
- Default fade shape: sets which *fade shape* is the default.
- Regions in edit groups are edited together: sets the criteria to see if editing actions apply to tracks grouped together in a group.
- Layering model: Ardour allows *layering* multiple regions in the same track. This selector defines how these layers are managed, either manually or by placing the latest on top.
- After splitting selected regions, select: determines which, if any, regions are selected after a split operation. The options are no regions, the regions created by the split, and if more than one region was selected to start with, the existing selection and the new regions.

## Modifiers

This page allows to choose how things are done in the editor. This is a very flexible way for Ardour to match an existing workflow, or speed up the editing process based on the user's most used actions.

The Reset to recommended defaults button at the bottom provides a way to revert any user made change to its default value.

## Mixer

- Solo contains settings that affect the use of *solo*, *muting*, and *panning*.
  - Solo controls are Listen controls: when enabled, the soloed track is soloed only on the monitor bus, the master fader mix is not affected by the solo. This option can also be set by enabling pre-fader listen or after-fader listen in the **Mixer** monitor section.
  - Exclusive solo when enabled will only solo the last track selected for solo. Previously soloed tracks will be un-soloed. This setting is also available from the **Mixer** monitor section.
  - Show solo muting when enabled outlines the mute button on tracks and busses when another track is soloed.
  - Soloing overrides muting when enabled allows a track to be heard when it is soloed while muted. This setting is also available from the **Mixer** monitor section.
  - Solo-in-place mute cut (dB): sets the attenuation of the other tracks when another track is soloed in place. This setting is also available from the **Mixer** monitor section. The default is “-inf” for  $-\infty$ , meaning the other tracks are totally muted.
  - Listen Position: determines what is listened to when the solo controls are used as listen controls. The options are after-fader or pre-fader.
  - PFL signals come from: determines whether the pre-fader listen position is before or after the pre-fader processors.
  - AFL signals come from: determines whether the after-fader listen position is before or after the after-fader processors.
- Default Track / Bus Muting Options sets the muting options for a newly created tracks or bus. The mute options for an existing track or bus are changed by the right-click context menu on a mute button.

- Mute affects pre-fader sends when enabled pre-fader sends will be muted by default.
- Mute affects post-fader sends when enabled post-fader sends will be muted by default.
- Mute affects control outputs when enabled control outputs are muted by default.
- Mute affects main outputs when enabled main outputs are muted by default.
- Send Routing affects *aux and external sends*.
  - Link panners of Aux and External Sends with main panner by default when enabled, sends follow the channel panner. When disabled, sends can be panned independently of the channel panner and fader. Double clicking the send in the processor box toggles the main panner and fader between the aux send and the channel.

## Signal Flow

- Monitoring
  - Record monitoring handled by: determines whether Ardour provides monitoring of incoming audio or whether monitoring is provided by hardware. See [Monitoring](#) for more information.
  - Tape machine mode when enabled defaults new audio tracks to tape machine mode. See [Track Types](#) for more information.
- Track and Bus Connections
  - Auto-connect main output (master or monitor) bus to physical ports auto-connects the outputs to the first N physical ports. In a session without a *monitor section*, the master-bus is connected to the system's playback ports, and if a monitor section exists, the monitor-bus' output are connected.
  - Connect track inputs: allows to choose when a new track is created whether its inputs will be automatically connected to the physical inputs of the system or not (hence the user has to manually connect it).
  - Connect track and bus outputs: allows to choose, for any new track or bus created, whether its output will automatically be connected to the master bus, directly to the physical outputs or to nothing (the user has to manually connect it).
  - Use 'Strict-I/O' for new tracks or busses determines the default choice for the *signal flow* of a newly created track or bus. This can be overridden in the *Add Track/Bus/VCA* dialog

## Audio

- Buffering settings determine how many seconds of audio off of disk will be buffered in memory. Longer settings reduce the risk of buffer under-runs but consume more memory.
  - Preset: will automatically choose the values for the playback and recording buffer based on the chosen size of the session. The **Custom** option allows to manually select the buffers with the two sliders below.
  - Playback (seconds of buffering): sets how many seconds of audio Ardour will buffer during playback.
  - Recording (seconds of buffering): sets how many seconds of audio Ardour will buffer during recording.
- Denormals are a specific type of very small numbers that can cause issues with CPU consumption when using some plugins in some circumstances. Ardour provides two methods of handling the issue. Trying different combinations of these settings may minimize CPU consumption.

- Use DC bias to protect against denormals adds a small constant value to numbers to move the numbers away from zero.
- Processor handling; if the computer’s hardware supports it, offers two methods that can be used individually or combined. Flush to zero and denormals are zero.
- Regions
  - Enable automatic analysis of audio generates the transient values (used in e.g. the *Rhythm Ferret*) automatically. When unchecked, the transient values will be generated on demand.
  - Replicate missing region channels: if a track is N-channel, and the region has fewer channels, this option copies the existing channel’s data for this non-existent one. If left unchecked, the missing channels will stay silent.

## MIDI

- Buffering
  - MIDI read-ahead time (seconds): defines how much time of MIDI data must be read in advance by Ardour and put in the buffer. More time means more stability while playing back, at the expense of more time to buffer the data. This should be set to a low value for a reasonably capable machine.
- Session
  - Initial program change: Ardour will send a MIDI program change message on the `ardour:MMC` out JACK port when a session is loaded and whenever this field is changed. A value of -1 means don’t send any program change message.
- Audition
  - Sound MIDI notes as they are selected in the editor will play any selected or added MIDI note when in Draw or Internal Edit modes. The note is sent as MIDI as if Ardour was playing it with the session, so the processors and signal routing will be applied.
  - Midi Audition Synth (LV2): allows to select in the list of LV2 instruments, which one will be used to audition MIDI when e.g. in the *import dialog*.

## Metronome

- Metronome handles the way Ardour’s metronome is played when enabled in the *Transport Bar*.
  - Emphasis on first beat plays a different sound when the first beat is played (e.g. 1/4 in 4/4, 1/3 in 3/4,...). When unchecked, all the beats are indistinguishable.
  - Use built-in default sounds when checked, uses Ardour’s own sounds for the metronome click. Unchecking this allows to set some custom sounds below.
  - Audio file: selects an audio file for the beats, in any *format* Ardour supports.
  - Emphasis audio file: in conjunction with Emphasis on first beat, selects an audio file for the first beats of each bar.
  - Gain level: allows the metronome’s click sounds to be boosted or attenuated.
- Options
  - Enable Metronome only while recording: when enabled, the metronome will remain silent unless Ardour is recording.

## Metering

- Metering
  - Peak hold time: allows the meter to keep displaying the highest signal level for a period of time before reverting to showing the actual instantaneous value (unless an even higher peak occurs). The longer this time is, the easier it is to spot peaks, at the expense of instantaneous accuracy.
  - DPM fall-off: describes how fast the Digital Peak Meters can go from a high value to a lower one. Faster values are more accurate but less readable.
  - Meter line-up level; 0 dBu: chooses a standard for the conversion between dBFS (Full Scale) which represent the numeric signal level, and dBu which represents the analog signal level. This value is used to configure meter-marks and color knee-points, or set the reference levels for various meter-types.
  - IEC1/DIN Meter line-up level; 0 dBu: sets the reference level for the IEC1/DIN Meter
  - VU Meter standard: selects which standard to use for the zero value of the vu-meters, i.e. the analog dBu value that will show as 0 on the VU-meter.
  - Peak indicator threshold [dBFS]: at that value and over, the signal will make the peak meter to turn red, indicating a level too high.
- Default Meter Types sets the default meters when creating a session or track. These meters can be changed afterwards by right-clicking a meter.
  - Default Meter Type for Master Bus: defines which kind of *meter* will be used when creating a new session (does not apply to the current session).
  - Default Meter Type for busses: defines which kind of meter will be used when creating a new bus (applies to the bus created after changing the value).
  - Default Meter Type for tracks: same as above, for tracks.
- Post Export Analysis
  - Save loudness analysis as image file allows, when the Analyze Exported Audio is checked in the *Export dialog*, to save the analysis graph as a file named `session.png` alongside the exported audio file(s) (in the same folder).

## Transport

- General
  - Stop at the end of the session causes the transport to stop during playback when it reaches the end marker. Behavior during recording is not changed.
  - Keep record-enable engaged on stop leaves the global record-enable engaged after transport is stopped. Does not affect track level record-enable which is never changed on stop.
  - Disable per-track record disarm while rolling, when enabled, will not allow the any track's record-enable to be disarmed during record, preventing accidentally stopping the recording of a take.
  - 12dB gain reduction during fast-forward and fast-rewind when enabled will reduce the unpleasant increase in perceived volume that occurs when fast-forwarding or rewinding through some kinds of audio.
  - Preroll: sets the duration of the preroll for playing and recording when using a preroll. Can be a musical duration (in bars) or a duration in seconds.
- Looping

- Play loop is a transport mode changes the behavior of the loop button, turning it into a toggle. When enabled, the loop button does not start playback but forces playback to always play the loop. Looping stays engaged when the transport is stopped. Playback continues where the transport stopped and continues to loop. When disabled, the loop button starts playing the loop but stop then cancels loop playback.
- Do seamless looping removes any clicks that might otherwise be audible when the transport moves from the end of the loop range back to the beginning.
- Dropout (xrun) Handling
  - Stop recording when an xrun occurs will stop the transport when an xrun occurs during recording, ensuring no audible glitches are recorded.
  - Create markers where xruns occur will create a new *marker* when an xrun occurs during recording at the location of the xrun. This marks where possible xruns might produce audible glitches.

## Sync

- External Synchronization
  - External timecode source determines which external source to use when Ardour is using an external *synchronization* source. Depending on the timecode source chosen, the additional preference options below are available.
  - Match session video frame rate to external timecode controls the value of the video frame rate *while chasing* an external timecode source. When enabled, the session video frame rate will be changed to match that of the selected external timecode source. When disabled, the session video frame rate will not be changed to match that of the selected external timecode source. Instead, the frame rate indication in the main clock will flash red and Ardour will convert between the external timecode standard and the session standard.
  - Sync-lock timecode to clock (disable drift compensation) When enabled, Ardour will never varispeed when slaved to external timecode. Sync Lock indicates that the selected external timecode source shares clock-sync (Black & Burst, Wordclock, etc) with the audio interface. This option disables drift compensation. The transport speed is fixed at 1.0. Vari-speed LTC will be ignored and cause drift. When disabled, Ardour will compensate for potential drift regardless if the timecode sources shares clock sync.
  - Lock to 29.9700 fps instead of 30000/1001, when enabled, will force Ardour to assume the external timecode source uses 29.97 fps instead of 30000/1001. SMPTE 12M-1999 specifies 29.97 df as 30000/1001. The spec further mentions that drop-frame timecode has an accumulated error of -86 ms over a 24 hour period. Drop-frame timecode would compensate exactly for an NTSC color frame rate of  $30 \times 0.9990$  (i.e. 29.970000). That is not the actual rate. However, some vendors use that rate—despite it being against the specs—because the variant of using exactly 29.97 fps has zero timecode drift.

## LTC

- Linear Timecode (LTC) Reader
  - LTC incoming port: specifies which physical incoming port of the system will provide the LTC signal.
- Linear Timecode (LTC) Generator
  - Enable LTC generator when enabled Ardour will output an LTC timecode signal on its *LTC-out* port. If this option is checked, the two options below are active:

- Send LTC while stopped, when enabled Ardour will continue to send LTC information even while the transport (playhead) is not moving.
- LTC generator level [dBFS]: specifies the peak volume of the generated LTC signal in dBFS. A good value is 0dBu=−18dBFS in an EBU calibrated system.

## MIDI

- MIDI Beat Clock (Mclk) Generator
  - Enable Mclk generator when enabled Ardour will generate a (tempo dependant) beat clock at a rate of 24 pulses per quarter note on the `ardour:MIDI clock` out JACK port.
- MIDI Time Code (MTC) Generator
  - Enable MTC Generator when enabled Ardour will generate MIDI time code on the `ardour:MTC out` JACK port.
  - Percentage either side of normal transport speed to transmit MTC: MIDI time code generation will be disabled when the transport speed is greater than normal speed plus this percentage or less than normal minus this percentage.
- MIDI Machine Control (MMC)
  - Respond to MMC commands when enabled Ardour will respond to MIDI Machine Control commands received on the `ardour:MMC in` JACK port.
  - Send MMC commands when enabled Ardour will send MIDI Machine Control commands on the `ardour:MMC out` JACK port.
  - Inbound MMC device ID: is the only device ID Ardour will respond to when an MMC command is received on the `ardour:MMC in` JACK port.
  - Outbound MMC device ID: is the MIDI device ID Ardour will use when it sends MMC commands.

## Control Surfaces

This tab contains settings for *control surfaces*.

It lists all the Control Surface protocols Ardour knows. To enable a Control Surface Protocol, the Enable checkbox next to its name should be ticked. Editing the settings related to this protocol can be done by double-clicking its name or clicking the Show protocol settings (only for Generic MIDI and Open Sound Control).

## MIDI Ports

- MIDI Port Options
  - MIDI input follows MIDI track selection allows Ardour to automatically connect the MIDI input to the selected track. Selecting a different MIDI track results in Ardour disconnecting the MIDI device from the former track and connecting it to the newly selected one, so that the MIDI device is always connected to the selected track. Which MIDI device will follow selection can be chosen below.
- MIDI Inputs This is a list of all the MIDI devices connected as inputs (capture devices) to Ardour. For each devices, there are 3 checkboxes:
  - Music Data if checked, Ardour will consider this device as a source for musical data input (notes, etc...)

- Control Data if checked, Ardour will consider this device as a source for control data input (play/stop, etc...)
  - Follow selection if the above MIDI input follows MIDI track selection is checked, Ardour will make this device follow track selection.
- **MIDI Outputs** This is a list of all the MIDI devices connected as outputs (playback devices) to Ardour. For each devices, there are 2 checkboxes:
    - Music Data if checked, Ardour will consider this device as a target for musical data output (notes, etc...)
    - Control Data if checked, Ardour will take this device as a target for control data output (play/stop, etc...)

## Plugins

The content of this preference page varies heavily between versions or Ardour: both the platform and the build-time options can make Ardour support some types of plugins and not others. While this documentation tries to show all possible options, most systems will only show a subset of the options hereunder, e.g. AudioUnits are macOS only...

- Scan/Discover
  - Scan for Plugins will initiate an immediate scan of the system for available plugins. Useful to get a newly installed plugin recognised by Ardour.
- General
  - Always Display Plugin Scan Progress When enabled a popup window showing plugin scan progress is displayed for indexing (cache load) and discovery (detect new plugins).
  - Silence plugins when the transport is stopped when stopping playback or recording, if this option is checked, the plugins that still emit sound (reverbs, etc...) will be stopped. If unchecked, the plugins will continue playing after the transport stop.
  - Make new plugins active when enabled, any plugin added to a track will be in active mode. If unchecked, the plugins will be added in inactive mode by default, hence have no processing effect on the track/bus.
  - Limit automatable parameters per plugin: as some plugins (synthesizers, ...) have a lot of parameters, and those parameters can be automated by Ardour, checking this will limit the number of parameters that are listed as automatable, hence making the lists shorter and the GUI more responsive.
- Plugin GUI
  - Automatically open the plugin GUI when adding a new plugin shows the plugins GUI as soon as it is added to the processing box. If unchecked, the plugin will be added in the processor box but the GUI will only be shown when double clicking it.
  - Show Plugin Inline Display on Mixer strip by default allows Ardour to show, in the *mixer strips*, a visual rendering of the effect. These Inline Display are a special feature of Ardour, so not all plugins are able to show this display. Most of Ardour's *own plugins* have an Inline Display. At any time, the plugin's Inline Display can be toggled on or off by double-clicking it.
  - Don't automatically open the plugin GUI when the plugin has an inline display mode: this option, available only if Automatically open the plugin GUI when adding a new plugin is checked, supercedes it and hides the plugin GUI at creation if it has an Inline Display, like Ardour's own **a-\*** plugins.

- Instrument
  - Ask to replace existing instrument plugin: if a MIDI track already has an instrument (i.e. MIDI to audio converter of some sort) and this option is checked, Ardour will detect it and offer to replace the existing instrument with the newly added one, avoiding a possible conflict.
  - Interactively configure instrument plugins on insert: when inserting a multichannel instrument plugin, if this option is checked, prompts the user for the channel configuration for this plugin.

## VST

- VST
  - Enable Mac VST support (requires restart or re-scan) makes a MacOs system able to run VST-Mac plugins. As stated, a new scan for plugins is required, be it manually or by restarting Ardour.
  - Scan for [new] VST Plugins on Application Start When enabled new VST plugins are searched, tested and added to the cache index on application start. When disabled new plugins will only be available after triggering a ‘Scan’ manually.
  - Verbose Plugin Scan: adds information about the plugin in the *Log window*.
  - Scan Time Out Specifies the default timeout for plugin instantiation. Plugins that require more time to load will be blacklisted. A value of &infinity; disables the timeout.
  - VST Cache: Clicking the Clear button removes all VST plugins from the list of plugins available to be inserted into the processor box. A new VST plugin scan is then required.
  - VST Blacklist: Clicking the Clear button makes blacklisted VST plugins available to be added to the processor box.
  - Linux VST Path: Clicking the Edit button pops up a dialog to manage the directories that will be searched for Linux VST plugins. When modified, Ardour will offer to scan those paths for plugins.
  - Path: are the paths chosen above.
  - Windows VST Path: Clicking the Edit button pops up a dialog to manage the directories that will be searched for Windows VST plugins. When modified, Ardour will offer to scan those paths for plugins.
  - Path: are the paths chosen above.

## Audio Unit

- Audio Unit
  - Scan for [new] AudioUnit Plugins on Application Start When enabled, new AU plugins are searched, tested and added to the cache index on application start. When disabled, new plugins will only be available after triggering a ‘Scan’ manually.
  - AU Cache: Clicking the Clear button removes all AU plugins from the list of plugins available to be inserted into the processor box. A new AU plugins scan is then required.
  - AU Blacklist: Clicking the Clear button makes blacklisted AU plugins available to be added to the processor box.

## Appearance

- Graphics Acceleration
  - Possibly improve slow graphical performance (requires restart) Ardour uses hardware accelerated gradient creation by default for speed. Sometimes though, a buggy driver can cause this to make the system slow or unstable. Checking this will make Ardour draw its own gradients without hardware acceleration, improving stability and responsiveness on those buggy systems.
- Graphical User Interface
  - Highlight widgets on mouseover, when checked, makes Ardour's widgets (buttons, sliders, ...) slightly change color when the mouse hovers them, visually indicating what a mouse action would interact with.
  - Show tooltips if mouse hovers over a control when checked, displays a little help bubble about the control the mouse hovers. The mouse pointer needs to stay idle for about 1 sec for the tooltip to appear.
  - Update clocks at TC Frame rate: Ardour updates its clocks every 100 ms. Checking this will make the clock refresh at every TimeCode frame which is more responsive, at the cost of a bit more system stress.
  - Blink Rec-Arm buttons: when enabled, the record-armed buttons on tracks will blink when they are armed but not currently recording. When disabled, the record-armed buttons on tracks will be outlined in red instead of blinking. The global record-arm button in the *Transport bar* is unaffected.
  - Blink Alert indicators: when enabled, the Alert indicators (like the Error Log or the Feedback button) will blink when they are active (when an error or feedback has been detected, respectively). When disabled, the indicators will turn red instead of blinking.
  - GUI and Font scaling: allows the display size of most of the text and buttons in the user interface to be scaled up or down. May require a restart to take effect.

## Editor

- General
  - Use name highlight bars in region displays (requires a restart): when enabled, the region name is displayed, in the editor, in its own opaque bar at the bottom of the region. When disabled, the region name is overlaid at the top of the region, possibly over audio waveforms or MIDI notes.
  - Region color follows track color: when enabled, the background color of regions in the editor will be displayed using the color assigned to the track. When disabled the default region background color will be used.
- Waveforms
  - Show waveforms in regions when enabled shows a visual representation of the region's audio waveform.
  - Show waveform while recording when enabled, will draw the audio waveform in regions being recorded, in near real time. When disabled, only a region block will be drawn while recording, reducing CPU requirements.
  - Show waveform clipping: when enabled the waveform displayed will show peaks marked in red if they exceed the clip level.

- Waveform Clip Level (dBFS): sets the level at which the waveform shown in an audio region will be drawn in red to indicate clipping. Setting lower than 0.0 dBFS can be useful if any tool in the audio chain has problems near 0.0 dBFS.
- Waveform scale: when waveforms are shown in audio regions, they can be displayed using a *linear* or a *logarithmic* scale. See *Waveform display*.
- Waveform shape: when waveforms are shown in audio regions, they can be displayed using a *traditional* or a *rectified* shape. See *Waveform display*.
- Editor Meters
  - Show meters in track headers, when enabled, shows a small meter in the Editor’s *track headers*. The meter is shown on the right side area of the header and provides an instant, if unprecise, view of the levels on this track/bus.
  - Limit track header meters to stereo: if a track has more than two outputs (e.g. with a drum plugin), limits the number of meters in the track header to the first two ones. Only affects audio meters, not MIDI.
- MIDI Regions
  - Display first MIDI bank/program as 0: when patches and bank changes are displayed in the editor, if this option is checked, the numbering will be zero-based instead of one-based, i.e. banks/programs will be numbered 0, 1 ,2... instead of 1, 2, 3...
  - Don’t display periodic (MTC, MMC) SysEx messages in MIDI Regions: if checked, will hide these control messages from the MIDI regions for better legibility.

## Mixer

- Mixer Strip
  - This table enables (checked) or disables (unchecked) the display of controls in the *mixer strip*. Controls whose display can be toggled are: **Input**, **Phase Invert**, **Record & Monitor**, **Solo Iso/Lock**, **Output**, **Comments** and VCA Assigns.
  - Use narrow strips in the mixer for new strips by default When enabled, new mixer strips are created in narrow format. When disabled, they are created in wide format. Existing mixer strips width can be toggled with the width control at the top left of the mixer strip.

## Toolbar

- Main Transport Toolbar Items: this section allows to toggle the visibility of some elements of the main toolbar:
  - Display Record/Punch Options toggles the visibility of the *punch and record* slice of the main toolbar.
  - Display Monitor Options toggles the visibility of the *monitor options* slice of the main toolbar.
  - Display Selection Clock toggles the visibility of the *selection clocks* slice of the main toolbar.
  - Display Secondary Clock toggles the visibility of the *secondary clocks* slice of the main toolbar.
  - Display Navigation Timeline toggles the visibility of the *navigation/mini timeline* slice of the main toolbar.
  - Display Master Level Meter toggles the visibility of the *selection clocks* slice of the main toolbar.

- Lua Action Script Button Visibility enables or disables the visibility of the four columns of *Lua script buttons*. Each column contains two user-assignable buttons.

## Theme

- Theme
  - Draw “flat” buttons: when enabled, button controls in the user interface will be drawn with a flat look. When disabled button controls will have a slight 3D appearance.
  - LED meter style if checked, the bar meters in the editor and mixer will be styled to look like if they were made of LEDs, with a dotted bar. Unchecking this option makes the bars flat and continuous.
  - Waveforms color gradient depth: determines how much gradient effect is applied to the inner of audio waveforms displayed in the editor. Values range from 0.0, no gradient effect, to 1.0, maximum effect.
  - Timeline item gradient depth: Determines how much gradient effect is applied to the backgrounds of regions displayed in the editor. Values range from 0.0, no gradient effect, to 1.0, maximum effect.
  - Icon Set: Changes the mouse cursor icons used to indicate different tool modes in the editor. An example would be the icons used to indicate whether the cursor will select a region or change the length of a region.

## Colors

- Colors
  - Color Theme allows to switch between some presets bundled with Ardour, changing both the palette and items colors, hence styling Ardour all at once.
  - The table allows to change the color settings in Ardour by acting on three parameters:
    - \* Items that allow to choose any color from the palette (see below) to color a UI element. Clicking on a color sample in the **Color** column bring up the Palette, to choose from.
    - \* Palette that allows to create a set of colors that will be used in the UI. Using a palette allows for better consistency, instead of picking “free” colors for each UI element. Clicking on a color patch brings up a full color selector, to assign this color to this patch of the palette.
    - \* Transparency where possible, allows to select, with a slider, the transparency of the UI element, with 0 (slider to the left) being fully opaque.
  - Restore Defaults turns all the palette, item colors and transparency back to Ardour’s default base setting, in case Ardour’s appearance has turned into a toddler’s toy.

## Quirks

- Various Workarounds for Windowing Systems: As Ardour is available on a number of platforms and windowing systems, some specific workarounds are sometimes required to provide a smooth experience to the user.
  - Use visibility information provided by your Window Manager/Desktop allows the system window manager’s rules for the windows visibility to supercede Ardour’s.

- All floating windows are dialogs: when enabled, Ardour will use type “Dialog” for all floating windows instead of using type “Utility” for some of them. This may help usability with some window managers. This setting requires a restart of Ardour to take effect.
- Transient windows follow front window.: when enabled, transient windows will follow the front window when toggling between the editor and mixer. This setting requires a restart of Ardour to take effect.
- Float detached monitor-section window: as the *monitor section* can be detached from the mixer, this option makes it a floating window, which may be handled differently by the windowing system and easier to access.

## Video

- Video Server
  - Show Video Export Info before export Shows a warning message when exporting a video about licensing and offers to open the *export video* section of this manual.
  - Show Video Server Startup Dialog: when using video inside Ardour, this video is accessed via Xjadeo from a source file through a Video Server. This options shows the server’s startup dialog (useful for debugging a malfunctioning video).
  - Advanced Setup (remote video server) can be used when the setup is more complex than opening a local file with Ardour. The tools used behind the scene by Ardour allow a lot of flexibility, so for a competent user, the options below are provided to access a distant file (i.e. on another machine). The default options for the two following fields (“`http://localhost:1554`” and “`/`”) are suitable for local files.
  - Video Server URL: Base URL of the video server delivering the video through the network (`http://IP-or-address:port`).
  - Video folder is the server’s local path to the document-root, i.e. the files that can be delivered by the server.
- Video Monitor
  - Custom Path to Video Monitor (xjadeo) - leave empty for default: Ardour bundles offer xjadeo bundled, so it should run flawlessly. Though, for custom builds or if a newer version of xjadeo is available, one can specify a path to the wanted version of xjadeo.

## 9 - Session Properties

Session properties control aspects of the workflow or configuration that pertain to the current session only; these settings are initially set from the template used to create the session. They can be found in Session > Properties, and are stored in the session file.

### 9.1 - Timecode Tab

This tab is used to change how Ardour interprets and manipulates timecode.

- Timecode Settings lets you set the number of frames per second and pull up/down to match the timecode used other synchronized systems.
- External Timecode Offsets allows Ardour to a fixed offset from other synchronized systems. Slave Timecode offset adds the specified offset to the received timecode (MTC or LTC). Timecode Generator offset adds the specified offset to the timecode generated by Ardour (so far only LTC).

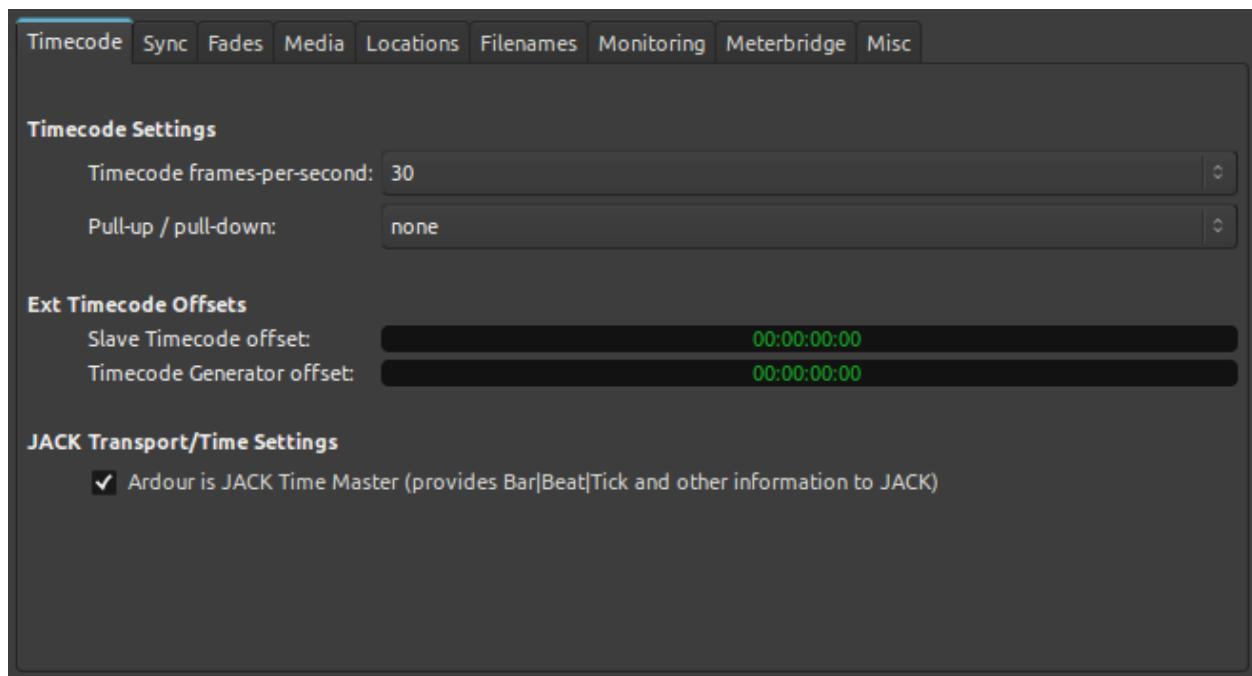


Fig. 4: The Session Properties dialog.

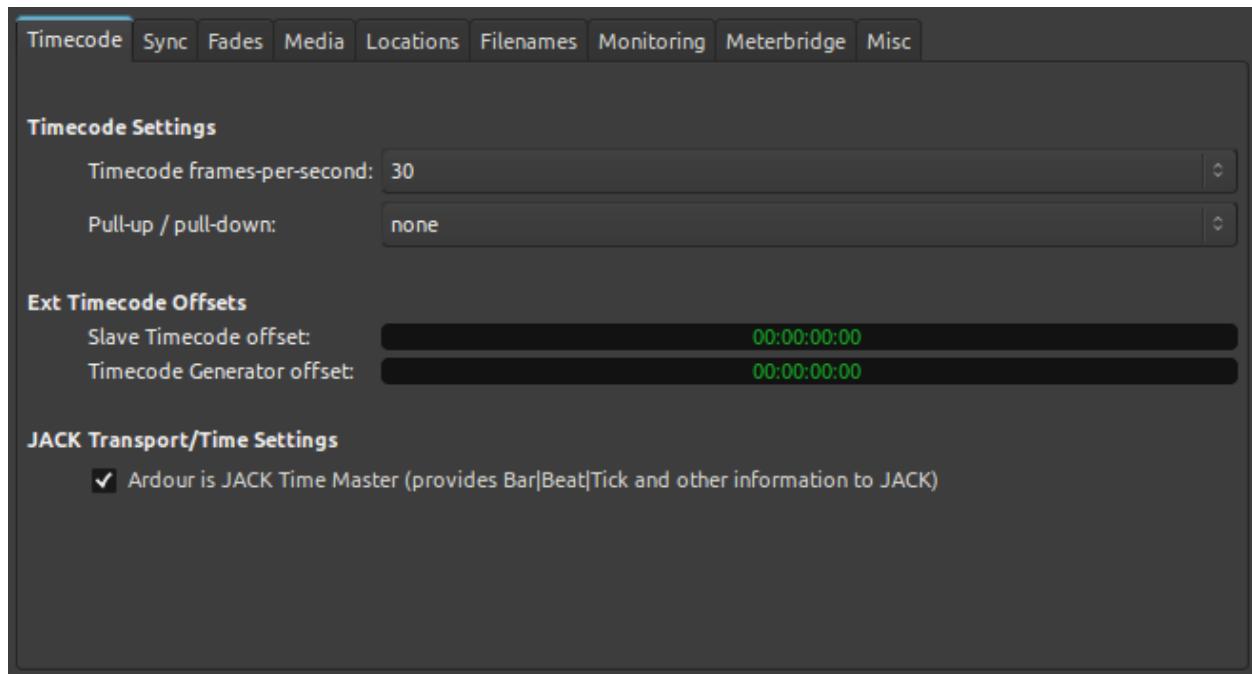


Fig. 5: Session properties timecode tab

- Jack Transport / Time Settings determines whether Ardour controls Bar|Beat|Tick and other information for Jack.

### 9.2 - Sync Tab



Fig. 6: Session properties sync tab

This tab is used to modify the timecode settings when working with video to use the imported video's timecode settings instead of the session defaults.

### 9.3 - Fades Tab

Change how Ardour works with region crossfades.

- Destructive crossfade length is used when an operation on a region is destructive, such as when recording in a track is in tape mode.
- When Region fades **active** is checked, the region fades set up in the mixer are used during playback. When unchecked, the fades are ignored.
- When **Region fades visible** is checked the region fades are visible in the the **Editor**.

### 9.4 - Media Tab

Change how sound is stored on disk. These options do not change how sound is handled internally.

- Sample format defaults to 32-bit floating point, the same as the internal representation. 24 and 16-bit integer representation are also available.
- **File type** options are WAVE, WAVE-64, and CAF.

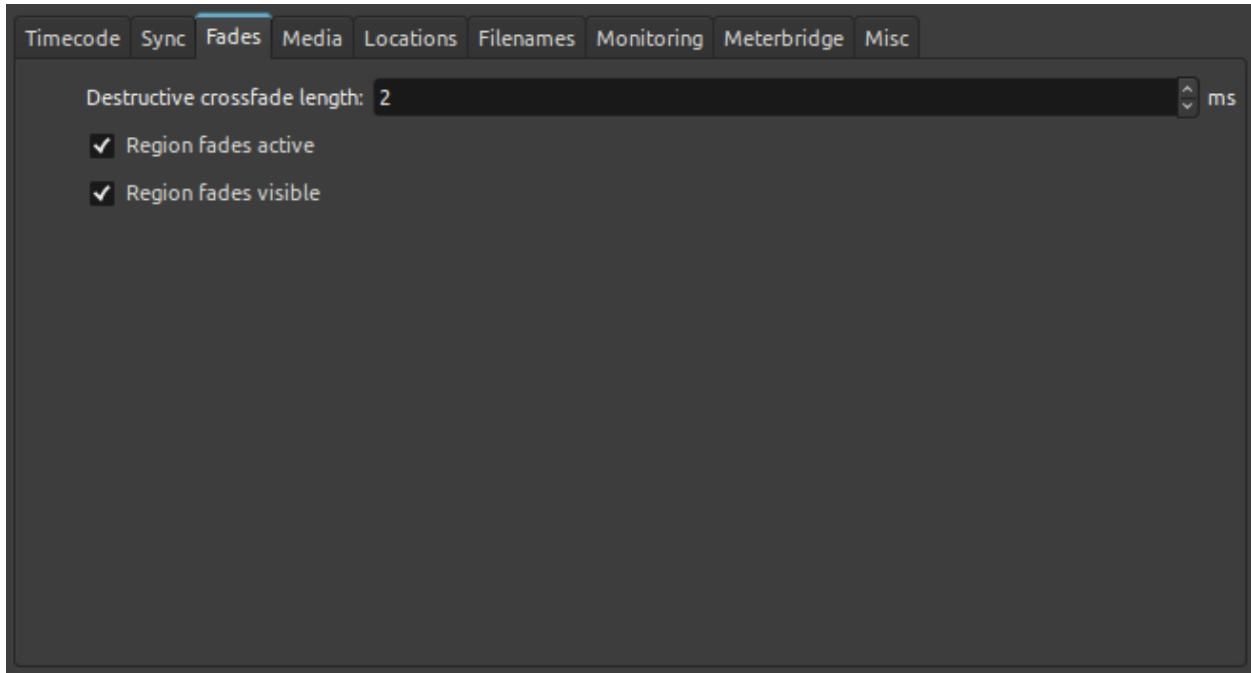


Fig. 7: Session properties fades tab

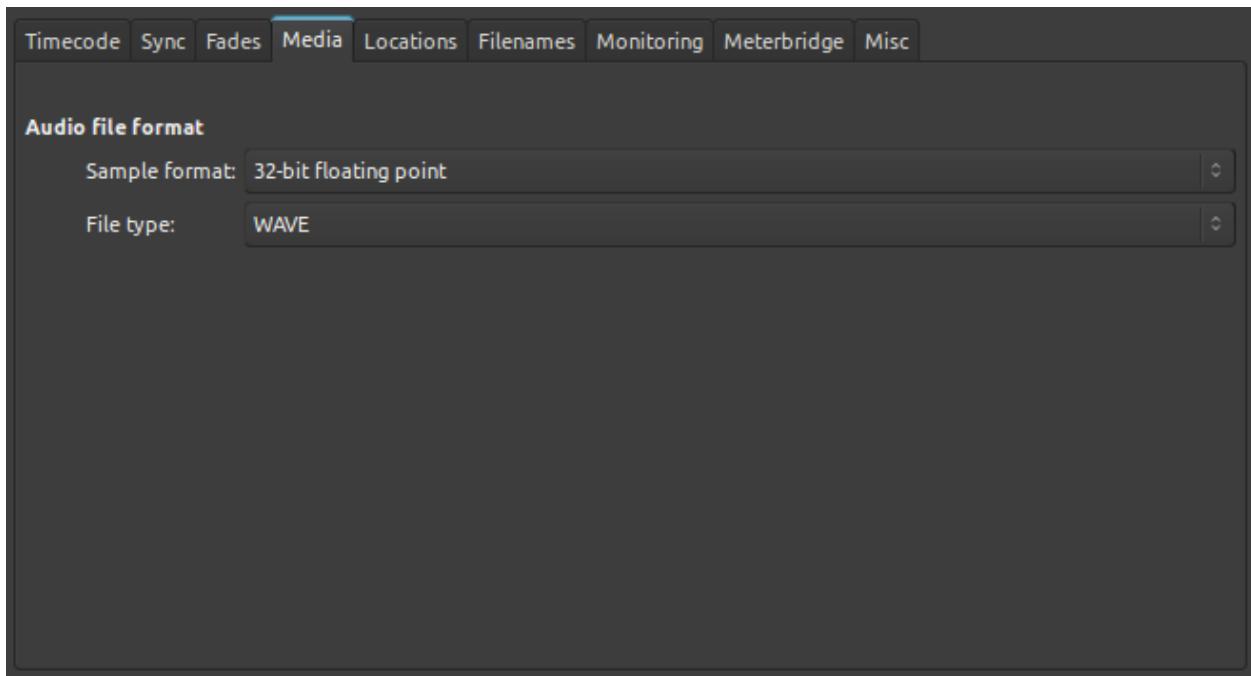


Fig. 8: Session properties media tab

## 9.5 - Locations Tab

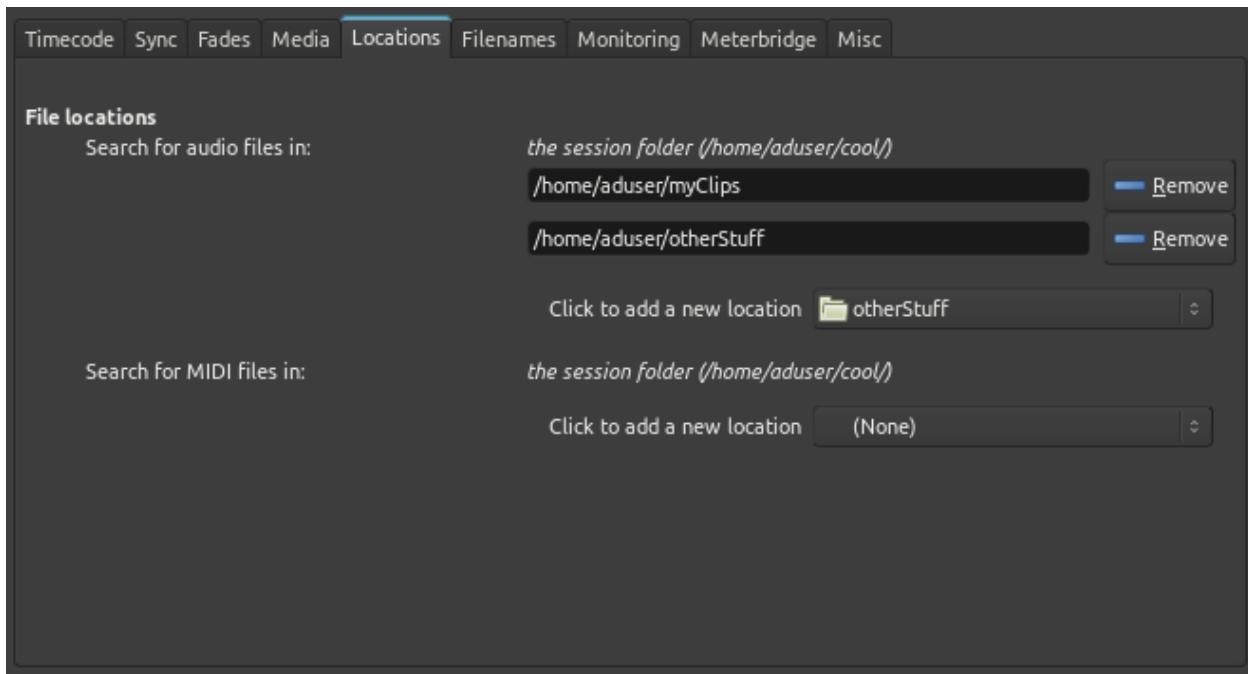


Fig. 9: Session properties locations tab

These options add file locations that will be searched to find the audio and midi files used by the session. This is useful when the files have been imported into the session but not copied into the session.

To add a location, navigate to the directory where the files are stored. Drill down into the directory and then click open. The directory will show up in the dialog. The remove button next to the added directory can be used to remove it from the search path.

## 9.6 - Filenames Tab

This tab is used to change how Ardour names recorded regions. If Prefix track number is selected a unique number will appear on each track in the Editor window and will prefix the region name. If the track number is 2 and the region would have been Gtr-1.1 with track number prefix turned on the region will be named 2\_Gtr-1.1 instead. See XX for base of the region name.

If Prefix take name is selected and the Take name has Take1 the region will have the name Take1\_Gtr-1.1 instead. If both boxes are checked the name will be Take1\_2\_Gtr-1.1 instead.

When Prefix take name is enabled, the first time a track is recorded it will have the specified take name. When recording is stopped, any trailing number on the end of the take name will be incremented by 1. If the track name specified doesn't have a number on the end, the number 1 will be suffixed.

## 9.7 - Monitoring Tab

Provides options affecting monitoring.

The **Track Input Monitoring automatically follows transport state** affects how input monitoring is handling. See *Monitor Setup in Ardour*.

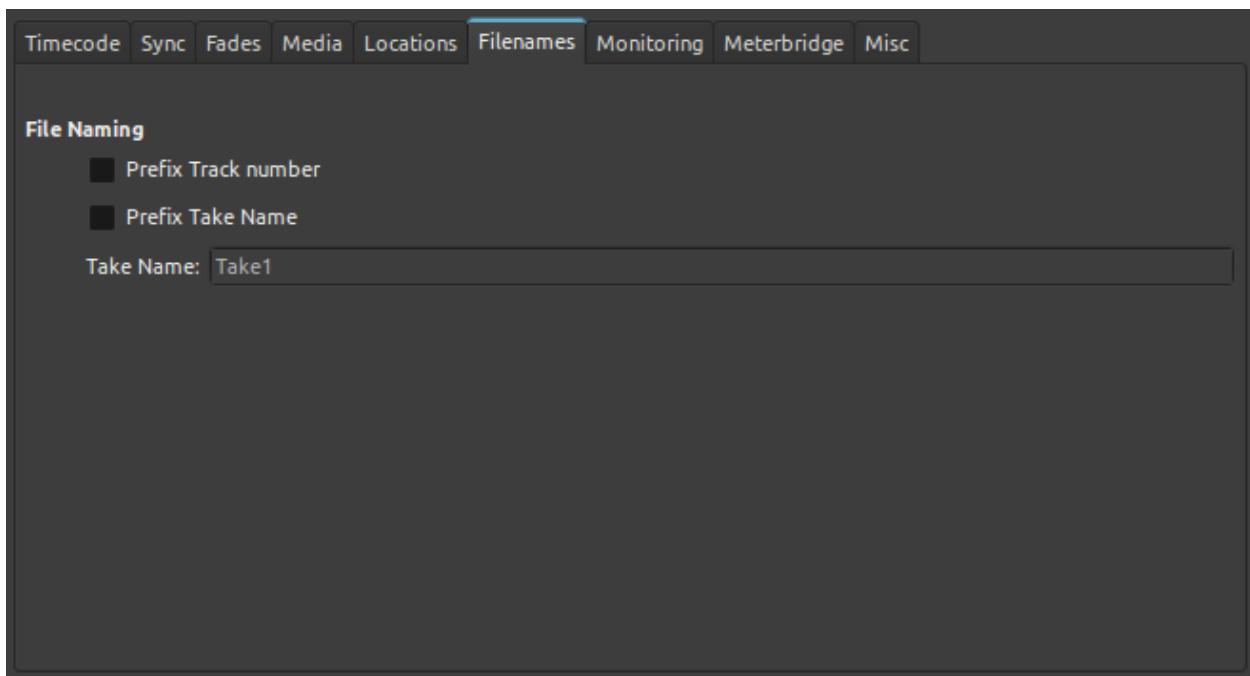


Fig. 10: Session properties filenames tab

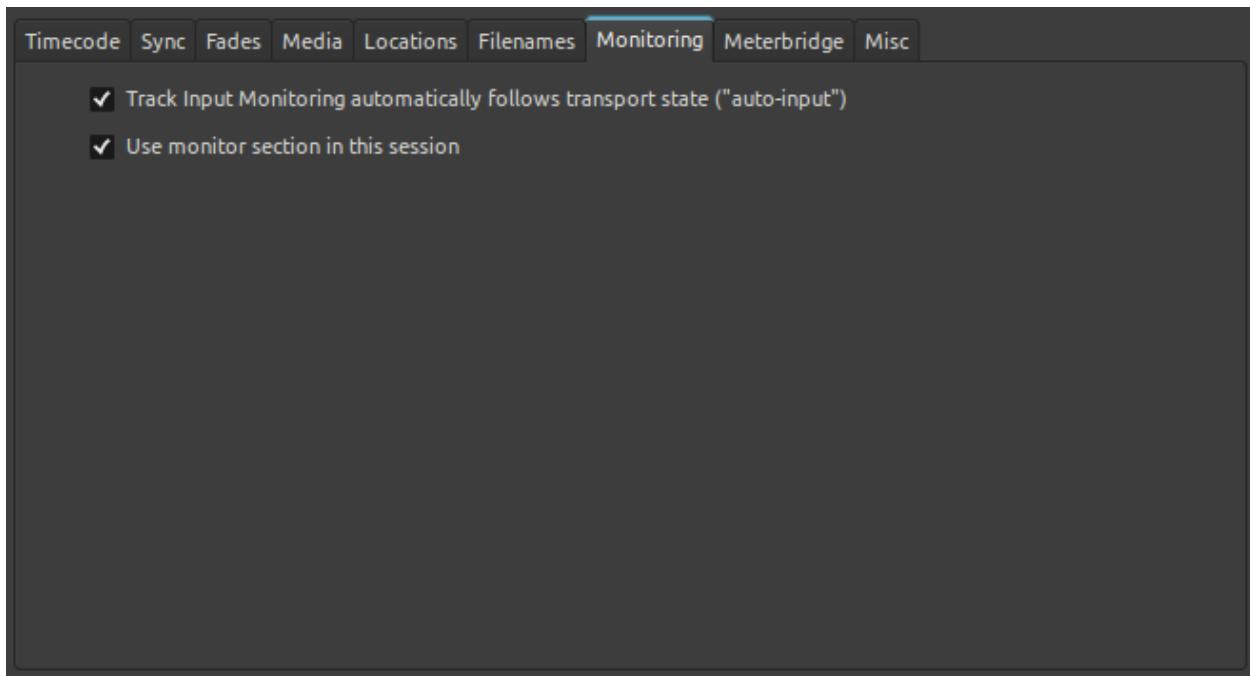


Fig. 11: The session properties 'monitoring' tab

The ‘Use monitor section’ displays an extra section in the **Mixer** window that is modelled on the similarly named section on large analog consoles. More information can be found on the *Monitor Section* page.

## 9.8 - Meterbridge Tab

The meters from audio tracks always display in the Meterbridge. This tab changes what additional controls are also displayed.

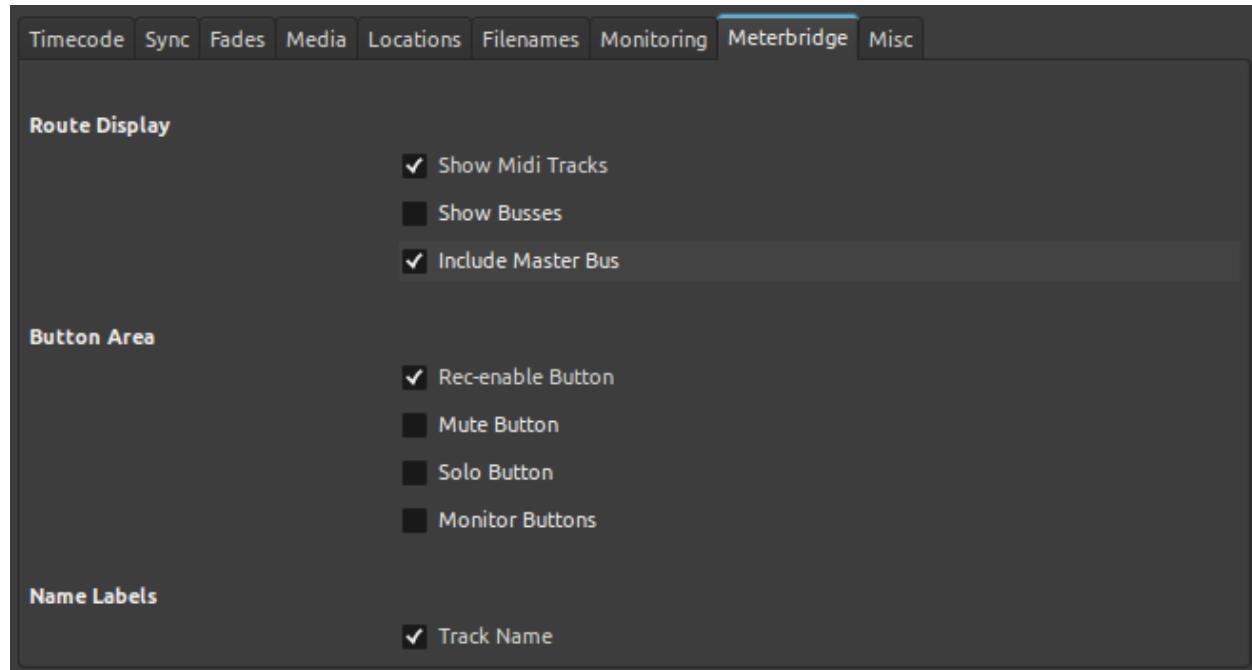


Fig. 12: Session properties meterbridge tab

- Route Display has options for showing midi tracks, busses, and the master bus.
- Button Area has options for adding record enable, mute, solo, and input monitor buttons.
- Name Labels adds the track name and, if numbers are enabled on the filenames tab, the number.

## 9.9 - Session Misc Tab

This tab has several things that don’t fit on the other tabs.

- MIDI Options
  - If MIDI region copies are independent is selected, when a MIDI region is copied or duplicated, the new region is not linked to the region it was copied from. If it is not selected, the copied regions are linked and any editing of one of the linked regions changes all of the linked regions.
  - The Editor can be configured to handle overlapping MIDI notes several ways.
    - \* never allow them
    - \* don’t do anything in particular
    - \* replace any overlapped existing notes
    - \* shorten the overlapped existing note

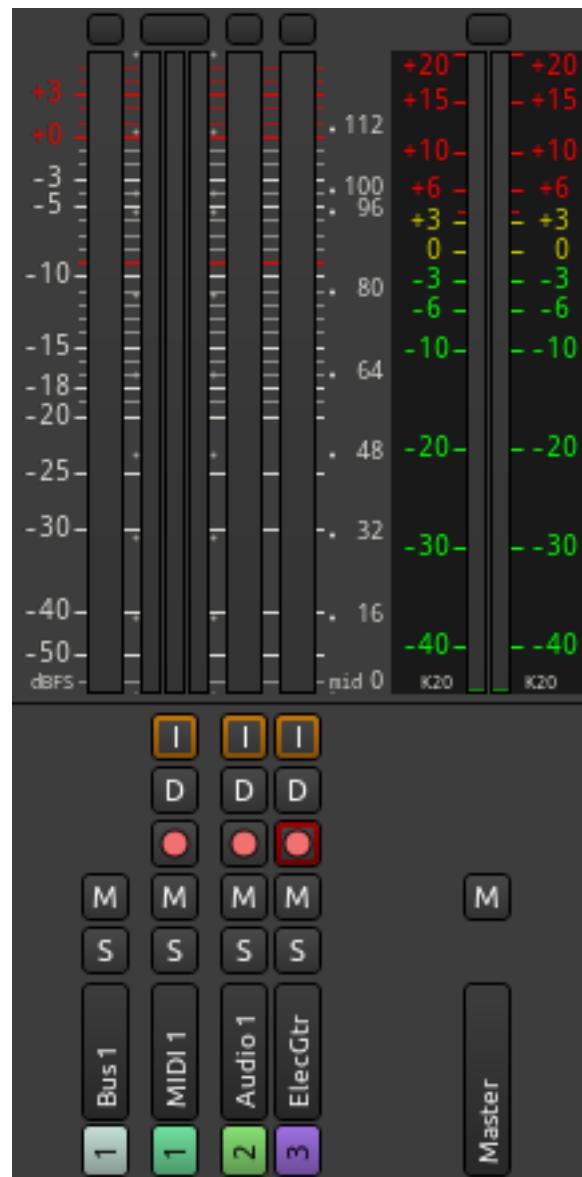


Fig. 13: Meterbridge with all options on

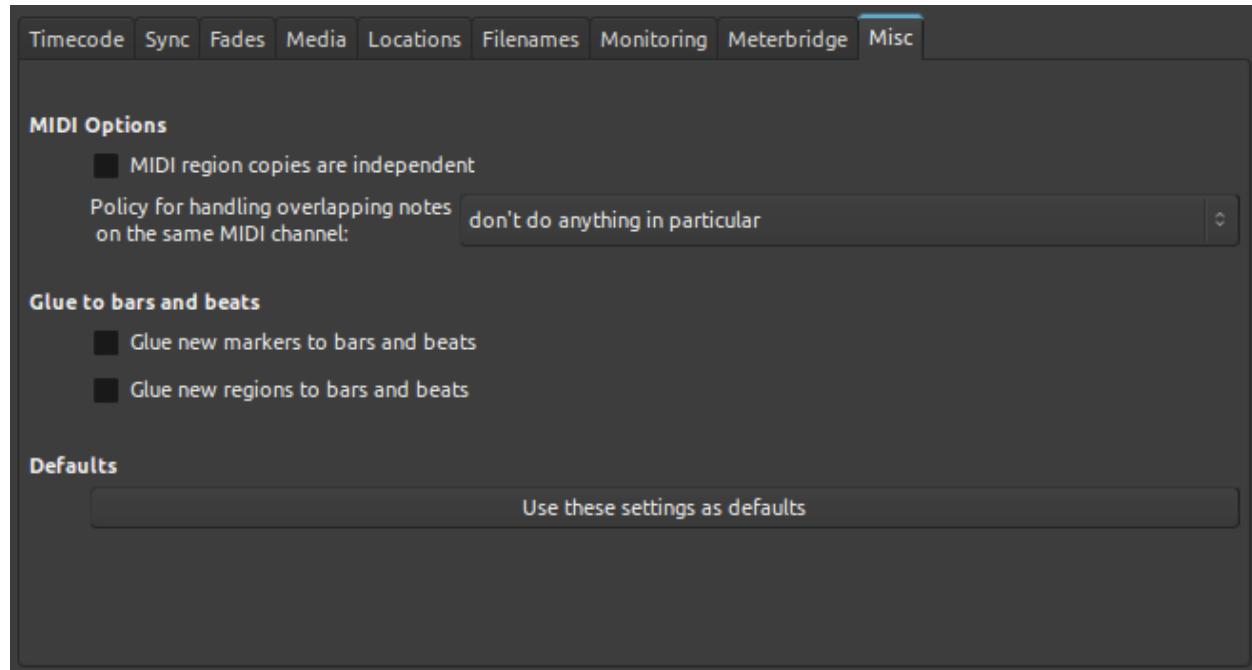


Fig. 14: Session properties misc tab

- \* shorten the overlapped new note
- \* replace both overlapping notes with a single note
- Glue to bars and beats
  - New markers can be glued to bars and beats
  - New regions can be glued to bars and beats
- Settings from the session properties dialogs can be saved to the default session template.

## 10 - Configuring MIDI

MIDI is a way to describe musical performances and to control music hardware and software.

Ardour can import and record MIDI data, and perform a variety of editing operations on it. Furthermore, MIDI can be used to control various functions of Ardour.

### MIDI Handling Frameworks

MIDI input and output for Ardour are handled by the same “engine” that handles audio input and output.

OS X	CoreMIDI is the standard MIDI framework on OSX systems.
Linux	ALSA MIDI is the standard MIDI framework on Linux systems.
Windows	There is no single standard MIDI framework on Windows, but Ardour can work with ASIO and others.

On Linux systems, QJackCtl control software displays ALSA MIDI ports under its “ALSA” tab (it does not currently display CoreMIDI ports). By contrast, JACK MIDI ports show up under the MIDI tab in

QJackCtl.

## JACK MIDI Configuration

By default, JACK will **not** automatically detect and use existing MIDI ports. One of several ways of bridging between the native MIDI frameworks (e.g. CoreMIDI or ALSA) and JACK MIDI must be chosen, as described in the following sections.

### 10.1 - MIDI on Linux

The right approach for using MIDI on Linux depends on which version of JACK is in use. The world divides into:

Systems using JACK 1, versions 0.124 or later	On these systems, JACK must be started with the <code>-X alsa_midi</code> server argument. To support legacy control applications, the <code>-X seq</code> argument to the ALSA backend of JACK can also be used to get the exact same results.
All others	Using <code>a2jmidid</code> acts as a bridge between ALSA MIDI and JACK. The <code>-X seq</code> or <code>-X raw</code> arguments should <i>not</i> be used—the timing and performance of these options is unacceptable.

### Using `a2jmidid`

`a2jmidid` is an application that bridges between the system MIDI ports and JACK.

First it must be ensured that there is no ALSA sequencer support enabled in JACK. To check that, one must open QJackCtl's Setup window and set Settings > MIDI Driver to none, then uncheck the Misc > Enable ALSA Sequencer support option. The jack server must then be restarted before going on.

### Checking for `a2jmidid` availability

Next, it must be checked whether `a2jmidid` is already installed. This is done by starting the JACK server, then going to the command line and typing:

`a2jmidid -e` If `a2jmidid` does not exist, it must be installed with the software manager of the Linux distribution in use until this command responds.

### Checking available MIDI ports

If JACK is correctly configured for MIDI, then the MIDI ports should appear in `qjackctl` under Connections > MIDI.

### Making it automatic

Once it has been verified that the ports appear in JACK as expected, this can be made to happen whenever JACK is started:

- If a newer version of JACK 1 is in use, by just making sure the `-X alsa_midi` or `-X seq` options are enabled for whatever technique is being used to start JACK.
- For other versions of JACK, by adding `a2jmidid -e &` as an “after start-up” script in the Setup > Options tab of QJackCtl, so that it is started automatically whenever JACK is started.

## 10.2 - MIDI on OS X

In order for CoreMIDI to work with Jack MIDI, a CoreMIDI-to-JACK-MIDI bridge is required. This feature is available on versions equal to or greater than version 0.89 of JackOSX.

### Routing MIDI

#### Inside Ardour

MIDI ports show up in Ardour's MIDI connection matrix in multiple locations. Bridged CoreMIDI ports as well as JACK MIDI ports that have been created by other software clients will show up under the "Other" tab. Bridged CoreMIDI hardware ports show up under the "Hardware" tab.

#### External Applications

There are multiple options for connecting MIDI ports outside of Ardour.

- [MIDI Monitor](#) is a handy tool for doing various MIDI-related tasks.
- [MIDI Patchbay](#) allows connection of ports and filters MIDI data.

## Part III - Ardour's Interface

### 11 - Ardour's Interface Overview

In Ardour, work is done in two main windows: the Editor and the Mixer.

```
' <#main-menu>'____  
' <#status-bar>'____  
' <#transport-bar>'____  
' <#selection-and-punch-clocks>'____  
' <#other-toolbar-items>'____  
' <#other-toolbar-items>'____  
' <#transport-clocks>'____  
' <#other-toolbar-items>'____  
' <#selection-and-punch-clocks>'____  
' <#mini-timeline>'____  
' <#other-toolbar-items>'____  
' <#audiomidi-mixer-strips>'____  
' <#toolbox>'____  
' <#zoom-controls>'____  
' <#grid-controls>'____  
' <#edit-point-control>'____  
' <#nudge-controls>'____  
' <#selection-and-punch-clocks>'____
```

```
' <#editor-lists>'_____
' <#ruler>'_____
' <#summary>'_____
' <#track-and-bus-groups>'_____
' <#bus-controls>'_____
' <#audio-track-controls>'_____
' <#automation>'_____
' <#audio-track-controls>'_____
' <#midi-track-controls>'_____
' <#midi-track-controls>'_____
' <#main-menu>'_____
' <#status-bar>'_____
' <#transport-bar>'_____
' <#selection-and-punch-clocks>'_____
' <#other-toolbar-items>'_____
' <#other-toolbar-items>'_____
' <#transport-clocks>'_____
' <#other-toolbar-items>'_____
' <#selection-and-punch-clocks>'_____
' <#mini-timeline>'_____
' <#other-toolbar-items>'_____
' <#favorite-plugins-window>'_____
' <#strips-list>'_____
' <#groups-list>'_____
' <#track-and-bus-groups>'_____
' <#master-bus-strip>'_____
' <#monitor-section>'_____
' <#control-masters-mixer-strips>'_____
' <#audiomidi-busses-mixer-strips>'_____
' <#audiomidi-mixer-strips>'_____
' <#audiomidi-mixer-strips>'_____

```

The Editor and Mixer windows. Click on a section to access its description. The Editor and the Mixer share the same toolbar (the top of the window). The sections displayed in this toolbar can be customized to the user's workflow, by checking options in Preferences > Appearance > Toolbar.

Switching between the Editor and the Mixer windows is done:

- with the *Mode Selector buttons* in the upper right
- with the M shortcut

- with the menu Window > Editor (*or Mixer*) > Show.

Both windows can be visible at the same time (e.g. for a multi-monitor setup) using Window > Editor (*or Mixer*) > Detach option in the same submenu.

## The Editor

The Editor window includes the editor track canvas where audio and MIDI data can be arranged along a timeline. This is the window where editing and arranging a project is done. The window has a general “horizontal” sense to it: the timeline flows from left to right, the playhead showing the current position in the session moves from left to right—the window really represents time in a fairly literal way.

It is possible to show a single channel strip in the editor window, and some people find this enough to work on mixing without actually opening the mixer window. Most of the time though, both of these windows will be needed at various stages of a session’s lifetime.

## The Mixer

The Mixer window represents signal flow and is the window that will probably be used most when mixing a session. It includes channel strips for each track and bus in the session. It has a general “vertical” sense to it: signals flow from the top of each channel strip through the processing elements in the strip to reach the output listed at the bottom.

To learn more about the process of mixing, see [Mixing](#).

## 12 - Main Menu

The Main Menu is made of 8 menus that allow the user to interact with Ardour, and below is a reference of all the menu actions.

Notice that a lot of this menus, and probably all of the most used ones have keyboard shortcuts to ease and fasten the work. A sum-up of these shortcuts can be found on the [Defaults Keyboard Bindings](#) page, and every menu in Ardour can be reassigned to an user defined key binding, or used via [OSC](#) with a control surface.

### 12.1 - The Session Menu

The Session menu groups together everything related to the session and the file operations.

New...	Creates a <i>new</i> session
Open...	Opens an existing session
Recent...	Opens a list of recent sessions that can be opened
Close	Closes the current session (but not Ardour)
Save	Saves the current session
Save As...	Saves to a new session (with options)
Rename...	Changes the name of the session
Snapshot (& keep working on current version)...	Create a <a href="#">Snapshot</a> but any subsequent change will be saved to this session
Snapshot (& switch to new version)...	Same thing, and any subsequent change will be saved to this new snapshot
Save Template...	Saves the session as a <a href="#">template</a> , without the audio
Archive...	Exports the session as a <i>compressed file</i> for archiving or sharing purposes
Metadata	

Table 1 – continued from previous page

Edit Metadata...	Opens the <i>Metadata</i> window, where information about the session can be edited.
Import Metadata...	Creates the metadata by extracting them from another session
Add Track, Bus or VCA...	Adds a <i>new track/bus/VCA</i> to the session, same as the Track > Add Track menu
Import	Opens the <i>Import</i> windows, to add media to the session
Import PT session	Import a ProTools© session file. Not everything in the original session can be imported.
Scripting	
Add Lua Script...	Loads or adds a <i>Lua Session script</i> to the current session
Remove Lua Script	Removes a loaded Lua Session script from the session
Open Video...	Imports a <i>video file</i> in the session
Remove Video	Removes the video part of the session (the video timeline disappears)
Export	
Export to Audio File(s)...	<i>Export</i> all or part of the session in audio form
Stem export...	<i>Exports each track</i> as its own audio file (for e.g. DAW interchange)
Export to Video File	Exports the session to a <i>video file</i>
Clean-Up	
Bring all media into session folder	Copies all the media files imported from outside the session folder in the session folder
Reset Peak Files	Reinitializes the buffered images representing the audio files
Clean-up Unused Sources...	Quarantines all the media files not used in the session to a specific sub-directory
Flush Wastebasket	Deletes those quarantined files
Properties	Shows the <i>Session Properties</i> dialog, allowing to fine-tune the parameters of the session
Lock	Locks the session by showing an Unlock window that (until clicked) blocks Ardour from exiting
Quit	Exits Ardour. Prompts for saving the session if it has been modified.

## 12.2 - The Transport Menu

The Transport menu handles how Ardour handles the playback and playhead.

Start/Stop	Starts or stops the playhead, and recording if it is armed
Play	
Play Selection	Only plays the selected part of the session, be it a range or a selection
Play Selection w/Preroll	As the previous menu, except it starts the playback a few bars before the start of the selection
Start/Continue/Stop	Leaves loop play or range play mode but without stopping playback
Play from Edit Point and Return	Starts the playback at the <i>Edit point</i> , and when stopped, goes back to the edit point
Play Loop Range	If a <i>Loop range</i> is defined, play it and loop until stopped
Start Recording	This is a shortcut to trigger the global recording, and start playback
Stop and Forget Capture	Stops the recording, removes the newly created material, and forgets it
Enable Record	Triggers the global recording. Next time “Play” is pressed, it will start recording
Record w/Preroll	As the Start Recording menu, except it starts the recording a few bars before the start of the selection
Record w/Count-In	As the Start Recording menu, except it waits for 2 bars before starting the recording
Set Loop from Selection	Converts the selection into a <i>Loop range</i> by placing loop markers at the start and end of the selection
Set Punch from Selection	Same thing, for <i>Punch</i>
Set Session Start/End from Selection	Same thing, for the start and end markers of the session, defined by the selection
Forward	Plays the audio forwards from the playhead on. If the audio is reversed, it plays it forwards
Rewind	Plays the audio backwards from the playhead on. If the audio is reversed, it plays it backwards
Transition to Roll	Plays the audio forwards, with a speed of 1 (real time)
Transition to Reverse	Plays the audio backwards, with a speed of 1 (real time)
Playhead	
Playhead to Mouse	Set the position of the playhead at the current position of the mouse
Playhead to Active Mark	If a marker is selected, set the position of the playhead at the position of the active marker

Table 2 – continued from previous page

Center Playhead	Centers the view on the playhead without changing the zoom level
Nudge Playhead Forward	Shifts the position of the playhead to the right by the amount specified in the Nudge value
Nudge Playhead Backward	Same thing, to the left
Move to Next Transient	When transient have been set, moves the playhead to the next transient
Move to Previous Transient	Same, to the left
Playhead to Next Grid	Regardless of the state of the Grid Mode, goes to the next grid boundary
Playhead to Previous Grid	Same, to the left
Playhead to Next Region Boundary	Moves the playhead to the right to the next beginning or end of region
Playhead to Previous Region Boundary	Same, to the left
Playhead to Next Region Boundary (No Track Selection)	Moves the playhead to next beginning or end of region, bypassing track selection
Playhead to Previous Region Boundary (No Track Selection)	Same, to the left
Playhead to Next Region Sync	Moves the playhead to next Region Sync Point, that is by the amount specified in the Region Sync value
Playhead to Previous Region Sync	Same, to the left
Jump to Next Mark	moves the playhead to the next <i>marker</i> on the Ruler
Jump to Previous Mark	Same, to the left
Go to Zero	Sends the playhead to the 00:00:00:00 time, regardless of the current session marker
Go to Start	Sends the playhead to the Start marker of the session
Go to End	Sends the playhead to the End marker of the session
Go to Wall Clock	Sends the playhead to the current value of system time, as defined in the system's clock
Active Mark	
To Next Region Boundary	Moves the currently selected <i>marker</i> to the next region boundary
To Previous Region Boundary	Same, to the left
To Next Region Sync	Moves the currently selected to the next region sync point
To Previous Region Sync	Same, to the left
Markers	
Add Mark from Playhead	Creates a Marker at the position of the playhead
Remove Mark at Playhead	Removes any marker at the position of the playhead
Toggle Mark at Playhead	Combine the 2 previous: if a marker exists, deletes it, otherwise adds one
Locate to Mark <i>n</i>	If it exists, goes to the <i>n-th</i> marker
Set Session Start from Playhead	Puts the Start of the session marker at the playhead's position
Set Session End from Playhead	Puts the End of the session marker at the playhead's position
<input type="checkbox"/> Time Master	Sets Ardour as the Time master, i.e. Ardour sends the time signal to other DAWs
<input type="checkbox"/> Punch In/Out	Based on the Punch in and Punch out markers if they exist
<input type="checkbox"/> Punch In	Based on the Punch in marker, only allow to record from that point
<input type="checkbox"/> Punch Out	Based on the Punch out marker, forbids recording before that point
<input type="checkbox"/> Audio Input	If checked, automatically switch the <i>monitor</i> from <i>input</i> to <i>output</i>
<input type="checkbox"/> Follow Edits	If checked, selecting a region moves the playhead to its beginning
<input type="checkbox"/> Auto Play	If checked, moving the playhead in the ruler starts the playback
<input type="checkbox"/> Auto Return	If checked, when the playback is stopped, go back to the previous position
<input type="checkbox"/> Click	Activates/deactivates the click track (metronome)
<input type="checkbox"/> Follow Playhead	If checked, while playing, when the playhead reaches the end of a region, it loops back to the start
<input type="checkbox"/> Stationary Playhead	If checked <i>and</i> if Follow playhead is checked, on playback, the playhead stays at the current position
Panic	Immediately stops all MIDI playback (useful e.g. when a MIDI message is received)

### 12.3 - The Edit Menu

The Edit menu groups together the actions related to the edition, and so will be mostly used while in Editor mode.

Undo ( <i>action</i> )	Reverts the last editing operation, namely <i>action</i>
Redo	Does the last editing operation again, after an Undo
Undo Selection Change	Reverts the last selection operation
Redo Selection Change	Does the last selection operation again after an Undo Selection Change
Cut	Deletes the current selection, but puts it in memory ready to be pasted
Copy	Copies the current selection to memory
Paste	Pastes the memory at the <i>edit point</i> , after a Cut or Copy operation
Select	
Select All Objects	Selects all the regions and automation points in the session
Select All Tracks	Selects all the tracks, busses and control masters in the session
Deselect All	Deselects all objects or tracks, nothing is selected
Invert Selection	Select the previously unselected regions, and deselect the previously selected ones
Set Range to Loop Range	Creates a range selection on the selected tracks, based on the selected loop markers
Set Range to Punch Range	Same as above, based on the selected punch markers
Set Range to Selected Regions	Same as above, based on the selected regions (i.e. from the start of the first to the end of the last)
Select All After Edit Point	Select all the regions and automation points that exist after the Edit Point
Select All Before Edit Point	Same as above, but before the Edit point (i.e. to the left of it)
Select All Overlapping Edit Range	Select all the regions and automation points of which at least a part is overlapping the edit range
Select All Inside Edit Range	Selects all the regions that are completely inside the selection range, i.e. overlapping the edit range
Select All in Punch Range	Selects all the regions of which a part is in the punch range. If some tracks have no punch markers, they are ignored
Select All in Loop Range	Same as above, based on the loop range
Move Range Start to Previous Region Boundary	Extends the left boundary of the range to the left to the next region start
Move Range Start to Next Region Boundary	Same as above, to the right (reduces the selection)
Move Range End to Previous Region Boundary	Same as above, with the right edge of the range, to the left (reduces the selection)
Move Range End to Next Region Boundary	Same as above, with the right edge, to the right (extends the selection)
Start Range	Sets the left edge of the range to the Edit point
Finish Range	Sets the right edge of the range to the Edit point
Select Next Track or Bus	Select the track or bus under the currently selected one. If multiple tracks are selected, the first one is selected
Select Previous Track or Bus	Same as above, with the track/bus above the first one selected.
Delete	Deletes all that is currently selected
Crop	Cuts the parts of the regions that are outside the range boundaries. Only applies to regions
Split/Separate	Cuts the selected regions at the Edit point, separating them in two regions
Separate	
Separate Under	Removes all the parts of the regions that are under the selected one. Only applies to regions
Separate Using Loop Range	Cuts the selected regions or the regions on the selected tracks along the loop range markers
Separate Using Punch Range	Same as above, with the Punch range markers
Align	
Align Start	Moves the selected regions to align the beginning of the regions to the Edit point
Align Start Relative	When multiple regions are selected, moves all the regions together as a group
Align End	Moves the selected regions to align the end of the regions to the Edit point
Align End Relative	When multiple regions are selected, moves all the regions together as a group
Align Sync	Moves the selected regions to align the Sync point of the regions to the Edit point
Align Sync Relative	When multiple regions are selected, moves all the regions together as a group
Fade	
Fade Range Selection	For all the regions that either begin or end in the range, create a fade in or out
Set Fade In Length	If the edit point is within the region boundaries, adjusts selected audio length
<input type="checkbox"/> Fade In	Toggles the fade in on the selected region on or off
Set Fade Out Length	Same as above, for the fade out
<input type="checkbox"/> Fade Out	Toggles the fade out on the selected region on or off
Remove Last Capture	Destroy the last recording. A prompt reminds the user this <i>cannot</i> be undone

Table 3 – continued

Edit point	
Change Edit Point	Toggles between the mouse and the playhead as the Edit point
Change Edit Point Including Marker	Toggles between the mouse, the playhead and marker as the Edit point
Snap Mode	
( <input type="checkbox"/> No Grid	Disables <i>snapping</i> , i.e. allows free movement of regions and boundaries
( <input type="checkbox"/> Grid	Forces snapping, so any move of region boundary will be lined to the grid
( <input type="checkbox"/> Magnetic	If the movement of the region or boundary happens near a grid line, snap to it
Next Snap Mode	Toggles between the No Grid, Grid and Magnetic snap modes
Next Snap Choice	Circles through the snap choices, as detailed below
Previous Snap Choice	Circles through the snap choices, as detailed below, in reverse order
Next Musical Snap Choice	Circles through the musical snap choices, e.g. those expressed in bars and measures
Previous Musical Snap Choice	Same as previous, but in reverse order
Snap To	
Snap to CD Frame	The grid unit will be 1/75th of a second
Snap to Timecode Frame/Second/Minute	The grid unit will be based on the timecode settings for the session
Snap to Second/Minute	The grid unit will be based on absolute times
Snap to nth	The grid unit will be $1/n$ beats and will depend on the tempo and metronome
Snap to Beat	Same as above, whole beat
Snap to Bar	Same as above, whole bar
Snap to Mark	The grid will be made of markers
Snap to Region Start	No grid, the regions will snap to the closest region start on any track
Snap to Region End	Same as above with the regions' ends
Snap to Regions Sync	Same as above, with the Sync points (by default, start of the region)
Snap to Region Boundaries	Same as above, for both the starts and ends of regions
Tempo	
Set Tempo from Region = Bar	Computes the tempo so that the duration of the first selected region is a bar
Set Tempo from Edit Range = Bar	Same thing, with the current Range instead of a region
[ <input type="checkbox"/> Smart Object Mode	Toggles the Smart Mode, allowing the mouse to be in Range Mode in the Editor
Scripted Actions	
[ <input type="checkbox"/> Script Manager	Shows the <i>Script manager</i> , allowing to use and manage the Lua scripts
Unset #n	Deactivate the $n$ th script
Preferences	Displays the <i>Preferences</i> panels, allowing to change Ardour's behaviour

## 12.4 - The Region Menu

The Region Menu is where the user can tweak its regions, the parts of audio or MIDI that sit on the timeline.

Insert Region from Region List	If a region is selected in the Editor List, add it at the Edit point
Play	Starts playback at the beginning of the selected region(s), and stops at its(their) end
Loop	Creates a loop range on the selected region's boundaries, and starts the looped playback
Rename...	Changes the name of the region, that appears in its top left area
Properties...	Shows the Region properties window, that displays detailed information about the region
Loudness Analysis...	Shows the Audio Report/Analysis window, that displays detailed 'dBFS information < 0.000000
Spectral Analysis...	Shows the Audio Report/Analysis window, that displays a integrated spectral view of the region
Edit	
Combine	Creates a new region by joining the selected audio regions in the same track, and replaces them
Uncombine	Splits back the compound created by <i>combining</i> into its original audio regions
Pitch Shift...	Changes the tune of the audio region, by octave, semitones or percentage, based on spectrum analysis
Split/Separate	Cuts the selected regions at the Edit point, separating them in two regions

Table 4 -

Split at Percussion Onset	Allows splitting the selected regions on its Percussion Onsets marker as set by the Rhythm Ferret
Make Mono Regions	Creates mono regions out of a stereo or multichannel region by splitting it into its discrete channels
Close Gaps	Extends (or reduces) the selected regions to be perfectly aligned. Optionally, sets up a loop marker at the end of the region
Place Transient	Places a transient at the Edit Point. Used e.g. for the Pitch Shift... action
Rhythm Ferret...	Opens the Rhythm Ferret which is a powerful tool to sequence audio files
Strip Silence...	Opens the Strip Silence window which is a very handy tool to remove all audio under a certain threshold
Reverse	Mirrors the audio horizontally
Layering	
Raise to Top	On overlapping regions, puts the selected one(s) on top
Raise	On overlapping region, makes the selected one(s) one layer higher
Lower	Makes the selected region(s) one layer lower
Lower to Bottom	Sends the selected region to the background
MIDI	
Transpose...	On a MIDI region, shows the <i>Transpose MIDI window</i> , allowing to shift the pitch of the notes
Insert Patch Change...	Inserts a patch change at the Edit Point, allowing a change of patch, channel, program, etc.
Quantize...	Shows the <i>Quantize window</i> , allowing to perfectly align the MIDI notes to the musical grid
Legatize	Shortens or elongates the MIDI notes to make them perfectly sequential, i.e. the end of one note triggers the start of the next
Remove Overlap	Shortens or elongates the MIDI notes to make them perfectly sequential, i.e. the end of one note triggers the start of the next
Transform...	<i>Transform window</i> , that allows for mathematical operations on the midi notes
Unlink from Other copies	Makes the selected MIDI region independent, e.g. editing this region won't affect any other copy
List Editor...	Shows the <i>List Editor</i> which sequentially lists all the MIDI events in the region, and allows for editing them
Gain	
<input type="checkbox"/> Opaque	When checked, makes the region opaque audio-wise, i.e., the underlying regions won't be visible
<input type="checkbox"/> Mute	When checked, mutes <i>only</i> the selected region on the track, without muting the track.
Normalize...	Shows the Normalize region dialog, which allows to scale the region level by setting its volume
Boost Gain	Increases the gain on the selected region by boosting the audio, without touching the envelope
Cut Gain	Reduces the gain without touching the envelope or automation
Reset Envelope	If the gain envelope has been edited, resets it to its initial value (constant at 0 dB)
<input type="checkbox"/> Envelope Active	When unchecked, disables any envelope editing that has been made. The envelope will be constant at 0 dB
Position	
Move to Original Position	Moves the region where it was initially recorded or inserted in the session
Snap Position to Grid	If the Grid Mode is set to <i>Grid</i> , snaps the region to the nearest grid line
<input type="checkbox"/> Lock	Locks the selected regions at their current positions in time and tracks, avoiding any movement
<input type="checkbox"/> Glue to Bars and Beats	Locks the region position to relative to the musical grid, i.e. a change of tempo will move the region
<input type="checkbox"/> Lock to Video	Same as above, relative to the position in the video
Set Sync Position	Creates or move the Sync position, i.e. the point of the region that will be aligned or synchronized to another region
Remove Sync	Removes any user defined Sync point, and resets the sync position to the beginning of the region
Nudge Later	Moves the region to the right by the amount shown in the <i>nudge timer</i>
Nudge Earlier	Same as above, to the left
Nudge Later by Capture Offset	Moves the region to the right by the capture latency computed by Ardour based on the current session settings
Nudge Earlier by Capture Offset	Same as above, to the left
Sequence Regions	Puts the selected regions one after the other, so that the end of one region is the beginning of the next
Trim	
Trim Start at Edit Point	If the Edit Point is within the region boundaries, shortens the region to align its start to the edit point
Trim End at Edit Point	Same as above, for the end of the region
Trim to Loop	Uses both the start and end Loop markers to shorten the region
Trim to Punch	Same as above with the Punch markers
Trim to Previous	On overlapping regions, shortens the selected one so that the previous region is completely removed
Trim to Next	Same as above, with the end of the selected region aligned to the start of the following region
Ranges	

Table 4 -

Set Loop Range	Creates a Loop range based on the selected regions, i.e. the start of the loop range is the start of the selected region, and the end is the end of the selected region.
Set Punch	Same as above, for the Punch range
Add Single Range Marker	Same as above, for the Edit range
Add Range Marker Per Region	For each selected region, creates its own Edit range based on the boundaries of each region.
Set Range Selection	Creates a range selection based on the boundaries of the selected regions
Fades	
<input type="checkbox"/> Fade In	Activates/deactivates the Fade In at the start of the region
<input type="checkbox"/> Fade Out	Same as above, for the Fade out at the end of the region
<input type="checkbox"/> Fades	Shortcut to activate/deactivate both the fade in and fade out
Duplicate	
Duplicate	Creates a copy of the selected region(s) and appends it to the original
Multi-Duplicate...	Shows the Duplicate dialog, allowing to create multiple copies, or a non-integer number of copies.
Fill Track	Creates duplicates until it fills the session, i.e. reaches the End marker of the session. This is useful for creating a loop.
Export...	Shows the <i>Export dialog</i> , with all parameters set to export only the selected region(s)
Bounce (without processing)	Creates a bounce, i.e. a version of the region with all the edits (boundaries, envelope), but no processing.
Bounce (with processing)	Same as above, <i>with</i> the effects of the mixer strip
Remove	Deletes the region from the edit (no file is harmed in the process, and the region stays in the session).

## 12.5 - The Track Menu

The Track menu is where one can deal with the tracks, busses and control masters.

Add Track, Bus or VCA...	Shows the <i>Add Track, Bus or VCA... window</i> , where one can add one or more tracks, busses or control masters to the session and define its parameters
Duplicate Tracks/Busses	Shows the Duplicate Tracks and Busses window, allowing to duplicate the selected track(s) and optionally, its playlist
Toggle Record Enable	Sets the Record Enable mode On on the selected track(s). These tracks will record audio/midi next time the global record is active and playback is started.
Toggle Solo	Sets the solo On on the selected tracks, so only these tracks will play
Toggle Mute	Mutes the selected tracks, they won't play until unmuted
Insert Time	Shows the <i>Insert Time window</i> , allowing to insert a blank time in the selected tracks' playlist
Remove Time	Same as <i>above</i> , but to remove time
Move Selected Tracks Up	Changes the position of the selected tracks one track up towards the top. In the mixer, the tracks will be moved to the left.
Move Selected Tracks Down	Same as above, towards the bottom
Height	
Fit Selection (Vertical)	Will fit the selected track(s) in the window. If too many tracks are selected, they'll be reduced to their minimum height.
Largest	Sets the selected tracks height to a very high value, hence making the tracks wide on screen
Larger	Same as above, but a little less high
Large	Same as above, but again less high
Normal	Sets the height of the track to its default value which is a trade-off between readability and number of tracks displayed
Small	Reduces the size of the tracks to a low value, increasing the number of on screen tracks
Toggle Active	Toggles the active state of a track. An inactive track will be grayed and won't play any sound. That can be seen in the A column of the <b>'Tracks and Busses List &lt;#the-tracks-and-busses-list &gt;'</b>
Remove	Deletes this track and its playlist (no file is harmed in the process, and the regions from the playlist stay in the Editor for later use)

## 12.6 - The View Menu

The View menu sets how the session is seen, and what's visible or not.

Maximise Editor Space	Puts the Editor window in full screen mode
Maximize Mixer Space	Puts the Mixer window in full screen mode
Primary Clock	
Focus On Clock	Sets the focus on the <i>main clock</i> , allowing to type in numbers directly to change the
Timecode	Sets the main clock in timecode mode, so it displays time in the Hours:Minutes:Seconds
Bars & Beats	Sets the main clock in musical time mode, so it displays time in the Bars:Beats:Ticks
Minutes & Seconds	Sets the main clock in absolute time mode, so it displays time in the Hours:Minutes:Seconds
Samples	Sets the main clock in samples time mode, so the time is displayed in samples from

Table 5 – continued from previous page

Secondary Clock	
Timecode	Same as for the main clock (see above)
Bars & Beats	Same as for the main clock
Minutes & Seconds	Same as for the main clock
Samples	Same as for the main clock
Zoom	
Zoom In	Zooms in, focusing the <i>Zoom Focus</i> (see below)
Zoom Out	Zooms out
Zoom to Session	Adjust the zoom value so that all the session (as defined by its start and end markers) fits in the window
Zoom to Selection	Adjust the zoom value so that all the selected regions fit in the window
Fit Selection (Vertical)	Fits the selected track(s) in the window. If too many tracks are selected, they'll be re-scaled to fit.
Toggle Zoom State	Reverts to last zoom state (kind of “undo” for zoom, even if edits have been made in the meantime)
Expand Track Height	Increases the height of the selected tracks. If no track is selected, then all the tracks are expanded.
Shrink Track Height	Same as above, but reduces the height of the tracks
Zoom Focus	
Zoom Focus Left	Sets the screen's left side as the zoom target, i.e. when zooming in, the left side of the screen is the center of the zoomed-in area
Zoom Focus Right	Same, with the right of the screen
Zoom Focus Center	Same, with the center of the screen
Zoom Focus Playhead	Sets the playhead as the focus point of the zoom, i.e. the point in time that will stay at the center of the zoomed-in area
Zoom Focus Mouse	Same as above, with the mouse pointer
Zoom Focus Edit Point	Same as above, with the Edit Point
Next Zoom Focus	Circles between the previous modes
Rulers	
<input type="checkbox"/> Min:Sec	Shows (when checked) or hides a line in the <i>Ruler</i> with the time formatted as Hours:Minutes:Seconds
<input type="checkbox"/> Timecode	Same as above, with the time formatted as Hours:Minutes:Seconds:Frames
<input type="checkbox"/> Samples	Same as the above, with the time displayed in samples from the absolute start
<input type="checkbox"/> Bars & Beats	Same as the above, with the time formatted as Bars:Beats:Ticks
<input type="checkbox"/> Meter	Shows / hides the Meter line in the ruler, where the signature can be adjusted along with the tempo
<input type="checkbox"/> Tempo	Shows / hides the Tempo line, where the BPM can be changed with markers
<input type="checkbox"/> Ranges	Shows / hides the Range line, where ranges can be defined
<input type="checkbox"/> Loop/Punch	Shows / hides the Loop/Punch line, where loops and Punches can be defined
<input type="checkbox"/> CD Markers	Shows / hides the Range line, where CD Markers can be defined
<input type="checkbox"/> Markers	Shows / hides the Markers line, where custom markers can be defined
<input type="checkbox"/> Video	Shows / hides the Video timeline, where frames of the video are shown for syncing purposes
Video Monitor	
<input type="checkbox"/> Original Size	When the <i>Video Monitor</i> is active, resets its size to the original size, i.e. 1 pixel in the X and Y dimensions
<input type="checkbox"/> Letterbox	When checked, forces the ratio (width/height) to be the one of the original video. If the video is wider than the monitor, it will be scaled down to fit
<input type="checkbox"/> Always on Top	Stays above all other windows, enabling to work in Ardour without the video window obscuring other windows
<input type="checkbox"/> Fullscreen	Sets the Xjadeo window to be fullscreen. Can be useful in a dual monitor setup
<input type="checkbox"/> Timecode	When checked, displays a Timecode over the video, in the Hours:Minutes:Seconds:Frames format
<input type="checkbox"/> Frame number	When checked, shows the absolute frame number inside the video, i.e. this image is frame 123 of the sequence
<input type="checkbox"/> Timecode Background	Adds a black background to the timecode for readability
Scroll	
Scroll Tracks Down	Scrolls the view toward the bottom of the session from one screen (vertically, so along the Y axis)
Scroll Tracks Up	Same as above, towards the top
Scroll Forward	Scrolls the view toward the right of the session from one screen (horizontally, so along the X axis)
Scroll Backward	Same as above, to the left
Views	
Save View <i>n</i>	Saves the position on the timeline in the memory, horizontally and vertically (along the X and Y axes)
Go to View <i>n</i>	Loads and displays a saved position (see above)

Table 5 – continued from previous page

<input type="checkbox"/> Show Editor Mixer	When checked, the selected tracks' mixer strip is displayed on the left of the editor
<input type="checkbox"/> Show Editor List	In the Editor window, shows the <i>Editor List</i> , giving access to a number of handy lists
<input type="checkbox"/> Toggle Mixer List	In the Mixer view, shows the Mixer list, giving access to some handy lists ( <b>'Favorites'</b> )
<input type="checkbox"/> Toggle Monitor Section Visibility	If the Use monitoring section on this session has been checked in the <i>Session Properties</i> dialog, this option will toggle the visibility of the monitor section in the Editor
<input type="checkbox"/> Show Measure Lines	If checked, in the Editor, shows a vertical white line at each measure start
<input type="checkbox"/> Show Summary	If checked, in the Editor, shows the <i>Summary</i> , allowing a faster navigation in the session
<input type="checkbox"/> Show Group Tabs	If checked, makes the groups visible as tabs on the left in the Editor, and on the top in the Mixer
<input type="checkbox"/> Show Marker Lines	If checked, each marker is extended across all the tracks in the editor with a line of dots

## 12.7 - The Window Menu

The Window menu deals with the layout of the different windows, and their visibility.

<input type="checkbox"/> Audio/MIDI Setup	Shows the <i>Audio/MIDI Setup window</i> <#newopen-session-dialog> ___, where the sound system configuration can be modified
<input type="checkbox"/> Editor	
<input type="checkbox"/> Show	Switches to the Editor view
<input type="checkbox"/> Hide	Hides the Editor, hence showing the Mixer when the windows are attached
<input type="checkbox"/> Attach	If the Editor window is detached, separated from the main window, attach it back
<input type="checkbox"/> Detach	If the Editor is attached to the main window, detach it (makes the Editor a separated window, useful for multi-monitor setup)
<input type="checkbox"/> Mixer	
<input type="checkbox"/> Show/Hide/Attach	Sets <b>Detach</b> for the Editor, for the <i>Mixer</i> window
<input type="checkbox"/> Preferences	
<input type="checkbox"/> Show/Hide/Attach	Sets <b>Detach</b> for the Editor, for the <i>Preferences</i> window
<input type="checkbox"/> Meterbridge	Shows the <i>Meterbridge window</i> , that displays all the tracks' meter at once and their recording status, and is very handy for multitrack recording
<input type="checkbox"/> Scripting	Opens the <i>Lua Scripting window</i> , allowing to edit and run Lua scripts
<input type="checkbox"/> Tracks and Busses	Opens the Tracks and Busses window, which is a shortcut to many tracks/busses operations (routing, effects, ...)
<input type="checkbox"/> Locations	Opens the <i>Ranges and Marks window</i> , a single point of control for all range and location markers
<input type="checkbox"/> Binding Editor	Opens the <i>Key Bindings window</i> , which allows for easy creation or modification of any keyboard shortcut
<input type="checkbox"/> Bundle Manager	Opens the Bundle Manager window, allowing to create and manage <i>Bundles</i> , which are a way to simplify connection management, by defining groups of ports
<input type="checkbox"/> Big Clock	Opens the <i>Main Clock</i> as its own separate (and huge) window, which is helpful when recording
<input type="checkbox"/> Video Monitor	If a <i>video</i> has been imported in the session, opens a video window (namely, <i>Xjadeo</i> ), synced to the timeline
<input type="checkbox"/> Midi Tracer	Opens the MIDI Tracer window, allowing to follow each and every MIDI message entering or leaving Ardour
<input type="checkbox"/> Audio Connections	Opens the <i>Audio Connection Manager window</i> , a way to make connections to, from and within Ardour's mixer
<input type="checkbox"/> MIDI Connections	Same as above, for the MIDI connections
<input type="checkbox"/> Log	Shows the Log window, where Ardour lists useful information, warnings and errors

## 12.8 - The Help Menu

The Help Menu gives access to useful information about Ardour.

<input type="checkbox"/> About	Shows the About Ardour window, which contains information about the version, config, authors, and license of Ardour
Chat	This is a shortcut to the webchat version of the Freenode IRC channel of Ardour, where the developers meet, and questions can be asked if the Manual is not enough
Manual	Link to a FLOSSManual guide to Ardour
Reference	Link to this manual, hosted on ardour.org
User Forums	Link to ardour.org's user forum
How to Report a Bug	Link to an helping page about reporting bugs
Report a Bug	Link to Ardour's Mantis bugtracker
Ardour Website	Link to Ardour's main and official website
Ardour Development	Link to the developers' part of the official website

## 13 - Status Bar

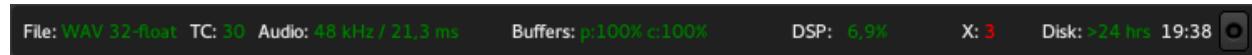


Fig. 15: The Status Bar

The status bar is an informative bar at the top of the window, showing:

File:	the file format used in the session, including when recording
TC:	is the timecode, i.e. the number of frames per second used by the session (for videos)
Audio:	gives the sample rate used in the session, and the latency computed from the buffer size
Buffers:	describe how much data is buffered, see below
DSP:	for Digital Sound Processing, shows how much of the CPU is used by Ardour and its plugins
PkBld:	(only shows up while creating peaks) displays the number of peak files left to create
X:	shows the number of xruns since Ardour's launch, see below
Disk:	reports the remaining hard disk space as the time that can be recorded with the current session setting
Wall Clock	showing the system time (especially useful in full screen mode)
Log button	that indicates if Ardour has encountered any warning or error.

Right clicking anywhere on the Status Bar allows to choose which of this information we want displayed, through a checkbox menu.

The buffers are labelled as p for playback and c for capture. If the system is fast enough, these buffers should be 100% full at all times, showing the system has time to precompute all the data before delivering it to the audio system. A buffer constantly under 20% is a sign of an underpowered computer system or of too much processing.

An Xrun (short for buffer over- or under-run) happens when the system has been forced to skip audio frames, e.g. if the latency asked is too short for the computing power of the machine. It usually results in clicks, pops and crackles if it happens while recording.

The log button turns yellow when a warning is shown, and red when an error occurs. Clicking the log button gives access to the log.

## 14 - Transport Bar

### The transport controls

The Transport Bar groups all the actions regarding the control of playback and recording.



Fig. 16: The transport controls

This bar is made of (from left to right):

Midi Panic	Immediately stops all midi output.
Enable/disable Audio Click	Toggles (on/off) a click track (metronome) along the <i>tempo</i> . Right clicking brings up the Click submenu from the Preferences. Scrolling with the mouse wheel adjusts the gain of the click.
Go to Start of the Session	Jumps back at the beginning of the session, as defined by the <i>start marker</i> .
Go to End of the Session	Jumps forward to the end of the session, as defined by the <i>end marker</i> .
Play Loop Range	Repeats the defined <i>loop</i> as defined by the <i>Loop range</i> , until the “Stop playback” button is pressed. Clicking the “Play loop Range” button while already active switches to normal Play mode, which exits the loop without stopping and restarting playback.
Play Range/Select MIDI	If a range has been defined using the Range Mode button, plays the range, or if an audio or MIDI region is selected, plays this region. In both cases, the playback stops at the end of the range or selected region.
Play from playhead	Starts the playback and optionally record (more below).
Stop	Whatever the playing mode (loop, range, ...) stops all playback. Depending on other settings, some effects (like chorus or reverb) might still be audible for a while.
Toggle Record	Global switch button to activate/deactivate recording. While active, the button blinks red. The button doesn’t start recording by itself: if one or more tracks are marked as record-enabled, pressing the “Play from Playhead” starts recording on those tracks.

All these actions are bound to keyboard shortcuts, which allows for speedier use and more focused work.

If Ardour is synchronized with other devices then some or all of these control methods may be unavailable—depending on the synchronization protocol, Ardour may respond only to commands sent from its master device(s).

Under these buttons is the Shuttle Speed Control that allows to scrub through the audio quickly. The slider decides the playback speed: the further from the center it is set, the faster the playback will scrub in both directions.

The Shuttle Speed Control supports 2 operating modes, that can be chosen with right click > Mode:

- Sprung mode that allows for a temporary scrub: it only scrubs while the mouse is left clicked on the control.
- Wheel mode that allows to set a playback speed until the “Stop” button is pressed, which stops the playback and resets its speed.

On the left of the slider is the positional sync button (which might show Internal, or MTC or several other values), than can be used to control whether or not the transport position and start is controlled by Ardour, or by an external positional synchronization source, such as MIDI Time Code (MTC), Linear Time Code (LTC) or JACK (see *Timecode Generators and Slaves*).

The current playback status (Stop, Play, or speed %) is shown on the right of the speed slider.

### Using Key Bindings

Ardour has many available commands for playback control that can be bound to keys. Many of them have default bindings, some do not, so the list below shows both the default bindings and internal command names for some of them.

Space	Switch between playback and stop.
Home	Move playhead to session start marker
End	Move playhead to session end marker
→	Playhead to next region boundary
←	Playhead to previous region boundary
0	Move playhead to start of the timeline

Go to the Transport and Transport > Playhead to find more.

### 15 - Transport Clocks



Fig. 17: The transport clocks in Ardour

Clocks in Ardour are used to display time values precisely. In many cases, they are also one way to edit (change) time values, and in a few cases, the only way. All clocks share the same basic appearance and functionality, which is described below, but a few clocks serve particularly important roles.

In the transport bar of the editor window there are two clocks (on a large enough screen), that display the current position of the playhead and additional information related to transport control and the timeline.

These are called the transport clocks; the left one is the primary transport clock and the right one is the secondary transport clock.

All the clocks in Ardour share the same powerful way of editing time. Refer to [Editing Clocks](#) to learn how. Editing the time in the transport clocks will reposition the playhead in the same way that various other editing operations will.

### The Special Role of the Secondary Transport Clock

On a few occasions Ardour needs to display time values to the user, but there is no obvious way to specify what units to use. The most common case is the big cursor that appears when dragging regions. For this and other similar cases, Ardour will display time using the same units as the secondary clock.

#### Why are there two transport clocks?

Having two transport clocks allows seeing the playhead position in two different time units without having to change any settings. For example, one can see the playhead position in both timecode units and BBT time.

### Special Modes for the Transport Clocks

In addition to the time-unit modes, each of the two transport clocks (again, on a sufficiently large screen) can be independently set to display Delta to Edit Point in whatever time units its current mode indicates. This setting means that the clock shows the distance between the playhead and the current edit point, and it may show a positive or negative value depending on the temporal order of these two points. The clocks will use a different color when in this mode to avoid confusion.

To switch either (or both!) of the transport clocks into this mode, use *Edit > Preferences > Transport* and select the relevant checkboxes.

Note that when in Delta to Edit Point mode, the transport clocks cannot be edited.

### The Big Clock

To show the current playhead position in a big, resizable window, activate *Window > Big Clock*. The big clock is very useful when working away from the screen but still wanting to see the playhead position clearly (such as when working with a remote control device across a room). The big clock will change its visual appearance to indicate when active recording is taking place. Below on the left is a screenshot showing a fairly large big clock window filling a good part of the display, and on the right, the same clock during active recording.

## 16 - Selection and Punch Clocks

### The Selection Clocks

The current selection range, as set with the *Range Mode tool*, is displayed in these three clocks: start of the range, end of the range, and length.

Clicking on the range clocks will locate to either the beginning or end of the punch range.

Right clicking on any of the clocks brings up a context menu allowing to change the type of time display between the *4 clock modes*, and to copy the selected clock's time to the clipboard.



Fig. 18: The range clocks

### The Punch Controls & clocks



Fig. 19: The Punch Controls, and the related Punch clocks

The punch controls available in the main toolbar, work in conjunction with the punch clocks, only visible while in Editor Mode.

The In and Out buttons relate to the Punch range, and allow to use only one of the two punch boundaries, or both:

In only	Records from the In marker on, without a end boundary
Out only	Records until the Out marker, without a beginning boundary
In <i>and</i> Out	Records only between the In and Out markers

The punch clocks can be controlled the same way as the range clocks (moving the playhead, and changing the display mode).

### Recording mode

The Rec button affects how the tracks behave when recording:

Non-Layered OFF ( <i>default</i> )	Tracks in normal mode will record non-destructively — new data is written to new files, and when overdubbing, new regions will be layered on top of existing ones. This is the recommended mode for most workflows.
Non-Layered ON	Tracks using non-layered mode will record non-destructively — new data is written to new files, but when overdubbing, the existing regions are trimmed so that there are no overlaps. This does not affect the previously recorded audio data, and trimmed regions can be expanded again at will. Non-layered mode can be very useful for spoken word material, especially in combination with <i>push/pull trimming</i> .

See *Track Modes* for more information.



Fig. 20: The Mini-Timeline.

## 17 - Mini-Timeline

The mini-timeline allows, as the *Summary* does, navigation of a session. Its main advantage, though, is that it stays visible even when in Mixer mode.

The range of time covered by the mini-timeline is set by right clicking the timeline, and choosing a time span from 30 seconds up to 20 minutes.

The mini-timeline also shows all markers (start, end and any user defined ones). Clicking a marker jumps to that point on the timeline, allowing for quick access to key timings in the session.

While hovering with the mouse over the mini-timeline:

- left clicking moves the playhead to the time under the mouse cursor
- using the scroll wheel scrolls the playhead back and forth inside the session
- using scroll wheel scrolls more finely inside the session
- using scroll wheel scrolls even more finely inside the session

## 18 - Other Toolbar Items

### The Monitor Options

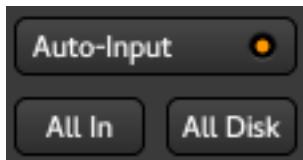


Fig. 21: The Monitor options

These buttons allow switching the monitoring mode globally, for all the tracks at once. The monitoring mode allows to decide what the user wants to be listening to, between:

- All In: all the tracks play what's on their *Inputs*.
- All Disk: all the tracks play the actual content of the playlist on *Disk*.

The Auto Input switch allows Ardour to auto-select what is played, which is:

- When not playing: all tracks are on In (to listen to any connected source)
- When playing, all tracks are on Disk (to play whatever was recorded on those tracks)
- When recording, on rec-enabled tracks: In and on non rec-enabled ones: Disk

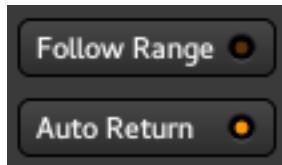


Fig. 22: The playhead options

### The Playhead Options

Those 2 buttons control the behaviour of the playhead:

- Follow Range is a toggle that can be used to control whether or not making a range selection will move the playhead to the start of the range.
- Auto Return is a toggle switch too. When active, pressing the Stop button returns the playhead to its previous position, and when inactive, pressing Stop keeps the playhead at its current location. Activating Auto Return can be useful for hearing the same piece of audio before and after tweaking it, without having to set a loop range on it.

### The Status indicators

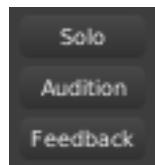


Fig. 23: The Status buttons

The Status buttons show the current session state:

Solo	Blinks when one or more tracks are being soloed, see <i>Muting and Soloing</i> . Clicking this button disables any active explicit and implicit solo on all tracks and busses. Clicking this button deactivates the solo on every track/bus.
Audition	Blinks when some audio is auditioned, e.g. by using the import dialog, or using the Audition context menu in the <i>Regions List</i> . Clicking this button stops the auditioning.
Feedback	Blinks when Ardour detects a feedback loop, which happens when the output of an audio signal chain is plugged back to its input. This is probably not wanted and can be dangerous for the hardware and the listener.

### The Mode Selector



Fig. 24: The Mode Selector

The Mode Selector allows switching between the Editor and Mixer windows. If a window is detached, the corresponding button is lit in blue. Clicking the button switches the detached window visibility.

The global meter shows the levels of the master's output. Its the same meter that sits in the *Master's Mixer strip*, and also shows a peak indicator, that turns red when any level exceeds 0dB. It can be reset by a Left click.

The buttons in between the Mode Selector and the global meter are script buttons, which are user-definable buttons to attach any session *lua-script* to.

## 19 - Toolbox

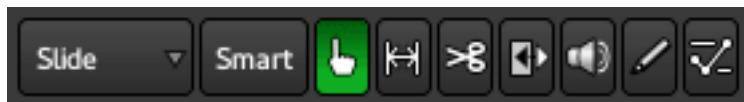


Fig. 25: Editor toolbar's tools, aka toolbox

### Global Edit mode

Ardour has a global edit mode selector at the left of the Editing toolbar, which affect how regions are moved or copied:

Slide	Regions move freely. Ardour creates overlaps when necessary.
Ripple	Editing affects the regions to the "right" of the edit (see below).
Lock	No region motion is permitted (except for "nudge").

Ripple Edit mode provides the following conveniences:

- Deleting a range will move later regions to compensate for the deleted time
- Deleting a region will move later regions to compensate for the deleted region's length
- Moving a region will move later regions to compensate for the length of the move
- Inserting a new region (via dragging or via Paste) will move later regions to the right to compensate

If Snap To Grid is enabled, then regions can only move so that they align with locations determined by the current snap settings (beats, or seconds, or other region boundaries, etc). See *Snap To the Grid* for details.

### The *Smart* switch

The Smart Mode button to the left of the mouse mode buttons modifies the Grab Mode. When enabled, the mouse behaves as if it is in "Range Mode" in the upper half of a region, and in "Grab Mode" in the lower half. This allows avoiding constant switching between these two modes.

**Mouse Modes**

Grab Mode	The Grab Mode is used for selecting, moving, deleting and copying objects. When in object mode, the mouse pointer appears as a hand whenever it is over the track canvas or the rulers. The mouse can now be used to select and perform operations on objects such as regions, markers etc. This is the most common mode to work in, as it allows you to select and move regions, as well as modify automation points on the automation tracks.
Range Mode	When in Range Mode, the mouse pointer appears as a vertical line whenever it is over the track canvas or the rulers. The mouse will now be able to select a point or range of time. Time ranges can be selected over one or several tracks, depending on the selection of your tracks. If none of your tracks are selected, the Range Tool will operate on all the session track visualized in the Editor. If you want to edit only particular tracks, select them before you apply the range tool.
Cut Tool Mode	When in Cut Tool Mode, the mouse pointer appears as a pair of scissors whenever it is over the track canvas or the rulers. This tools allows to cut any region into 2 regions at the mouse cursor, regardless of the Edit Point. If one or more track(s) is selected, then all the regions on these tracks will be split at the mouse cursor position. If no track is selected, then only the region hovered by the mouse cursor will be split.
Stretch Mode	When in time fx mode, the mouse pointer appears as a distinctive expanding square symbol whenever it is over the track canvas or the rulers. This mode is used to resize regions using a timestretch algorithm. Click on an edge of a region of audio and drag it one way or the other to stretch or shrink the region.
Audition Tool	Clicking a region using the audition tool will play this region to the control room outputs. You can also scrub with this tool by clicking and dragging in the direction you wish to listen. The amount you drag in one direction or the other will determine the playback speed.
Draw Tool	When in Draw Tool mode, the mouse pointer will change to a pencil. You can then click within an audio region to change the gain envelope for that region. This curve is separate from fader automation for individual tracks. It will remain locked to the region's time, so if the region is moved, the region gain envelope is moved along with it. The draw tool works on automation too, allowing the creation and modification of control points on the automation curves. Last, it is used on a MIDI region to edit the notes.
Internal/Regional Edit Mode	When in Internal Edit mode, the mouse pointer will change to cross-hairs. This tool acts on region gain and automation as the Draw tool. On a MIDI region, it allows to lasso-select multiple notes at a time.

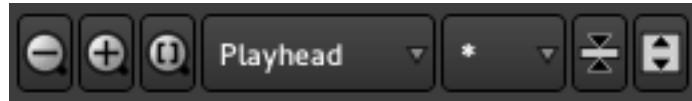
**20 - Controls****20.1 - Zoom Controls**

Fig. 26: Editor toolbar's zoom

The zoom controls allow to navigate the session along both the time and track axes.

The drop down Zoom Focus menu allows to select a focus point for the zoom, i.e. the center of the zoom. The choices are:

- Left of the screen

- Right of the screen
- Center of the screen
- Playhead
- Mouse
- Edit Point as set in the *Edit point* control.

The two leftmost zoom buttons (– and +) use this zoom focus to zoom out and in respectively.

The Zoom to session button is a handy shortcut to zoom out or in until all the session (as defined by its *start/end markers*) fits horizontally.

Changing the Number of visible tracks dropdown menu allows to fit this number of tracks vertically in the screen.

There *is* a minimal track height to keep it visible, so according to your screen vertical size, some high number can have no effect.

Inside this menu are two handy choices:

- Selected tracks that focus on the selected tracks. If the selected tracks are not contiguous, the unselected tracks inbetween will be hidden, see the *Track and Bus list*.
- All that fits all the tracks of the sessions vertically (provided there's enough screen estate).

The rightmost buttons Shrink tracks and Expand tracks reduce or expand the vertical size of the selected tracks. If no track is selected, all the tracks will be shrunk or expanded each time the button is pushed.

## 20.2 - Grid Controls

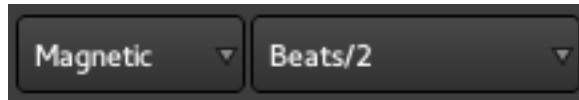


Fig. 27: Editor toolbar's grid.

Ardour's editor utilizes a grid to assist in the placement of regions on the timeline, or with editing functions that need to happen at a specific point in time. This snapping of the cursor and various objects to the grid can be toggled on or off, as does its behaviour, and grid units.

### About Snapping

There are two ways to think about aligning material to a grid. The first and most obvious one is where an object's position is clamped to grid lines. In Ardour, this is called absolute snap and is commonly used when working with sampled material where audio begins exactly at the beginning of a file, note or region.

The second, relative snap, is used when an object's position relative to the grid lines is important. In music, this allows to move objects around without changing the “feel” (or timing) of a performance.

Absolute snap is the default method of snapping in Ardour.

While dragging objects, pressing the absolute snap modifier key(s) switches from absolute to relative snap.

The snap can also be entirely disabled by using the snap modifier (see below).

Note that in relative snap mode the reference point is taken to be the distance to the nearest grid line.

Note also that when an object lies exactly on a grid line, there will be no difference between relative and absolute snap modes.

The relative snap and snap modifiers (along with other modifier keys) may be set in Edit > Preferences > User Interaction

For common use patterns, it is recommended to assign a unique key for one snap modifier and two keys for the other in such a way that they share an otherwise unused key. For example, the snap modifier may be chosen to be the key and the relative snap modifier to be the and keys.

### Snap Modes

Using the above modifications, Ardour supports three different modes of snapping to the grid:

No Grid	disables the grid. All objects move freely in this mode. In No Grid mode, the grid may be temporarily activated by pressing the snap modifier (for absolute snap) or switch to relative snap by pressing the relative snap modifier.
Grid	activates normal snapping. All positions of objects snap to the grid. (See <i>Grid Units</i> below to change the grid). Moving an object in “Grid”-mode, does not change its position until the mouse is far enough for the object to reach the next grid line. To maintain an objects’ position relative to the grid line, the “snap relative” modifier can be used. When holding down this modifier during a drag, the dragged object will jump while maintaining its original distance from the line. New objects will always be created at grid points. Holding down the snap modifier will disable the current grid setting and allow moving the object freely.
Magnetic	is a less strict type of snapping. Objects can still be moved to any position, but positions close to the relative or absolute grid points will snap. In order to move an object very close to a snap point, it may be necessary to zoom in to prevent snapping to that point, or to use the snap modifier to disable snap completely. As with Grid mode, the snap modifier will disable snap completely while the absolute snap modifier will move the “notch” of Magnetic snap to the grid lines.

### Syncing Regions to the Grid

By default, a region’s beginning will be used as the reference for both types of snapping, this behaviour can be changed by setting a sync point in the region, by selecting the region(s) and pressing V. This will set the sync point to the current *edit point*.

### Grid Units

The selector next to the grid mode selector defines the size of the grid elements. The grid can be set to several different units:

CD Frames	A CD Frame is 1/75th of a second. Snapping to CD Frames (using absolute snap) can be used to avoid issues with CD track lengths.
Timecode Frames/Seconds/Minutes	The duration of a frame depends on the timecode settings for the session.
Seconds/Minutes	These are absolute time units, unaffected by sample rate or timecode settings
Beats/N	Set the grid to units of 1/N beats, where N can be 128, 64, 32, 16, 8, 7, 6, 5, 4, 3, 2. The duration of a grid unit will depend on the tempo and meter in effect at that point in the timeline.
Beats	Set the grid to whole beats. The duration of a grid unit will depend on the tempo and meter in effect at that point in the timeline.
Bars	Set the grid to whole bars. The duration of a grid unit will depend on the tempo and meter in effect at that point in the timeline.
Markers	The grid lines are the markers.
Region Starts	The grid lines are constructed from region start points (see below).
Region Ends	The grid lines are constructed from region end points (see below).
Region Syncs	The grid lines are constructed from region sync points.
Region Bounds	The grid lines are constructed from region start or end points.

To use Region starts/ends/syncs/bounds as snap choices, it is necessary to have either:

- *No* tracks selected, which means that Ardour snaps to regions on any track, or
- Several tracks selected, which means that Ardour only snaps to regions on those selected tracks.

If items are moved on a track, and only the current track is selected, then snapping will only happen with other regions on the same track. This means that enabling Edit > Preferences > Editor > Link Selections of Regions and Tracks will make the “Region” grid unit unusable. This option should not be used in conjunction with the use any of the Region grid units.

### 20.3 - Edit Point Control

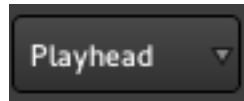


Fig. 28: Editor toolbar’s Edit Point

Editing operations in a Digital Audio Workstation like Ardour can be broken down according to how many points on the timeline are required to carry the operation out. Splitting a region for example, requires just one position on the timeline (the one where the split will happen). Cutting out a time range requires two positions, one for the start of the cut and one for the end.

In Ardour the edit point is the location where most single-point editing operations take place. It can be set to either of the following:

- the Playhead position
- the selected (or “active”) Marker
- the position of the Mouse (or touch) pointer

The default edit point is the location of the pointer.

There are two keybindings available to cycle through the edit point options. The most common workflow tends to involve switching back and forth between the playhead and mouse as the edit point. Pressing

the grave accent key ‘ switches between these two. Using ‘ cycles through all three choices (including the selected marker). The edit point can also be switched using the combo-selector just right of the snap/grid unit selector.

## 20.4 - Nudge Controls



Fig. 29: Editor toolbar's Nudge

The nudge controls will move the selected region(s) by a fixed amount of time. The left and right buttons move either backward or forward in time, and the small clock to the left of these buttons sets the amount of time to nudge by. As with all other clocks, you can right-click on the clock to choose the time representation you want to use.

If there are no selected objects, the nudge controls can be used to move the playhead backward or forward by the amount shown on the clock.

## 21 - Ruler



Fig. 30: Ardour's Ruler

The Ruler scales the session along time, allows navigating, and can be marked for different uses.

One of its main uses is to move the playhead: clicking anywhere on the timeline will bring the playhead at this location in time. Also, using the mouse's scrollwheel while hovering the Ruler will zoom in or out ( / ).

The Ruler is made of a succession of rows, each having a special role related to time. Adding or removing rows can be done by right clicking anywhere in the ruler's header on the left, and ticking any of:

Mins:Secs	scaling the session with the Mins:Secs:mSec notation
Timecode	scaling the session with the Hours:Mins:Secs:Frames notation
Samples	scaling the session with the sample number notation
Bars:Beats	slicing the time according to the time signature of the Meter
Meter	shows the time signature. It can be changed along the timeline, by Right click > New Meter. The Bars:Beats ruler will reflect the change.
Tempo	shows the BPM. It can be changed along the timeline, by Right click > New Tempo. The Bars:Beats ruler will reflect the change.
Range Markers	allow to create and modify ranges directly on the Ruler.
Loop/Punch Ranges	are special kind of ranges designed to be played as a loop and to do punch recording, i.e. recording on a precise section of time, respectively.
CD Markers	are markers designed to be used while creating a recording that has to be split in time, as an audio CD
Location Markers	is meant to receive any kind of marker, user generated or from Ardour itself.
Video Timeline	shows thumbnails of the <i>video</i> in the timeline

Most of the operations on the markers are described in [Working with Markers](#), while the Meter, Tempo, Bars:Beats and Timecode use are described in [Tempo and Meter](#).

## 22 - Summary



Fig. 31: Ardour's Summary

The Summary is a global overview of the session, allowing for a good “bird’s eye” view of where in time and tracks the work happens.

Each horizontal line represents a track in the session, with the colored bars being the audio and MIDI regions, colored as per their track’s color setting.

Two yellow vertical lines show the position of the *Start* and *End* markers, defining the session’s length. The red line shows the playhead’s position.

The transparent white rectangle represents what’s actually displayed in the Editor window, i.e. what part of the session is being looked at on screen.

The Summary also doubles as a navigator:

- the arrow on the left allow to scroll the view horizontally to the left, by 1 length of the view each time
- the arrow on the right allow to scroll the view horizontally to the right, by 1 length of the view each time
- the white rectangle can be dragged anywhere on the session, moving the view accordingly
- the right and left borders of the white square can be resized, zooming in and out accordingly

## 23 - Editor Lists

At the right hand side of the editor window is an optional area which provides one of a range of useful lists of parts of the session. It is not shown by default when first starting using Ardour. The Editor list can be hidden or shown using View > Show Editor List. The very right-hand side of the list gives a selection of tabs which are used to choose the list to view. The left-hand border of the list can be dragged to vary the width of the list.

### 23.1 - The Region List

The region list shows all the regions in the session. The left-hand column gives the region name, and there are a range of times given for information:

Position	position of the start of the region on the global timeline
End	position of the region on the global timeline
Length	duration of the region
Sync	position of the sync point, relative to the start of region (can be negative)
Fade In	duration of the fade in. Can't be less than 1 ms, to avoid clipping.
Fade Out	duration of the fade out (positive value, 1 ms).

The secondary clock defines the time unit used in this list. *Set the secondary clock* to the type of unit desired.

At the right of the list are four columns of flags that can be altered:

L	whether the region position is locked, so that it cannot be moved.
G	whether the region's position is 'glued' to bars and beats. If so, the region will stay at the same position in bars and beats even if the tempo and/or time signature change.
M	whether the region is muted, so that it will not be heard.
O	whether the region is opaque; opaque regions 'block' regions below them from being heard, whereas 'transparent' regions have their contents mixed with whatever is underneath.

Hovering the mouse pointer over a column heading shows a tool-tip which can be handy to remember what the columns are for.

A handy feature of the region list is that its regions can be dragged and dropped into a suitable track in the session.

### 23.2 - The Tracks and Busses List

This lists the tracks and busses that are present in the session. The list order reflects the order in the editor, and track or bus names can be dragged-and-dropped in the editor list to re-order them in the editor. The columns in the list represent the following:

V	whether the track or bus is visible; they can be hidden, in which case they will still play, but just not be visible in the editor; this can be useful for keeping the display uncluttered.
A	whether the track or bus is active; inactive tracks will not play, and will not consume any CPU.
I	for MIDI tracks, whether the MIDI input is enabled; this dictates whether MIDI data from the track's input ports will be passed through the track.
R	whether the track is record-enabled.
RS	whether the track is record safe; a record safe track cannot be armed for recording, to protect against a mistake.
M	whether the track is muted.
S	track solo state.
SI	track solo-isolated state.
SS	solo safe state.

Each icon in these columns can be clicked to toggle the track/bus state, which is a very fast way to set multiple tracks/busses state at once.

As with the region list, hovering the mouse pointer over a column heading shows a tool-tip which can be handy to remember what the columns are for.

### 23.3 - The Snapshot List

This list gives the snapshots that exist of this session. Clicking on a snapshot name will load that snapshot. See [Snapshots](#) for more information on snapshots.

### 23.4 - The Track and Bus Group List

This shows the track/bus groups that exist in the session. These groups allow related tracks to share various properties (such as mute or record enable state). For full details, see the section called *Track and Bus Groups*.

The columns in this list are as follows:

Col	the colour that the group uses for its tab in the editor.
Name	the group name.
V	whether the tracks and busses in the group are visible.
On	whether the group is enabled.
G	ticked if the constituents of the group are sharing gain settings.
Rel	ticked if shared gains are relative.
M	ticked if the constituents share mute status.
S	ticked if the constituents share solo status.
Rec	ticked if the constituents share record-enable status.
Mon	whether the constituents share monitor settings.
Sel	whether the constituents are selected together.
A	whether the constituents share active status.

### 23.5 - The Ranges and Marks Lists

The Ranges & Marks List is a tab in the Editor Lists area on the right of the Editor window. If the editor list area isn't visible it can be enabled by checking View > Show Editor List. The Ranges & Marks list can be used as a single point of control for all range and location markers (including the punch and loop ranges), or as a supplement to other methods of working with them.

## Common elements

Each section has a set of editable *clock widgets* which display the location of a marker, or the start, end, and duration times of a range, respectively.

The Use PH buttons allows to set the corresponding clock to the current playhead position. A Middle click on any of the clocks will move the playhead to that location. Both functions are also available from the clock context menus.

Right clicking on any of the clocks brings up a context menu that allows changing of the display between Timecode, Bars:Beats, Minutes:Seconds, and Samples.

The — (subtract) button in front of each user-defined range or marker in the list allows that particular item to be removed. The name fields of custom ranges and markers can be edited.

The Hide checkboxes make markers and ranges invisible on the respective ruler to reduce visual clutter; the markers remain active however, and can be used normally.

Selecting Lock prevents the respective marker from being moved until unlocked. Where applicable, Glue fixes the marker position relative to the current musical position expressed in bars and beats, rather than the absolute time. This will make the respective marker follow changes in the tempo map.

At the bottom of the list are buttons to add new markers or ranges.

## List sections

Loop/Punch Ranges	This list shows the current loop and punch range settings. Since these are built-in ranges, they cannot be renamed or removed.
Markers (Including CD Index)	This section lists the session's markers. By ticking CD, Ardour is instructed to create a CD track index from this marker, which will be included in the TOC or CUE file when exporting.
Ranges (Including CD Track Ranges)	This is the list of ranges (including CD track ranges). Ticking CD will convert the range to a CD track, which will again be included in exported TOC or CUE files. This is relevant for Disk-At-Once recordings that may contain audio data between tracks.

## 24 - Favorite Plugins Window

The Favorite Plugins window is on the top-left side of the Mixer Window. Like other elements in that window it has variable height and can be hidden by dragging it to zero-height. If it is not visible, the top-handle can be grabbed and dragged down to reveal it.

Plugin names that have a right facing triangle next to them have presets associated with them; clicking on the triangle will cause all presets associated with the plugin to show in the list.

## Features

The Favorite Plugins window provides easy access to frequently used plugins:

- Plugins can be dragged from the window to any track or bus *processor box*, which will add the plugin to that track or bus at the given position.
- The list includes user-presets for the plugins. Dragging a preset to a given track or bus will load that preset after adding the plugin.

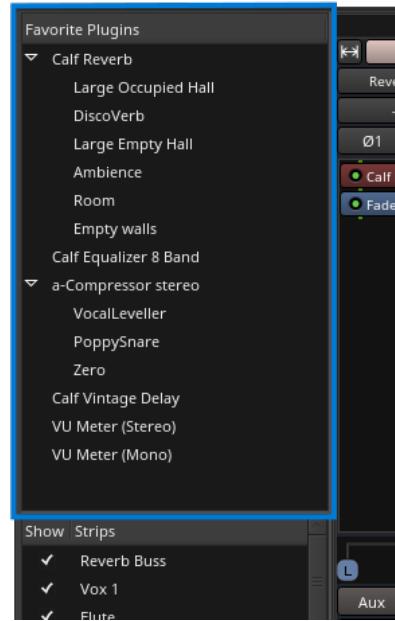


Fig. 32: The Favorite Plugins window.

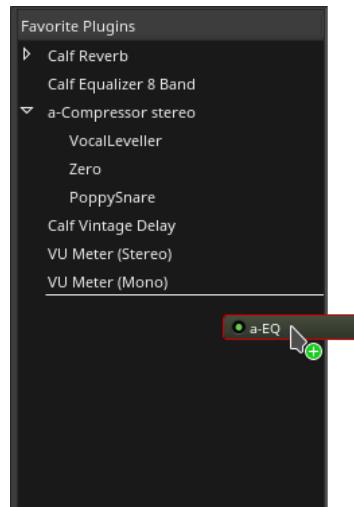


Fig. 33: Dragging a plugin to the window.

- Double-clicking on a plugin or preset adds the given plugin to all selected tracks/busses pre-fader. Other insert positions are available from the context menu (right click).
- Dragging a plugin from a track into the window will add it to the list and optionally create a new preset from the current settings. The horizontal line in the list shows the spot where the plugin will land.
- The context-menu allows the deletion of presets or removal of the plugin from the list.
- Plugins in the list can be re-ordered using drag & drop. The custom order is saved.

When favorites are added with the *Plugin Manager*, they are appended to the bottom of the list.

## 25 - Strips list

The Strips List is a quick way to manage big sessions, with lots of tracks, where the mixer would otherwise be too crowded.

It is a list of all the tracks, busses and VCA in the session, with a tick to allow for hiding or showing them. This visibility status also affects the Editor view, and is exactly the same as toggling the V checkbox in the *Tracks and Busses* panel of the Editor List.

Dragging and dropping tracks inside the Strips List allows to reorganise the tracks in the session, both in the Mixer and the Editor. Clicking a track scrolls the Mixer to show this track.

It is possible, by right clicking, to act on multiple tracks at once:

- Show All
- Hide All
- Show All Audio Tracks
- Hide All Audio Tracks
- Show All Audio Busses
- Hide All Audio Busses
- Show All MIDI Tracks
- Hide All MIDI Tracks

The + button under the list is a shortcut to create a new track, bus or VCA, as in clicking Track > Add Track, Bus or VCA....

## 26 - Groups list

The Groups List allows to quickly manage the *groups* of the session, and make use of them.

Each group has a Show checkbox to quickly toggle their visibility. Clicking an already selected group allows to rename it.

The context menu, reached by right clicking a group, allows for multiple mixing actions:

Create New Group From...	Creates a new group based on some track properties. The choice is: <ul style="list-style-type: none"> <li>• Selection... to create a group of all selected tracks</li> <li>• Record Enabled... to create a group of all the tracks that are record enabled</li> <li>• Soloed... to create a group of all the soloed tracks</li> </ul>
Create New Group with Master From...	Acts exactly as the previous choice, but also creates a Control Master tied to these tracks.
Assign Selection to Control Master...	Allows to link all the selected tracks to a chosen Control Master, whether or not they belong to a group.
Assign Record Enabled to Control Master...	Allows to link all the record armed tracks to a chosen Control Master.
Assign Soloed to Control Master...	Allows to link all the soloed tracks to a chosen Control Master.
Enable All Groups	Enable all the groups, i.e. their selected properties are synchronized.
Disable All Groups	Disable all the groups, i.e. changing a property in a track won't affect the others.

When a group is selected, right clicking it adds the following menu entries:

Create New Group with Master From...	Acts exactly as the previous choice, but also creates a Control Master tied to these tracks.
Edit Group...	Shows the Track/bus Group <i>window</i> .
Collect Group	Rearranges the tracks/busses order to visually group together the tracks belonging to the same group.
Remove Group	Deletes the group (but not the tracks/busses belonging to this group).
Assign Group to Control Master...	Allows to link all the tracks in the group to a chosen VCA.
Add/Remove Subgroup Bus	Creates/removes a new bus connected to the Master, and send the output of all the tracks in the group to this new bus.
Add New Aux Bus (pre/post-fader)	Creates a new bus connected to the Master, and create <i>Aux Sends</i> (pre or post-fader) in all the tracks in the group to this new bus.

The + button under the list allows the creation of an (empty) group, while the – button deletes the selected group (but not the tracks in this group).

## 27 - Mixer Strips

### 27.1 - Audio/MIDI Mixer Strips

A mixer strip in Ardour is a vertical view of the track, from a mixing point of view. This view is convenient to deal with I/O, effects, panning/muting, gain, etc... It has a general “top to bottom” flow.

The mixer strips breaks down into:

1. Header
2. Track name



3. Input(s)
4. Polarity *only for audio tracks*
5. Processor box
6. Panner
7. Recording options
8. Mute/Solo
9. Gain & Meter
10. Control master
11. Fader automation/mix group/metering point
12. Output(s)
13. Comments

## Headers

At the top of the window, is the *group tabs* (here, *recm...*). This allows to group tracks together for common controls.

Bellow are 3 buttons:

- The double arrow button allows to shrink/expand the width of the strip. Click the button will shrink/expand all the tracks at once
- The color bar shows the color of the track in the editor
- The X button toggles the visibility of the track OFF. To turn it back ON, one can either go to the *Tracks and Busses list* in the Editor view and check the “V” column on the track’s line or stay in the Mixer view and check the Show column of this strip in the *Strips list*.

Right clicking on the color bar will bring up a context menu, which is exactly the same as clicking on the Track name button.

## Track Name

Clicking the Track name button will bring up a menu:

Color...	Changes the strip/track color
Comments...	Shows an editor to put comments about the track, see below the Comments button
Inputs...	Shows the Routing grid for the inputs of the track
Outputs...	Shows the Routing grid for the outputs of the track
Save As Template...	Allows to save the track without its media content (I/O, effects,...) for later reuse
Rename...	Changes the name of the track (effective both in the Mixer and the Editor)
Active	Select the active status of the track. An inactive track won't output any sound
Strict I/O	While in <i>patchbayStrict I/O</i> mode, a track <i>always</i> has as many output as it has inputs, regardless of the effects. When disabled, a stereo effect put on a mono track will result in a stereo output for the strip.
Pin Connections...	Shows the Pin Configuration window, that shows (and allows to modify) all the signal flows inside the track
Adjust Latency...	Shows the Track Latency dialog, that allows fine-tune the latency to the track, in samples, msec or period
Protect Against Denormals	Uses a trick to get rid of <i>denormals</i> , which are very small numbers the CPU can have a hard time dealing with. To be used if the CPU consumption for plugins is noticeably higher than expected
Duplicate...	Copies the track to a new one, optionally with its playlist
Remove	Deletes the track and its playlist

## Inputs

The dropdown button shows the current input port(s), i.e. what's plugged to the "in" of the track. By default, each audio track is connected to the system inputs, ready for recording, as shown by the number(s). Clicking the dropdown Inputs button will allow to change the inputs, through a menu:

Disconnect	Disconnects everything, i.e. the track has no input
In <i>n</i>	Those are the system inputs, e.g. to record from the soundcard. A mono track will have <i>In 1</i> and <i>In 2</i> separated, while a stereo track can have <i>In 1+2</i>
<i>Track n output</i>	All the outputs of compatible tracks, e.g., a mono track can only receive a mono signal, a MIDI track can only receive MIDI signal, ...
Add Audio Port	Adds an audio input to the track, i.e. a mono audio track becomes a stereo one
Add MIDI Port	Adds a MIDI input to the track. Adding it to an audio track makes it a mixed Audio/MIDI track. This can be useful e.g. to feed some plugins with a MIDI signal to control the audio, like a vocoder
Routing Grid	Shows the <i>Routing Grid window</i> , which allows for more complex input configuration

The Routing Grid can also be shown by right clicking the dropdown Inputs button. It allows to make the connections through a matrix, and connect things that are not listed in the menu above, or connect to multiple sources at once, reduce the number of inputs, etc...

On audio tracks, is a Trim knob, as on traditional consoles. It set the base input level for the track, avoiding any clipping. Notice that it trims both any input, but (when playing back), also the level of the playlist as displayed in the Editor. It makes sense as while playing, the input of the track is the playlist, on which the mixer strip acts.

On midi tracks, it is replaced by a MIDI Input button, that allows/disallows MIDI input on the track.

### Polarity

On audio tracks only, the Polarity button(s), 1 per input, allow to reverse the signal, i.e. a negative value will be positive and vice-versa. This can help deal with phasing issues.

### Processor box

The processor box is where the effects are added. By default, one effect is always present: the Fader (see below). The effects can be added *pre-fader* and appear in brown, or *post-fader*, where they will appear in dark green. The signal flow is represented by lines, red for the MIDI and green for the audio.

It is also where the *Sends* come from, whether external or auxiliary.

To learn more about the processor box, see *The Processor Box*.

### Panner

The Panner visually displays how the sound will be distributed between the different outputs. They'll look and behave differently if the track is mono, stereo, or has multiple channels.

Right clicking the Panner will show a menu:

Bypass	When checked, the panner is grayed, and the signal is not affected by it
Reset	Resets the panner to its default settings, e.g. for a mono signal, it is centered
Edit...	Shows a Panner dialog, which allows for fine tuning of the panner

See *Panning* to learn more about how to control the panner, and what kind of panners are available inside Ardour.

### Recording options

The most noticeable button here is the Record Enable one, with a red circle. When enabled, next time the Global record will be armed and playback started, everything that comes from the input of the track will be recorded. Right clicking a disabled record button allows to enable Rec-Safe, thus protecting the track against accidental recording.

The buttons on the right, In and Disk, show what the user is listening to by lighting up, between the *Input* and the actual content of the playlist on *Disk*.

They also allow to override the automatic switching by pressing them to lock one source or the other to be what the user is hearing.

### Mute/Solo

These buttons allow to Mute (or silence) the track, or Solo them, shutting down the gain of the other tracks (totally by default, can be set to partially in the options). See *Muting and Soloing* for more information.

Notice that by default, Solo overrides Mute, i.e. if a track is both Soloed and Muted, it will play. That can be changed in the preferences.

The two led button above are related to solo:

- Solo Isolate, as the name suggests, isolates tracks or busses from the solo system. When tracks or busses are soloed the isolated ones will not mute.
- Solo Lock locks the solo into its current state (i.e. solo on or solo off). It will not allow the solo state to be changed until the lock is released.

## Gain & Meter

On the right of this part is a [Meter](#), displaying the level of the track's output after the fader. It can be set to display the signal at any point, see below *Metering Point*. Right clicking this meter shows a menu allowing to switch the meter type.

The big Gain slider on the left allows to change the gain of the track. Its default OdB value is reminded with a white horizontal line, and its precise value is shown in a text field above it, that doubles as a way to type in a numeric value.

The text field above the meter shows the “Peak”, i.e. the maximum value that has been reached during playback. To avoid distortion, the value should stay below OdB, and if it goes above this value, the text field will turn red. Clicking on this field will reset the Peak value (for a new measurement or a new part of the track).

Notice that if any gain automation has been set and the automation state is set on “Play” (see below), then the Gain fader is driven by the automation, and not by the user. The Gain fader will turn grey to show it is inactive.

## VCAs

If at least one VCA exists, this button will show up, allowing the user to link this track to any control master.

Clicking the button lists all the available control masters, and a menu option to Unassign all. Notice that a track can be a slave to as many VCAs as they are in the session, hence multiplying the number of VCA buttons. The displayed number is the number of the VCA, not the count of VCAs linked to the track. A track with no VCA assigned will show a unique button with a “-vca-” label instead of this number.

## Fader automation/mix group/metering point

### Fader automation mode

This button allows to choose the mode used regarding automation:

Manual	(default) The playback won't use the fader automation data
Play	Enables playback/use of fader automation data
Write	While the transport is rolling, all fader changes will be recorded to the fader automation lane
Touch	While the transport is rolling, touching the fader will initiate recording all fader changes until the fader is released. When the fader is not being touched, existing automation data will be played/used to control the gain level.

## Mix group

This button displays the mix group information as does the tab in the header (see above). It is convenient though, as it allows to quickly switch the track from one group to another with a drop down menu, also allowing to affect the track to a non-adjacent group (which the tab won't easily allow).

### Metering Point

The metering displayed in the meter is by default is ‘Post’, i.e. Post fader. It can be changed with this button to Any point of the signal flow:

In	The input of the track
Pre	Pre-fader
Post	Post-fader
Out	The output of the track
Custom	A <i>Meter</i> processor is added to the processor box and can be set anywhere (by dragging and dropping) to probe the signal flow at that point

### Output(s)

This button is exactly the same as the *Input* button, but applies to the *output* of the track.

### Comments

This buttons open up a little text editor, that can be used to add some written notes to the track, as e.g. a particular setting. The button’s caption is replaced by the beginning of the text, so it can be used as a “sub” name for the track.

## 27.2 - Audio/MIDI Busses Mixer Strips

An Ardour bus can be considered a virtual track, as in a track that doesn’t have a playlist (so, no regions). Its use is to “group” some audio signals to be treated the same way. One simple use case is to group all the audio tracks containing the different drums of a drum kit. Routing all the drum tracks’ outputs to a bus allows, once the different levels amongst the drums have been set, to adjust the global level of the drum kit in the mix.

Bus usage goes way beyond this simple example though: busses, as tracks, can receive plugins for common audio treatment, and be routed themselves as needed. This makes for a very useful tool that is very commonly used both for musical purposes and computing ones: instead of using e.g. ten discrete delay plugins on ten different tracks, busses are often used as receivers of *sends*, and only one delay plugin is used on this bus, reducing the processing power needed.

Audio Busses vs MIDI Busses Ardour supports two types of busses: Audio and MIDI. A MIDI bus differs from an audio bus just by:

- its input (which is midi, as shown by the red signal lines in the processor box) instead of  $n$  audio
- the fact that an instrument can be placed on it at creation time, whereas it can’t easily be done for an audio bus
- as for tracks, the MIDI bus doesn’t have a trim knob or invert phase button(s).

MIDI busses provide a particularly efficient workflow for virtual drum kits where the arrangement uses different MIDI tracks. Moreover, busses with both Audio and MIDI inputs are well suited for vocoders and similar plugins, where a MIDI signal controls an audio one.

Adding any audio input to a MIDI bus transforms it into an audio bus.



## Description

Busses look and behave exactly like tracks, so they share nearly *all of their controls*. The differences are:

- as the busses don't have a playlist (and cannot host any media), they can't be recorded on. The recording controls are not present
- an Aux button replaces these controls.

Clicking the Aux button makes every track that sends a signal to this bus through *Aux sends* blink in turquoise. Right clicking this button brings up a menu:

Assign all tracks (prefader)	Creates an Aux Send in every track, to this bus. The send is placed just before the fader
Assign all tracks and busses (prefader)	Creates an Aux Send in every track and every bus, to this bus. The send is placed just before the fader
Assign all tracks (postfader)	Same as above, but the send is placed just after the fader
Assign all tracks and busses (postfader)	Same as above, with tracks and busses
Assign selected tracks (prefader)	Same as for all tracks, but only applies to the selected tracks
Assign selected tracks and busses (prefader)	Same as for all tracks and busses, but only applies to the selected tracks and busses
Assign selected tracks (postfader)	Same as above, but the send is placed just after the fader
Assign selected tracks and busses (postfader)	Same as above, with tracks and busses
Set sends gain to -inf	For all the sends to this bus, put the send fader to $-\infty$ so no signal is sent
Set sends gain to 0dB	For all the sends to this bus, put the send fader at the default position, 0dB (100% of the signal is sent)

## Connecting a track to a bus

Depending on the user's workflow and the way busses are used, two possibilities exist:

### Connecting a track to a bus via its outputs

Connecting the output(s) of a track to the input(s) of the bus sends *all* the audio/MIDI to the bus. In the mixer strip, select (at the bottom) the OUTPUT button (often, by default, "Master"), and in the list, choose the input of a bus. Note that only the bus able to receive this output will show up, e.g. a mono bus won't be able to be connected to the output of a stereo track).

Obviously, doing so will (by default) disconnect the output from the Master's input, which means all the audio/MIDI will be routed to the bus. For more complex routing, the OUTPUT button allows to show the Routing Grid that allows to plug the output of the track to multiple outputs at once, be it busses, tracks, Master... The button will then reflect these multiple connections by showing a *\*number\**, number being the number of connections made in the routing grid.

### Connecting a track to a bus via Sends

This allows not to interrupt the natural flow of the signal, i.e. the track will still output to what its connected to (e.g. Master). The signal is "tapped" at the point of insertion of the send, to be sent to the



Fig. 36: Connecting a bus through a track's outputs



Fig. 37: Connecting a bus through a send

bus, by right clicking where in the signal flow the signal should be tapped, and selecting New Aux Send... > name\_of\_the\_bus.

By left-clicking the send meter, it is possible to adjust the amount of signal sent to the bus. This is often the way tracks are connected to an effect bus, like a Delay bus.

Busses can be plugged to other busses, through outputs or sends. Both example workflows discussed previously, i.e. busses for grouping tracks and busses for effects, can both coexist, as e.g. a “grouping” drum bus can have a send to a reverb bus, and be connected to a compressor bus.

### 27.3 - VCA Mixer Strips

Although track/bus *groups* offer a certain kind of grouped-control over gain, solo, mute and more, traditional mixing consoles have long had group master channels (“VCAs”) which allows to combine both a single fader to control the group level while also allowing to easily adjust the relative levels inside the group. For large projects, this can make mixing much easier to control.

Ardour implements those VCAs in a way that allows to use either or both of the conventions used on different traditional consoles for combining multiple masters:

- Nest VCAs (VCA 2 controls VCA 1 etc.)
- Chain VCAs (VCA 1 and VCA 2 both control track or bus *N*)

#### Description of the VCAs

A VCA strip is made of (from top to bottom in the screenshot):

1. Number of the VCA
2. X button: Allows to hide the VCA strip. Left clicking this button toggles the exclusive visibility of the tracks connected to this VCA
3. Name button
4. M: mutes the VCA, S: solos the VCA
5. Level meter: allows to adjust the level of the VCA
6. ~vca~: a VCA button to optionally connect to another VCA

Right-clicking the name button shows a context menu comprised of:

Rename	Renames the VCA
Color...	Changes the color of the VCA button in the tracks connected to this one
Drop All Slaves	Deletes all connections to this VCA, i.e. no tracks are controlled by this VCA anymore
Remove	Deletes this VCA

#### Connecting to a VCA strip

Connecting a track/bus/VCA to a VCA is as simple as clicking the VCA button that appears on any mixer strip under the main fader and choosing the VCA to connect to.

The VCA button only shows up in mixer strips when at least one VCA exists, i.e., a VCA must be created *before* connecting tracks to it.

Clicking the VCA button shows all the VCAs in the session, and any or all of this VCA can be checked to link them to the track, making this track controlled by multiple VCAs. The track will then show multiple

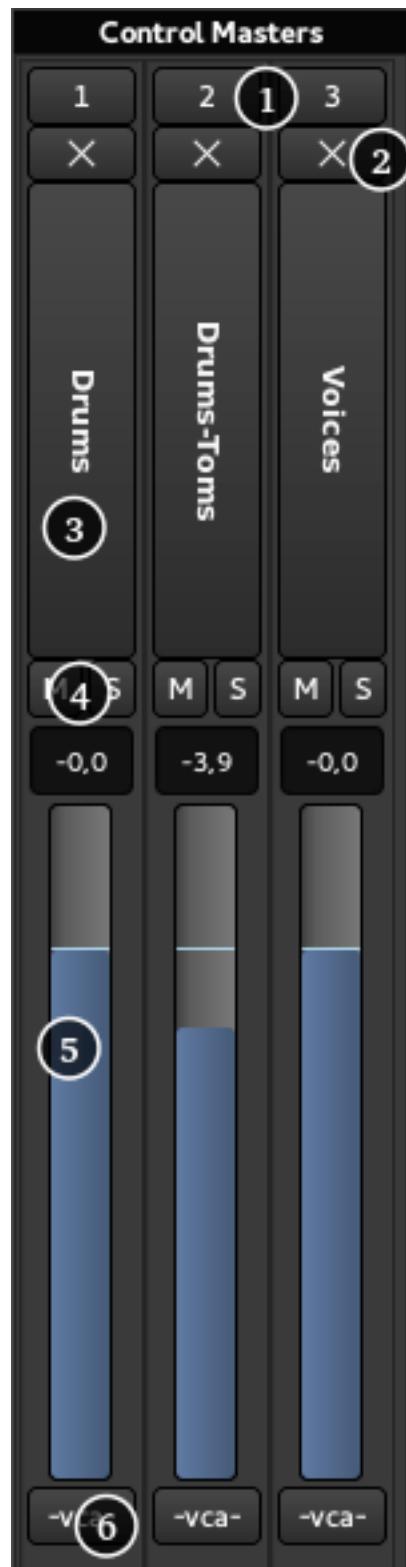


Fig. 38: A VCA mixer strip



Fig. 39: Connecting to a VCA

buttons. Disconnecting a VCA from a track is done by unchecking this VCA in the list that pops up, or clicking Unassign All to disconnect from all VCAs at once.

## 27.4 - Master Bus Strip

The Master strip in Ardour is very similar to the other busses mixer strips. The list of differences is (from top to bottom):

- There is no color affected to the master strip
- The master strip cannot be hidden, so there is not X in the top right
- It is by definition always solo, so no Solo, Iso or Lock buttons. It is replaced by a button to show the Monitoring section if *the session has one*
- It cannot belong to a mix group, so the button is removed.

The Master bus strip is *always* fixed, at the right end of the mixer, regardless of the scrolling position.

## 28 - Editor Tracks

### 28.1 - Audio Track Controls

At the top-left of the controls is the name of the track, which can be edited by double-clicking on it. The new name must be unique within the session.

Underneath the name is the track's main level fader. Changing it will affect the whole track :

- dragging will change the fader's value as per the mouse's position

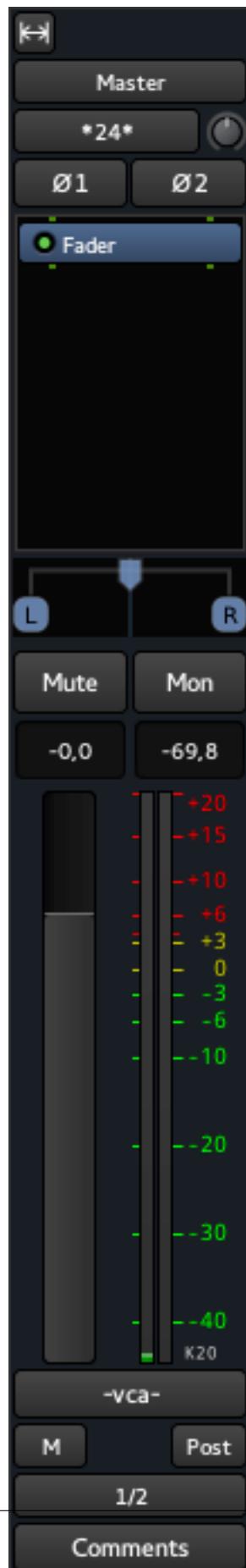


Fig. 40: The Master strip in the mixer



Fig. 41: An audio track header section

- clicking will set the fader to  $-\infty$
- clicking will reset the fader to its original 0dB position.

On the right-hand side of the headers are level meters for the outputs of the track (1 level per output).

The control buttons are:

(Record)	The button with the pink circle arms the track for recording. When armed, the entire button will turn pink, and change to bright red as soon as the transport is rolling and the track is recording. Right clicking will allow to enable/disable Rec-safe, protecting the track against accidental recording.
M (Mute)	Mutes the track. Right clicking displays a menu which dictates what particular parts of the track should be muted.
S (Solo)	Soloes the track. The behaviour of the solo system is described in detail in the section <a href="#">Muting and Soloing</a> . Right clicking will allow to enable/disable Solo isolate and Solo safe.
P (Playlist)	Opens a playlist menu when clicked. The menu offers various operations related to the track's playlist.
A (Automation)	Opens the automation menu for the track. For details see <a href="#">Automation</a> .
G (Group)	Allows to assign the track to an existing or a new group. For details see <a href="#">Track and bus groups</a> .

## 28.2 - MIDI Track Controls

A typical MIDI track header looks like this:

To show the full set of MIDI track controls, the *track height* must be increased beyond the default. MIDI tracks show only a few of the control elements when there is insufficient vertical space.

A MIDI track has the same basic controls as an *audio track*, with the addition of three extra elements:

1. Some meters for the track's outputs (MIDI in red, Audio in green)
2. The Scroomer, a zoom and scroll controller for the midi notes range
3. When the track is tall enough, the External MIDI Device selection dropdown appears.

Also, right clicking the (record) button shows the *Step Entry* dialog instead of controlling the rec-safe.

### The Scroomer

The Scroomer performs a couple of functions:

- The scrollbar controls the range of pitches that are visible on the track, as visualized by the piano keyboard. Dragging the body of the scrollbar up and down displays higher or lower pitches.



Fig. 42: A MIDI track header

- Dragging the scrollbar handles zooms in and out and increases or decreases the range of visible pitches.
- Double clicking the scrollbar auto-adjusts the zooms to make the range of visible pitches fit the actual content of the track.
- Clicking on the piano plays the corresponding MIDI note for reference.

## Channel and patch selection

### The Channel Selector

A MIDI track's data may span any number of the 16 available MIDI channels, and sometimes it is useful to view only a subset of those channels; different instruments may, for example, be put on different channels. In the context menu (right click), the Channel Selector allows to control the MIDI channel(s) that will be visible in the editor.

This window also gives control over which channel(s) will be recorded, and which will be played back, choosing between:

- All channels
- Only selected channels —Ardour then proposes to choose amongst the 16 channels which are to be recorded/played
- Force all to one channel —Ardour then ‘routes’ all the channels to one user selectable channel.

### The Patch Selector

The Patch Selector window is an easy way to set which instrument will be used on any of the MIDI channels. Although patches can be changed at any time using a *patch change*, this dialog provides an easy and convenient way to preview patches in software and hardware instruments. It integrates fully with Ardour’s support for MIDNAM (patch definition files), so Ardour can display named programs/patches for both General MIDI synths and those with MIDNAM files.

The window itself makes it easy to choose a channel, a bank number, optionally choosing a bank number through its MSB and LSB numbers (CC#00 and CC#32) for large banks, then choosing an instrument.

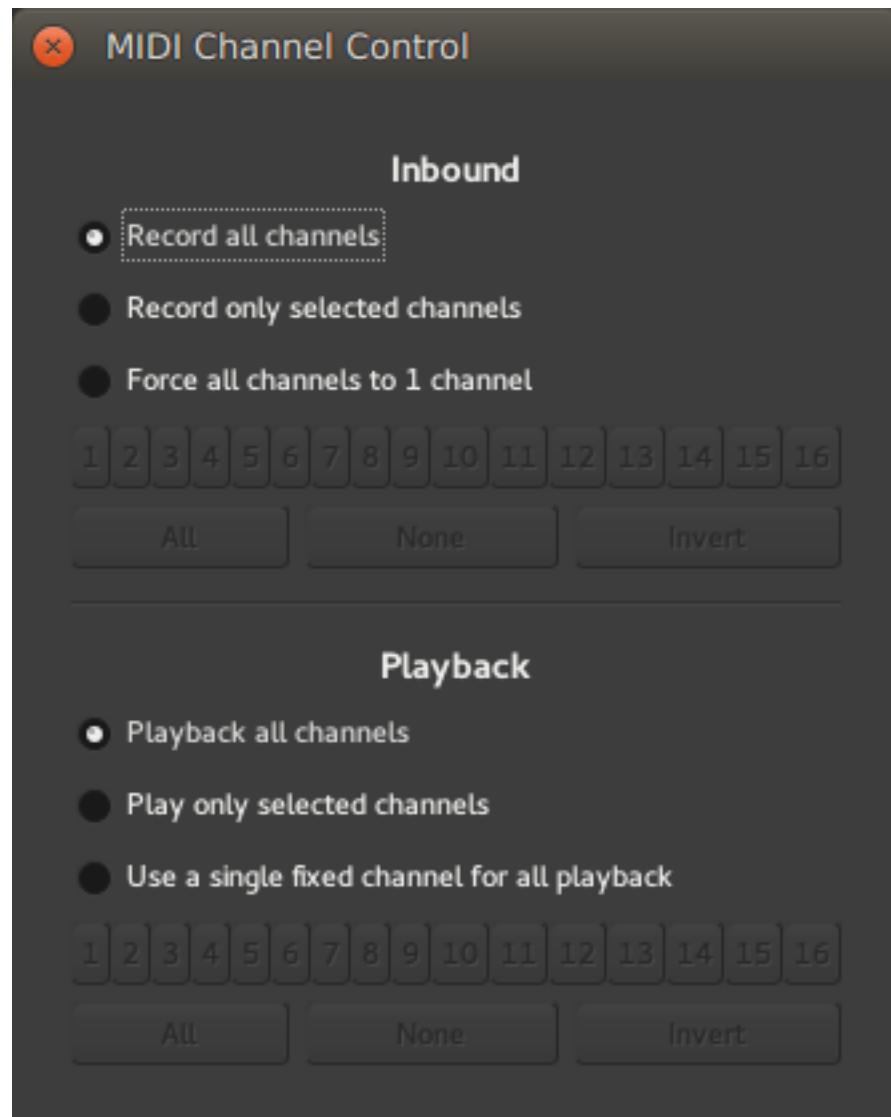


Fig. 43: The MIDI channel control window



Fig. 44: The Patch Selector window

The keyboard at the bottom of the window allows for a quick preview of the selected instrument, either automatically (using the buttons on top of the keyboard) or manually by either clicking a note or using the computer keyboard as a piano keyboard.

To edit the contents of a MIDI track see [MIDI Editing](#).

### 28.3 - Bus Controls



Fig. 45: A bus header section

The bus' header is very similar to the *audio track header*, minus :

- the playlist button, as a bus doesn't have any playlists or regions by itself, it is only a pipe to route audio or midi through
- the record button, for the same reason.

For more information about the bus concept, see [Understanding basic concepts](#).

### 29 - Track and Bus Groups

Tracks and busses can be put into groups. Members of a group can share various settings—useful for managing tracks that are closely related to each other. Examples might include tracks that contain multiple-microphone recordings of a single source (an acoustic guitar, perhaps, or a drum kit).



Fig. 46: Track headers for a group

Tracks and busses can be grouped in various ways. In the editor window, a track's controls might look like the adjacent image.

The green tab to the left of the track header indicates that this track is in a group called Fred. These tabs can be dragged to add adjacent tracks to a group.

#### Create New Groups

There are several ways to create groups for tracks and busses:

- Right-clicking on the group tab and using one of the Create... options there. A group can be created with no members, or one that starts with the currently selected tracks, or record-enabled tracks, or soloed tracks.

- Alternatively, clicking the g button on a track header to open the Group menu. The menu lists the available groups. Selecting one of these groups will add the track or bus to that group. The menu also allows creating a new group.
- Finally, the *Groups list* has a plus (+) button at the bottom of the list that can be clicked on to create a new group.

### Remove Groups

Context-clicking on a group tab and selecting Remove Group from the menu removes it. Removing a group does *not* remove the members of a group.

Groups can also be removed by selecting them in the *Groups list* and then pressing the minus (–) button at the bottom of the list.

### Add/Remove Tracks and Busses From a Group

Clicking the g button displays a menu with a list of the available groups. Selecting one of these groups adds the track or bus to that group. Selecting No Group removes it.

Alternatively, a group tab can be dragged to add or remove tracks from the group.

### Activate/Deactivate Groups via the Group Tab

Clicking on a group tab toggles the group between being active and inactive. An inactive group has no effect when editing its members. An active group will share its configured properties across its members. Tabs for disabled groups are coloured grey.

### Modify Group Properties

Edit the properties of a group is done by right-clicking on its tab and choosing Edit Group.... This opens the track/bus group dialog, which is also used when creating new groups.

### Group Color

Clicking on the color selector button changes a group's color. This affects the color of the group's tab in the editor and mixer windows. The color does *not* affect the color of the group members unless the shared Color property is enabled.

### Shared Properties

Gain means that the track faders will be synced to always have the same value; Relative means that the gain changes are applied relative to each member's current value. If, for example, there are two tracks in a group with relative gain sharing, and their faders are set to –3 dB and –1 dB, a change of the first track to a gain of –6 dB will result in the second track having a gain of –4 dB (the *difference* of the gains remains the same).

*Muting*, *Soloing*, *record enable*, *active state*, *color* and *monitoring* are all straightforward. They simply mean that all member tracks or busses will share the same settings in these respects.

Selection means that if a region is selected or deselected on one member track, *corresponding regions* on other member tracks will be similarly selected. Since region editing operations are applied to all currently selected regions, this is the way to make edits apply across all tracks in the group.

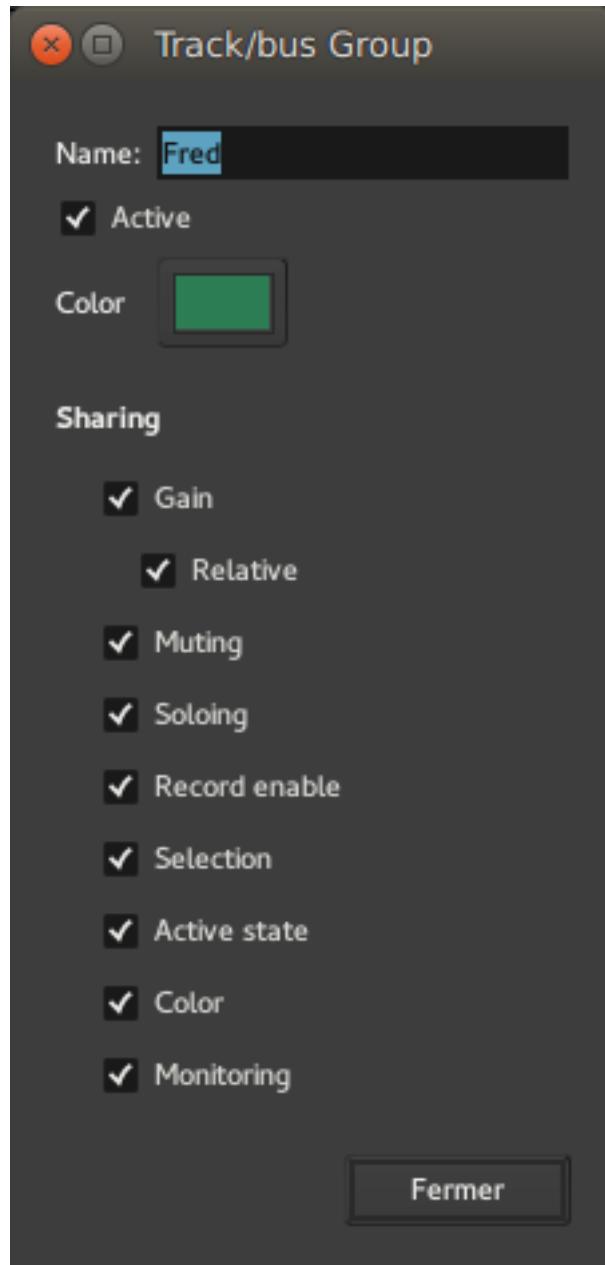


Fig. 47: The Track/bus Group dialog

### Group Tab Context Menu

Context-clicking on the group tab offers a further menu of group-related actions.

Create a New Group	create a new group
Create New Group from...	create a new group and automatically add ...
Selected	all currently selected tracks and busses
Rec-enabled	all currently record-enabled tracks
Soloed	all currently soloed tracks and busses
Collect Group	moves all the member tracks so that they are together in the editor window
Remove Group	removes the group (and only the group, not its members).
Add New Sub-group Bus	creates a bus (giving it the name of the group) and connects the output of each member to the new bus.
Add New Aux Bus	adds a bus and gives each member a send to that bus. There are two options for this, specifying whether the sends should be placed pre- or post-fader.
Fit to Window	will zoom the member tracks so that they fill the editor window.
Enable All Groups	makes all group active, including any hidden groups.
Disable All Groups	makes all groups inactive, including any hidden groups.

### 30 - Monitor Section

The Monitor section is an optional feature that provides Control Room/Monitor Speaker outputs. It can be activated for the current session in the *Session > Properties window* by enabling the Use monitor section in this session option in the Monitoring tab. By default the Monitor Section is fed with audio from the Master Bus, but depending on solo mode and other functions such as Auditioning, other audio sources may be temporarily heard instead.

The Monitor section appears on the right hand side of the Mixer, and comprises:

1. Detach/attach control. This separates the Monitor section into its own floating window
2. Status indicators for important functions
3. Solo behaviour selection
4. Show, hide and status of the Monitor Sections inline processors
5. Level controls for solo functionality
6. Level control for Monitor Dim
7. Individual monitor path controls
8. Mute, Dim and Mono functions for the monitor outputs
9. Monitor level control
10. Monitor output routing

### Status Indicators

The Status indicators, two of which also appear in the *Transport tool bar*, flash to indicate when that function is in operation:



- Soloing: This indicates when one or more tracks or busses are currently being soloed. See [Muting and Soloing](#). Clicking on this indicator cancels all currently soloed channels or busses
- Auditioning: This indicates when an audio file is being listened to directly, e.g. when using the import dialogue, or using the Audition context menu in the *Regions List*. Clicking this indicator cancels the current audition
- Isolated: This indicates when one or more tracks or busses are solo isolated. See [Muting and Soloing](#). Clicking on this indicator cancels any current isolation.

### Solo behaviour selection

The SiP, PFL and AFL controls inter-cancel with each other and select the desired Solo mode. Excl. Solo and Solo Mute then modify the modes behaviour. See [Muting and Soloing](#). The current mode is indicated by the illuminated ‘LED’ on the button.

SiP	This selects <i>Solo In Place</i> as the current solo mode and cancels the previous mode.
PFL	This selects <i>Pre Fade Listen</i> as the current solo mode and cancels the previous mode.
AFL	This selects <i>After Fade Listen</i> as the current solo mode and cancels the previous mode.
Excl. Solo	This enables or disables the <i>Exclusive Solo</i> option.
Solo » Mute	This enables or disables the <i>Solo Mute</i> option.

Changing the solo mode (SiP, PFL or AFL) will update the labels on the mixer strips’ solo controls accordingly.

### The Processors button

Clicking the Processors button show or hides the Monitor Sections processor box. This is used in the same way as processor boxes present in *tracks* and *busses*. It can be used to insert plugins, e.g. a room correction EQ or a specific metering type.

As this processing is local to the Monitor Section it is only applied to audio that is ultimately available at the monitor outputs.

### Solo level controls

These controls set the level of the audio when a channel or bus solo is engaged.

Solo Boost	This is the level that will be added to the current main monitor level when a track or bus is soloed, providing a convenient boost in level for the isolated signal. The rotary control has a range of 0dB to +10dB and can be set at any point between these two values. A drop down menu with pre-defined values is also provided for convenience.
SiP Cut	Only relevant to Solo in Place mode. This sets the level that all muted tracks or busses will be muted by. By default it is -&infinity; i.e. the non soloed tracks are totally inaudible. The level can be raised to make the other tracks audible, though dimmed. This is also sometimes referred to <i>Solo in Front</i> . The rotary control has a range of -inf to +0dB and can be set at any point between these two values. A drop down menu with pre-defined values is also provided for convenience.

## Dim level control

The Dim level control sets the amount by which the monitoring will be reduced when a Dim button is engaged. The rotary control has a range of -20dB to 0dB and can be set at any point between these two values. A drop down menu with pre-defined values is also provided for convenience.

## Monitor path controls

Each of the individual paths through the Monitor Section, (e.g. L and R for stereo), can be controlled individually. Four functions are available:

Mute	Mutes the selected path(s)
Dim	Reduces the selected path(s) level by the amount set with the Dim level control
Solo	Solos the selected channel(s)
Inv	Inverts the selected channel(s) polarity

## Global Monitor controls

Those buttons directly affect the output of the monitoring section:

- Mono: sums all of the paths to a single mono signal and applies it to all Monitor Section outputs.
- Dim: Reduces overall monitor level by the amount set with the Dim level control.
- Mute: Mutes all monitoring.

## Global Monitor level

This control sets the level for Monitor Section output. The rotary control has a range of -inf to +6dB and can be set at any point between these two values. A drop down menu with pre-defined values is also provided for convenience.

## Monitoring Output routing

Clicking on this button shows a menu that allows quick and convenient routing of the Monitor Section's outputs to audio hardware outputs, e.g. to feed control room monitors. It also has an option to open Ardour's routing matrix, where more detailed connectivity is available if routing to something other than hardware is required.

## Part IV - Sessions & Tracks

### 31 - Sessions

#### 31.1 - What's in a Session?

The Session is the fundamental document type that is created and modified by the Ardour workstation. A Session is a folder on a computer filesystem that contains all the items that pertain to a particular project or “recording/editing/mixing session”.

The Session folder includes these files and folders:

- `session_name.ardour` the main session snapshot

- `*.ardour`, any additional snapshots
- `session_name.ardour.bak`, the auto-backup snapshot
- `session_name.history`, the undo history for the session
- `instant.xml`, which records the last-used zoom scale and other metadata
- `interchange/`, a folder which holds the raw audio and MIDI files (whether imported or recorded)
- `export/`, a folder which contains any files created by the Session > Export function
- `peaks/`, a folder which contains waveform renderings of all audio files in the session
- `analysis/`, a folder which contains transient and pitch information of each audio file that has been analysed
- `dead sounds/`, a folder which contains sound files which Ardour has detected are no longer used in the session (during a Session > Clean-up > Clean-up Unused Sources operation, will be purged by Flush Waste Basket, see [Cleaning Up Sessions](#))

A session combines some setup information (such as audio and MIDI routing, musical tempo & meter, timecode synchronization, etc.) with one or more Tracks and Buses, and all the Regions and Plug-Ins they contain.

### 31.2 - Where Are Sessions Stored?

Sessions are stored in a single folder on the computer's filesystem. The first time Ardour is run, it will ask for the default location for this folder, with the initial choice being the current user's home folder.

After the first-run dialog, the default location can still be changed at any time via Edit > Preferences > Misc > Session Management. A particular (different) location for a session can also be specified when creating it, in the *New Session dialog*.

### 31.3 - New/Open Session Dialog

The initial Session dialog, displayed at each start of Ardour, consists of several consecutive pages:

#### Open Session Page

On this page, an existing session can be opened. Any *snapshot* of a particular session can also be accessed by clicking on the arrow next to the session name to display all snapshots, and then selecting one.

If the session is not displayed in the Recent Sessions list, the Other Sessions button will bring up a file selection dialog to navigate the file system.

Alternatively, a New Session can be created.

#### New Session page

This page allows to type in the name of a session, select a folder to save it in, and optionally use an existing *template*.

The different templates, both the "factory" ones and the ones created by the user, are easily available on the left-side panel. Depending on the chosen template, a specific Template Settings window may be shown, allowing the user to fine-tune the details of the template and/or choose between the different options of the template.

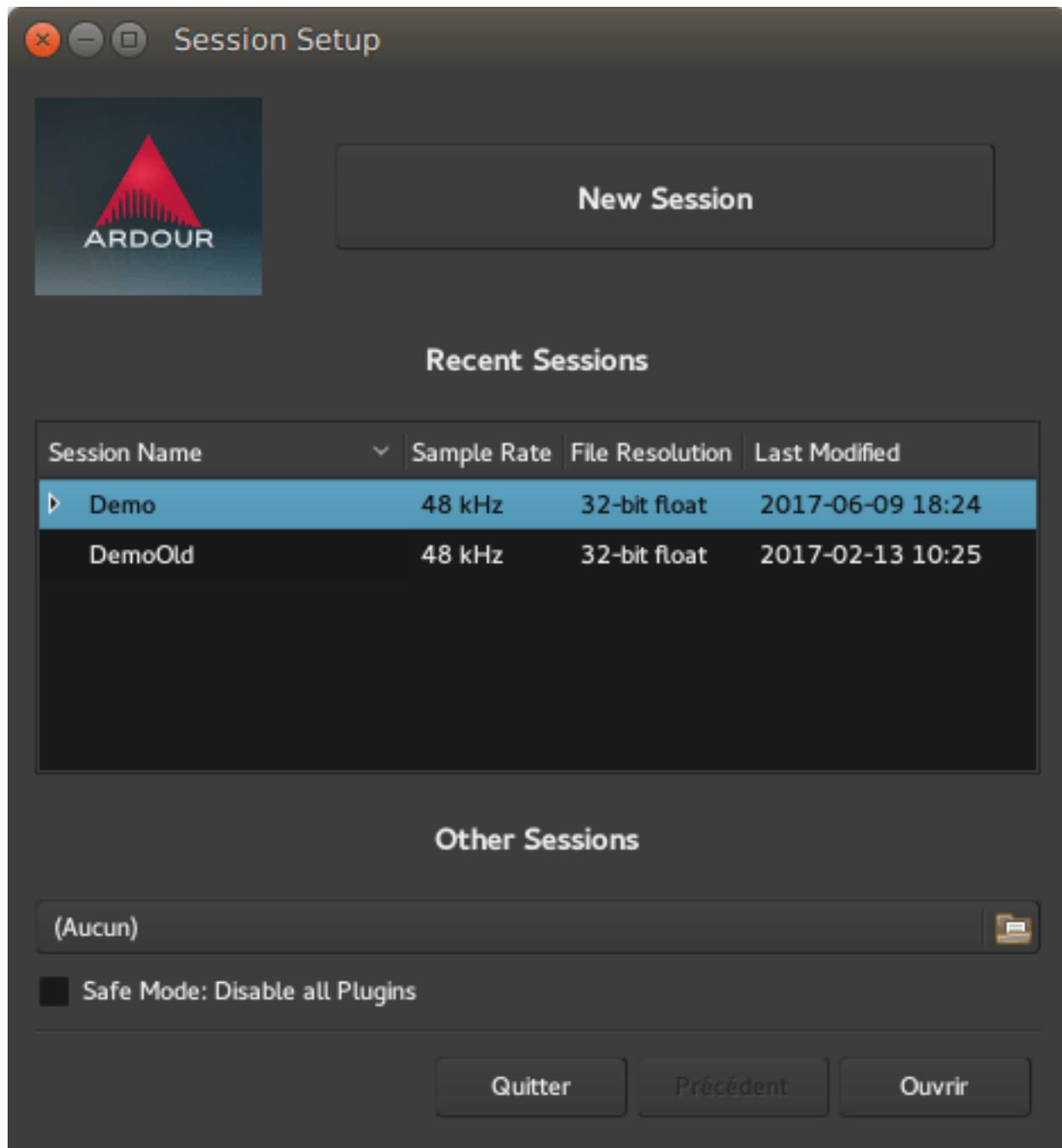


Fig. 49: The Session Setup Dialog

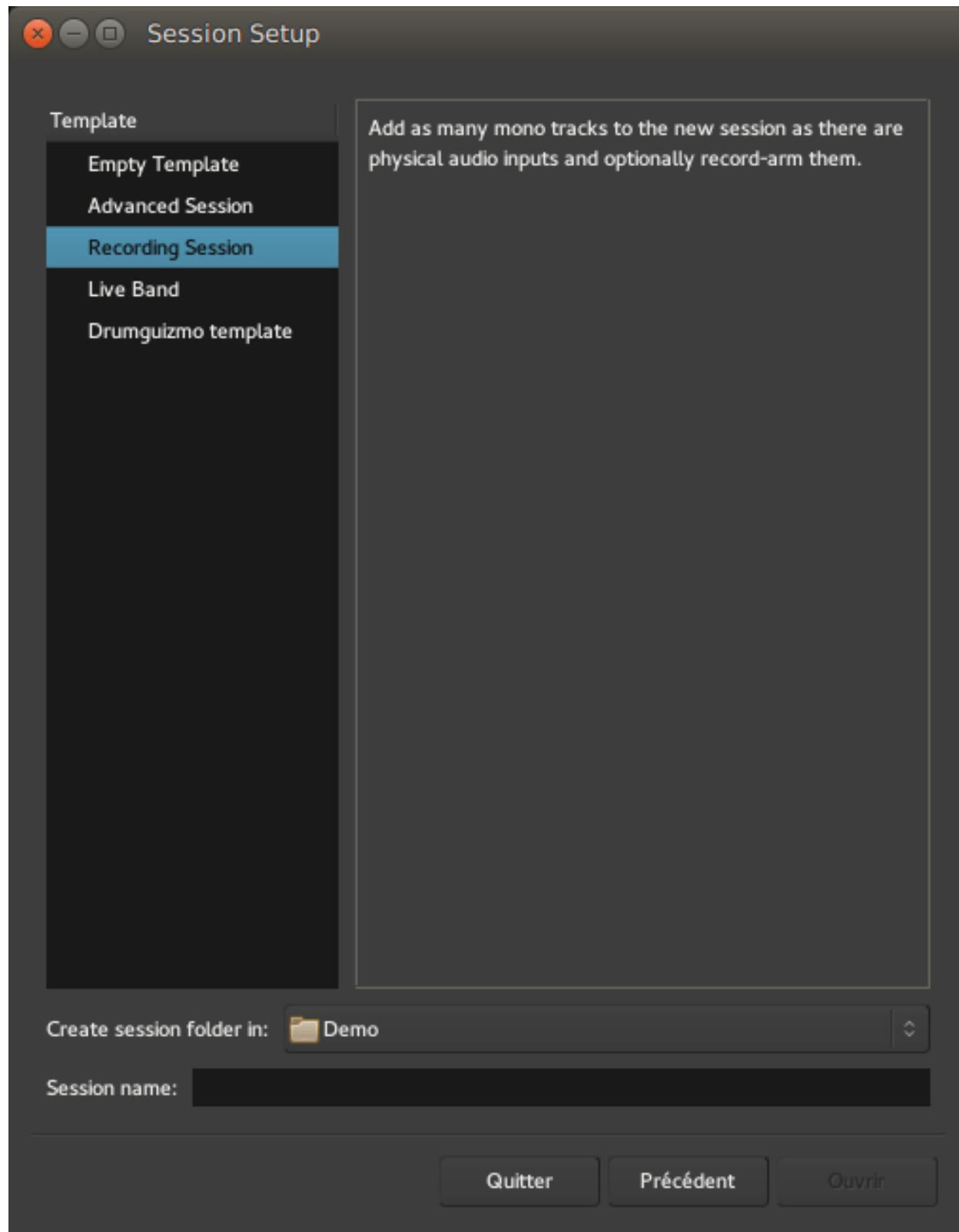


Fig. 50: The New Session Dialog

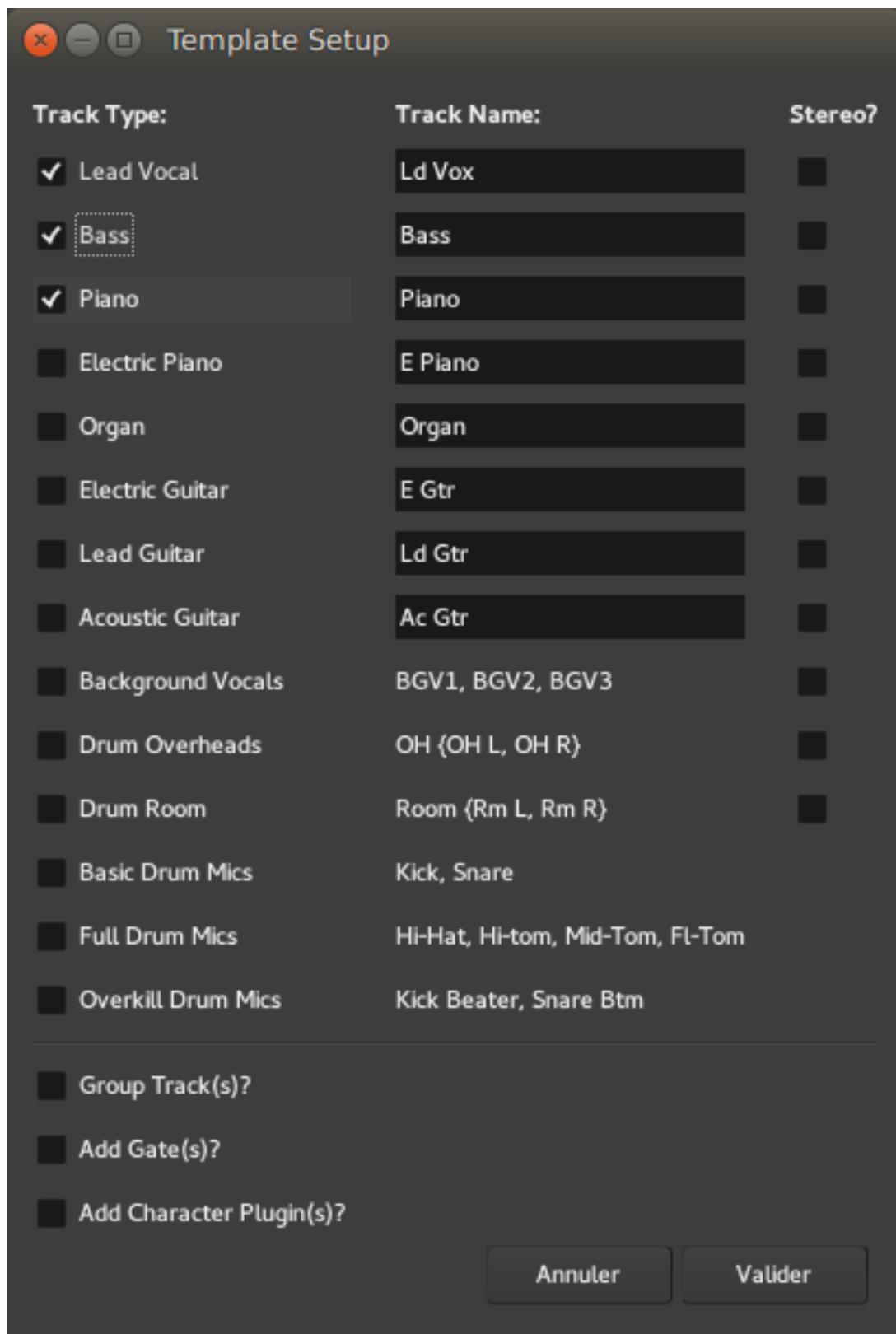


Fig. 51: One of the Template Setup Dialogs

Templates can be huge time savers when working on similar projects, or on usual projects, as they allow to preset and tweak a lot of the *session properties*, (like the availability of a *monitoring section*, connection to a Master Bus, etc.), and handle the creation of *tracks* of any kind.

The Empty Template preset allows to create a session “from scratch”. Everything a session template does can be done manually —albeit more tediously— and the resulting sessions will not differ whatsoever.

As of Ardour 5.12, which introduced the new template dialog, the factory templates are:

Empty Template	Creates an empty session with no tracks and no monitoring. A stereo Master Bus is created, and any track created defaults to output on this bus.
Advanced Session	Like the Empty Template, but adds the ability to easily manage the Master bus (channels, hardware connection, and track autoconnection), and the creation of a monitoring section.
Recording Session	Like the Empty Template, but allows the fast creation of a number of tracks, optionally ready to record.
Live Band	Fast tracks the creation of usual tracks for a band setup (vocals, guitars, piano, ...), and optionally adds usual effects on these tracks.

Selecting a template will display its description in the right-side panel, while hovering over a template name will show a tooltip indicating if it is a factory template, or, if it is a user-created one, which version of Ardour was used to create it.

Whether or not a template is used, and before the “Template Setup” dialog, the Audio/MIDI Setup will be shown.

## **Audio/MIDI Setup**

This window exposes the different audio options to be used by Ardour for the current work session, for hardware and software and is made of:

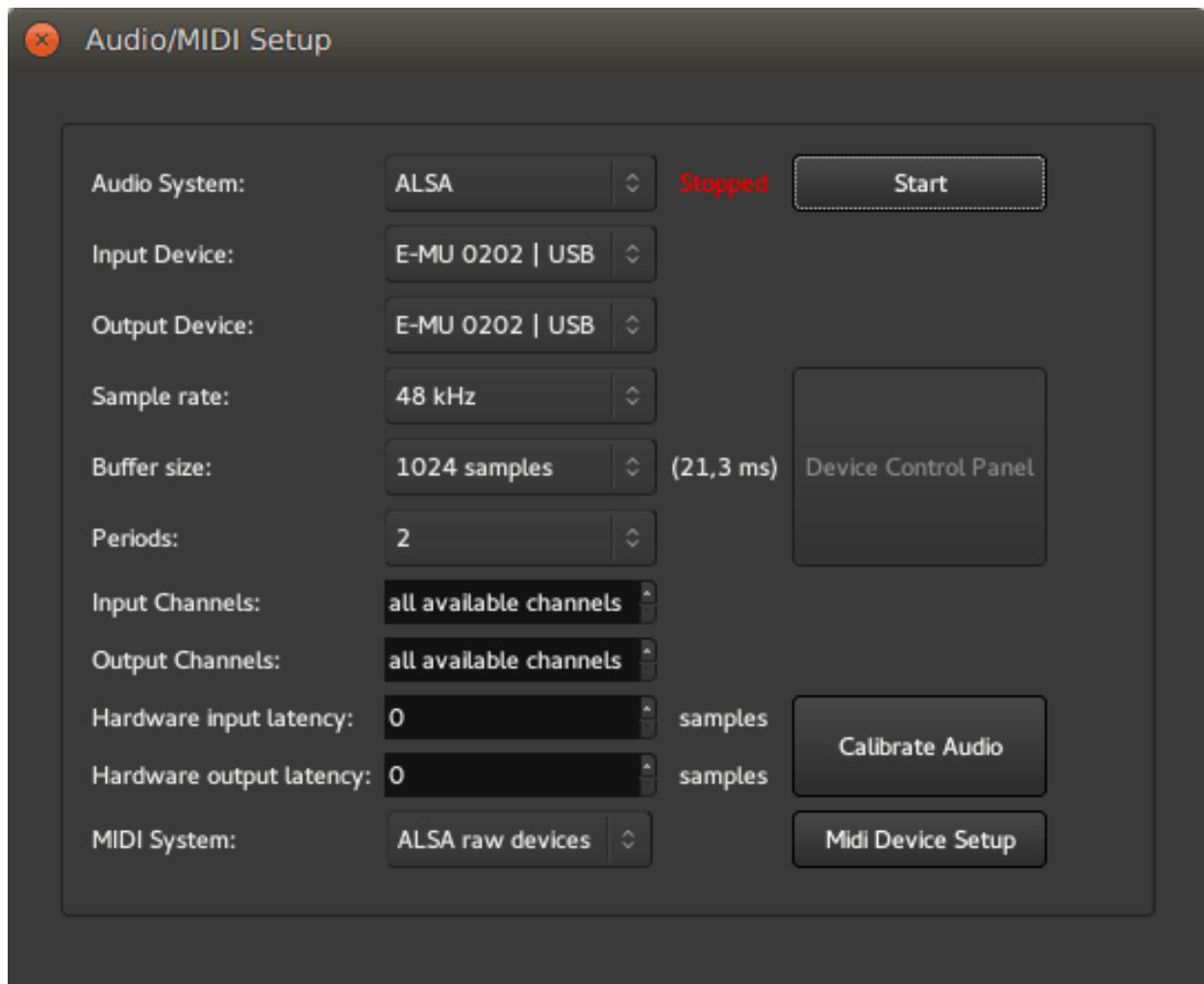


Fig. 52: The Audio/MIDI Setup Dialog

Audio System	Depending on the operating system, Ardour can possibly use different audio systems, e.g. on Linux, both ALSA and JACK are available.
Driver	On Mac OS X this will typically be CoreAudio. On Linux usually this will be either FFADO or ALSA, depending on whether or not a firewire device is used. Advanced users on all platforms may also use NetJack which provides network audio I/O.
Device	The selector should show all available interfaces provided by the driver above and which are capable of duplex operation. When using an Intel Mac running OS X and the builtin audio interface, its separate input and output devices must be <i>merged</i> first into a single “aggregate device” before Ardour will be able to use it.
Sample Rate	The selector will allow to select from any sample rate supported by the device selected above it.
Buffer Size	The size of the buffer used by the audio interface can be adjusted to allow for either lower latency, or lower CPU usage and higher latency.
In-put/Output Channels	Specifies the number of hardware channels to use. The default is all available channels.
Hard-ware In-put/Output Latency	Specify the hardware delay in samples for precise latency compensation.
Calibrate	This button runs a semi-automated guided process to obtain precise hardware latency measurements for the above option.
MIDI System	Selects the MIDI driver to use. On Mac OS X, this will be CoreMIDI. On Linux, it can be changed between two legacy ALSA drivers or the (preferred) new JACK+ALSA implementation.

### 31.4 - Renaming a Session

Using the Session > Rename menu allows to give the session a new name. A dialog will appear asking for the new one.

This operation does **not** make a new session folder—the existing session folder and relevant contents are renamed. If the session was not saved before a rename operation, it will be saved automatically and then renaming will continue.

Ardour’s Session > Save As operation will not make a new copy of the session folder and its contents. All it does is create a new session file.

### 31.5 - Session Metadata

#### Session Metadata

Sessions can have various items of metadata attached to them, and saved in the session file. These metadata are filled by the user via Session > Metadata > Edit Metadata....

These metadata will be exported as tags in the audio and video files that support it, as long as the right option is chosen at the *export* stage. All the video format can retain a subset of the metadata if the Include Session Metadata is checked in the *export video* window, while only the Ogg-Vorbis (tagged) and FLAC (tagged) audio format will be exported with the metadata (as Vorbis comment).

Ardour can also reuse the metadata from another session file in the current session, with Session > Metadata > Import Metadata.... This menu brings up a file selector, asking for the source ardour session file to extract

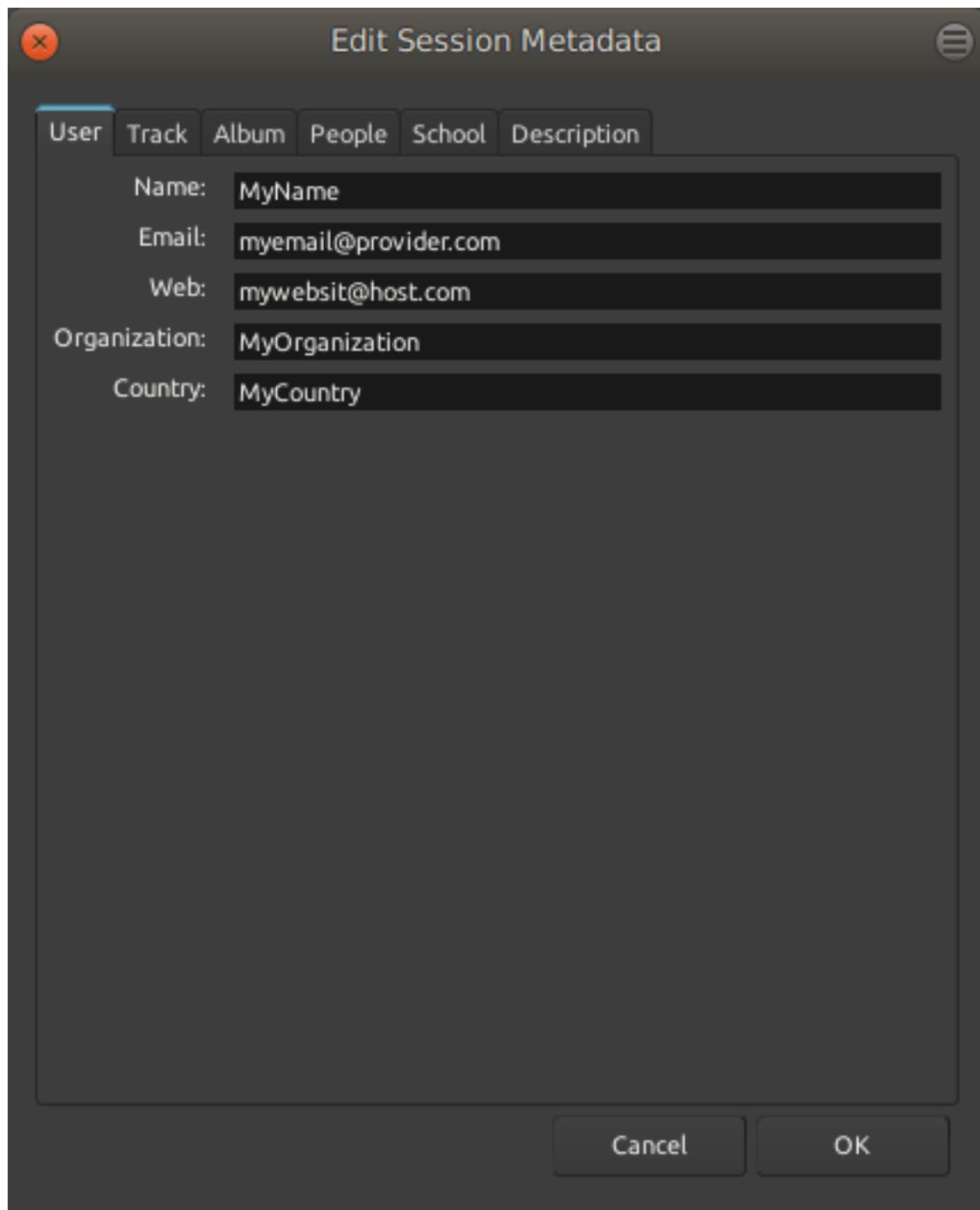


Fig. 53: The session metadata editor

the data from. This can be handy when reusing a lot of information (Author, Artist Name, etc...) or working on multiple tracks of the same media.

### 31.6 - Backup and Sharing of Sessions

An Ardour session is stored in a single folder on the computer's filesystem. This makes backup very easy: any tool capable of backing up a folder can be used to backup a session. The location of a session is picked when it is created —by default it will be in the default session location, which can be altered via *Edit > Preferences > General > Session*.

The single folder approach also makes sharing a project easy. Simply copy the session folder (onto a storage device, or across a network) and another Ardour user (on any platform) will be able to use it.

There is one complication in both cases: a session may reference media files that are stored outside of the session folder, if the user has opted not to select *Session > Import > Copy to Session* during import. Backing up a session with embedded files will not create a copy of the session containing those files. To bring those external files to the session folder, the *Session > Clean-up > Bring all media into session folder menu* can be used.

#### Using the dedicated Zip/Archive Current Session tool

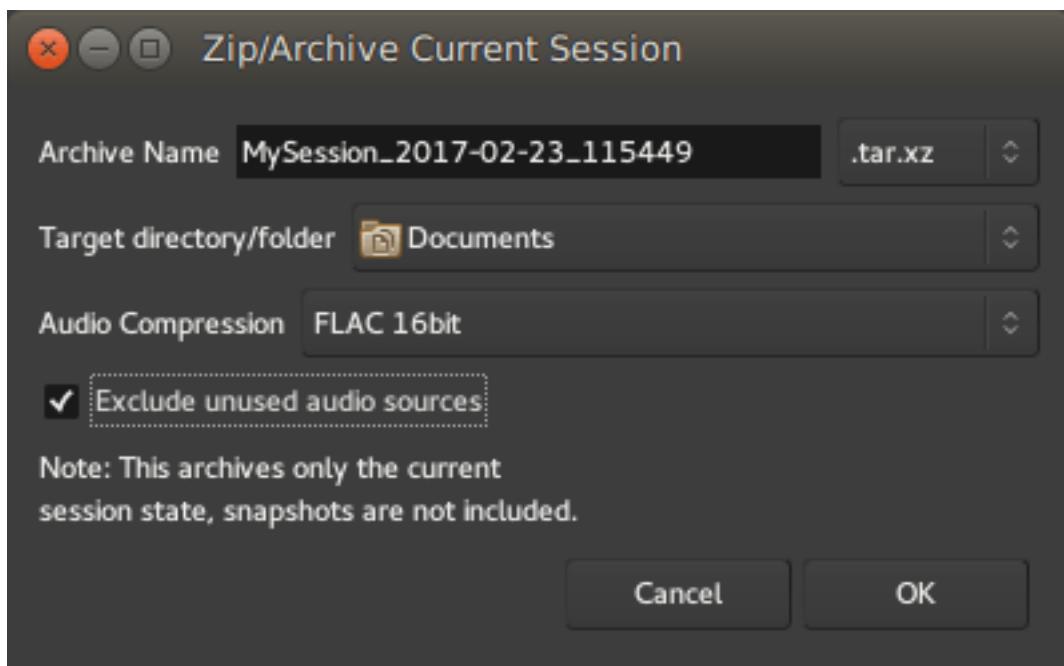


Fig. 54: The Zip/Archive Current Session window

The Zip/Archive Current Session tool is located in the *File > Archive...* menu.

It allows to create a single file containing everything useful in the session, to share it or back it up, conveniently compressed to a session-archive which is a zip-file (tar.xz to be specific) containing all the audio, MIDI, plugin-settings,... and the currently active session. Ardour can also extract those bundles (*Session > Open...*).

As opposed to zipping the entire session-folder manually,

1. the session-archive only contains the current session-snapshot and only files which are used

2. externally referenced files are included in the archive.

The window shows the following options:

Archive Name	The name of the archive file, defaulting to the name of the session followed by the date and time
a dropdown extension selector	allowing to choose between different kind or compressed archive file types
Target directory/folder	defining where in the filesystem the archive file will be generated
Audio Compression	a dropdown menu allowing to compress the audio files themselves by using an audio-tailored compression format, more on that below
Exclude unused audio sources	a checkbox to drop every audio that is in the session, but not actually used in the editor

The Audio Compression selection accepts any of:

- None
- FLAC 16bit
- FLAC 24bit

Encoding the audio sources to FLAC allows for a good size reduction of the session. It should be noted though that FLAC is a fixed-point format, meaning that if the audio in the session is in a floating-point format, this conversion will lose some information on the samples values that are rounded, though usually, this lost information cannot be perceived. Choosing “None” for Audio Compression does not compress the audio to FLAC, hence preserving the floating-point data at the cost of a bigger file size. Notice also that converting to FLAC automatically normalizes the audio.

Using the Exclude unused audio sources option allows to only keep the files actually used in the session, which can be useful to leave any unused take or reference material out of the backup, reducing the archive’s global file size.

## 32 - Tracks

### 32.1 - Track Types

Ardour offers three track types depending on the type of data they contain, and differentiates between three track modes, depending on their recording behaviour.

#### Track types

An Ardour track can be of type audio or MIDI, depending on the data that the track will primarily record and play back. *However, either type of track can pass either type of data.* Hence, for example, one might have a MIDI track that contains an instrument plugin; such a track would record and play back MIDI data from disk but would produce audio, since the instrument plugin would turn MIDI data into audio data.

Nevertheless, when adding tracks to a session, its content is typically known, and Ardour offers three choices:

Audio	An Audio Track is created with a user-specified number of inputs. The number of outputs is defined by the master bus channel count (for details see <i>Channel Configuration</i> below). This is the type of track to use when planning to work with existing or newly recorded audio.
MIDI	A MIDI track is created with a single MIDI input, and a single MIDI output. This is the type of track to use when planning to record and play back MIDI. There are several methods to enable playback of a MIDI track: add an instrument plugin to the track, connect the track to a software synthesizer, or connect it to external MIDI hardware. If an instrument plugin is added, the MIDI track outputs audio alongside MIDI data.
Audio/MIDI	There are a few notable plugins that can usefully accept both Audio and MIDI data (Reaktor is <a href="#">MIDI</a> , and various “auto-tune” like plugins are another). It can be tricky to configure this type of track manually, so Ardour allows to select this type specifically for use with such plugins. It is <i>not</i> generally the right choice when working normal MIDI tracks, and a dialog will warn of this.

## Track Modes

Audio tracks in Ardour have a mode which affects how they behave when recording:

Normal	Tracks in normal mode will record non-destructively—new data is written to new files, and when overdubbing, new regions will be layered on top of existing ones. This is the recommended mode for most workflows.
Non-Layered	Tracks using non-layered mode will record non-destructively—new data is written to new files, but when overdubbing, the existing regions are trimmed so that there are no overlaps. This does not affect the previously recorded audio data, and trimmed regions can be expanded again at will. Non-layered mode can be very useful for spoken word material, especially in combination with <i>push/pull trimming</i> .
Tape	Tape-mode tracks do <b>destructive</b> recording: all data is recorded to a single file and if a section of existing data is overdub, the existing data is destroyed irrevocably—there is no undo. Fixed crossfades are added at every punch in and out point. This mode can be useful for certain kinds of re-recording workflows, but is not suggested for normal use.

The screenshot on the right shows the subtle difference between an overdub in normal mode (upper track) and one in non-layered mode (lower track). Both tracks were created using identical audio data.

The upper track shows a new region which has been layered on top of the the existing (longer) region. It can be seen by the region name strips.

The lower track has split the existing region in two, trimmed each new region to create space for the new overdub, and inserted the overdub region in between.

## Channel Configuration

Ardour tracks can have any number of inputs and any number of outputs, and the number of either can be changed at any time (subject to restrictions caused by any plugins in a track). However it is useful to not have to configure this sort of thing for the most common cases, and so the *Add Tracks* dialog allows to select “Mono”, “Stereo” and few other typical multichannel presets

The name of the preset describes the number of input channels of the track or bus.

If Ardour is configured to automatically connect new tracks and busses, the number of outputs will be determined by the number of inputs of the master *bus*, to which the track outputs will be connected.

For example, with a two-channel master bus, a Mono track has one input and two outputs; a Stereo track has two inputs and two outputs.



Fig. 55: Normal and non-layered overdubbing comparision

If Edit > Preferences > Signal Flow > Track and Bus Connections is set to manual, then tracks will be left disconnected by default and there will be as many outputs as there are inputs. It is up to the user to connect them as desired. This is not a particularly useful way to work unless something fairly unusual is done with signal routing and processing. It is almost always preferable to leave Ardour make connections automatically, even if some changes are manually done later.

### 32.2 - Adding Tracks, Busses and VCAs

A track, bus or VCA can be added to a session by either:

- Choosing Track > Add Track, Bus or VCA....
- Right-clicking in an empty part of the track controls area.
- Clicking the Plus (+) button underneath the list of tracks in the mixer.

Any of these actions will open the Add Track/Bus/VCA dialog.

The list of available track templates (both factory and user-created ones) in the left panel allows to choose the *track(s) type* (e.g. Audio, MIDI, bus, VCA etc.). Some templates can do even more, like the factory-provided Live Band that automatically creates a number of usual tracks for a common band setup. See *New Session* for more information about templates.

The common templates have parameters to tweak:

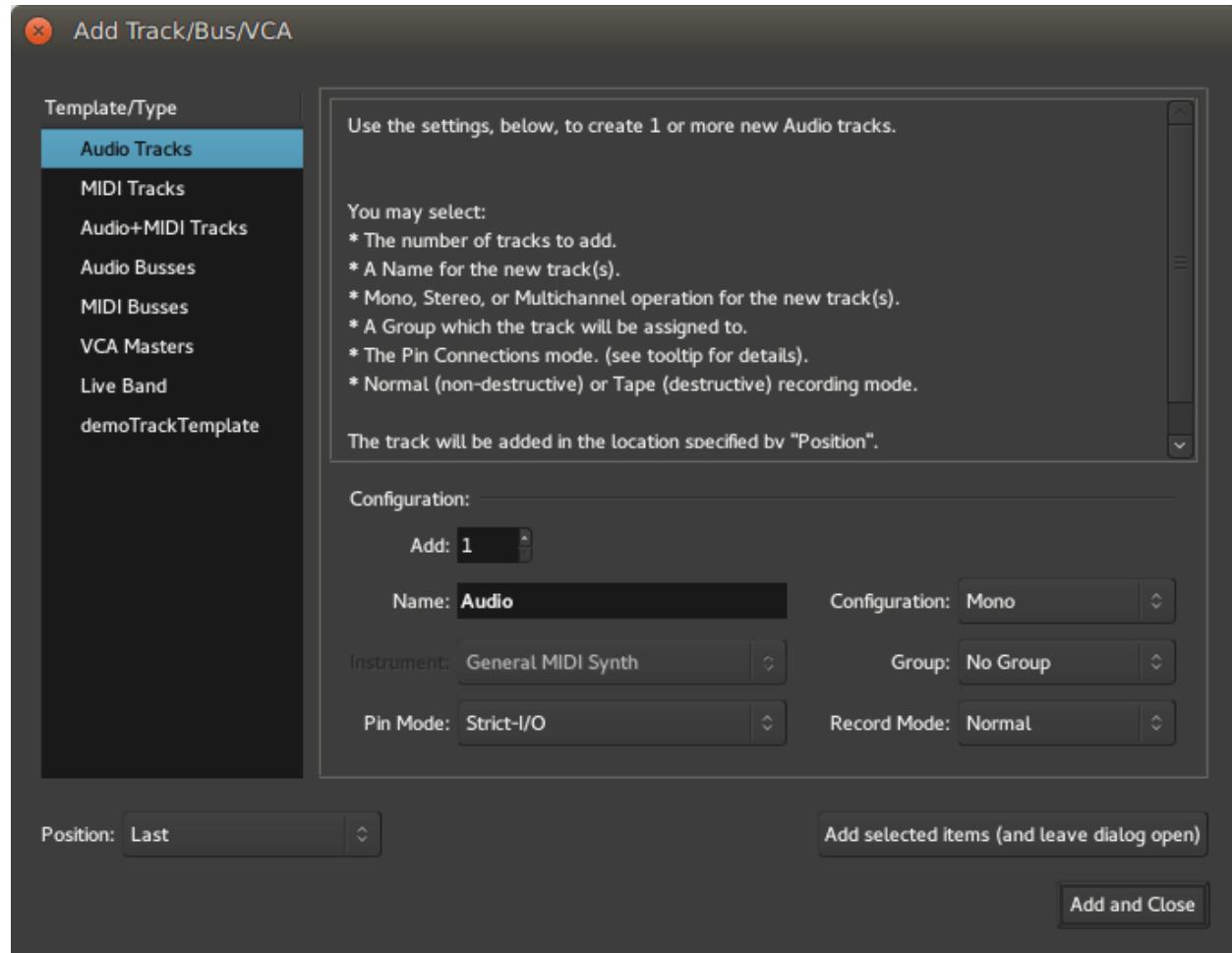


Fig. 56: The Add Track/Bus/VCA dialog.

Add	Selects the number of tracks, busses or VCAs to create.
Name	Defines the name of the new track(s). If multiple tracks are created, or if a track with the same name already exists, a space and number will be appended at the end (e.g.: Audio 1, Audio 2...).
Configuration	This menu allows to choose from a number of route templates, which determine the number of input ports and optionally contain plugins and other mixer strip configuration. The most common choices here are <i>mono</i> and <i>stereo</i> .
Instrument	This option is only available for MIDI tracks and busses and allows the selection of a default instrument from the list of available plugins.
Group	Tracks and busses can be assigned groups so that a selected range of operations are applied to all members of a group at the same time (selecting record enable, or editing, for example). This option assigns the new track/bus to an existing group, or create a new group.
Pin Mode	Defines how the number of output responds to adding a plugin with a different number of outputs than the track itself. In <i>Strict I/O</i> mode, plugins cannot alter the track's channel count, while in <i>Flexible I/O</i> mode, it will automatically adapt to the I/O of its plugins. See <i>Signal flow</i> to learn more about those options.
Record mode	This option is only available for audio tracks and affects how it behaves when recording. See <i>Track Modes</i> for details.
Position	Defines where in the track list is the track created. The default is <i>Last</i> , i.e. after all the tracks and busses, and can also be <i>First</i> , <i>Before Selection</i> (to place it just above the selected track) or <i>After selection</i> .

Multiple tracks of different types can be created by using the Add selected items (and leave dialog open) button, which, used in conjunction with the Add field, allows for a very efficient and fast way to create a base track setup.

New tracks appear in both the editor and mixer windows. The editor window shows the timeline, with any recorded data, and the mixer shows just the processing elements of the track (its plugins, fader and so on).

### Removing Tracks and Busses

Removing tracks and busses, is done by selecting them, right-clicking and choosing Remove from the menu. A warning dialog will pop up, as track removal cannot be undone; this option should be used with care!

### 32.3 - Controlling Track Ordering

Ardour does not impose any particular ordering of tracks and busses in either the editor or mixer windows. The default arrangements are as follows:

- In the Editor window, the Master bus will always be on top unless hidden. Tracks and busses will appear in their initial order, from top to bottom. The monitor section (if used) is never visible in the editor window.
- In the Mixer window, the tracks and busses will be displayed in their initial order, from left to right. The Master bus is always on the far right and occupies its own pane, so that it is always visible no matter how many other mixer strips are present. If a Monitor section is used, it shows up at the right edge of the mixer window; it can also be torn off into a separate window.

## Reordering Tracks

The track ordering of the Editor and Mixer is synchronized: if a track is reordered in one window, the ordering in the other window will follow.

### Reordering in the Editor Window

Reordering is done by selecting the tracks to be moved, then using Track > Move Selected Tracks Up (shortcut:  $\uparrow$ ) or Track > Move Selected Tracks Down (shortcut:  $\downarrow$ ).

Alternatively, the *Tracks & Busses panel of the Editor Lists* can be used, if visible. Here, tracks and busses can be freely dragged-and-dropped into any desired order.

### Reordering in the Mixer Window

Within the *Strips List* pane at the top left of the Mixer window, tracks and busses can be freely dragged-and-dropped into any desired order.

### “Collecting” Group Members

Tracks and Busses that are members of a group can be reordered so that they display contiguously within the Editor and Mixer windows, by Right-clicking on the group tab and choosing Collect.

### Ordering of New Tracks

When *adding new tracks*, the Insert: field allows to determine their placement. New tracks will be placed *Last* by default, so after the rightmost (in the mixer) or bottom-most (in the editor) selected track. If no tracks are selected, new tracks will be added at the end, regardless of the choice.

Because new tracks are automatically selected, they can be quickly reordered in the editor window via the keyboard shortcuts after adding them.

### Track Ordering and Remote Control IDs

Every track and bus in Ardour is assigned a remote control ID. When a *control surface* or any other remote control is used to control Ardour, these IDs are used to identify which track(s) or buss(es) are the intended target of incoming commands.

Remote IDs are assigned to tracks and busses in the order that they appear in the mixer window from left to right, starting from #1; manual assignment of remote IDs is not possible. The master bus and monitor section can be accessed by name.

### 32.4 - Track Context Menu

Within the editor window, context-clicking (right click) on either a region or empty space within a track displays the track context menu. The context menu provides easy access to many track-level operations.

If a region is clicked, the first item in the menu is the name of the region. If a *layered region* is clicked, the next item in the menu is Choose Top. If selected, a dialog appears that allows to change the vertical order of layers at that point. See *Layering Display* for more details.

The rest of the track context menu is structured as follows:

Play	
Play from Edit Point	Plays from the location of the current <i>Edit Point</i> .
Play from Start	Plays from the start of the session
Play Region	Plays the duration of the session from the start of the earliest selected region
Select	
Select All in Track	Selects all the regions and automation points in the current track
Select All Objects	Selects all the regions and automation points in the session
Invert Selection in Track	Select the previously unselected regions, and deselect the previously selected regions
Invert Selection	Select the previously unselected regions, and deselect the previously selected regions
Set Range to Loop Range	Creates a range selection on the selected tracks, based on the selected loop markers
Set Range to Punch Range	Same as above, based on the selected punch markers
Set Range to Selected Regions	Same as above, based on the selected regions (i.e. from the start of the first selected region to the end of the last selected region)
Select All After Edit Point	Select all the regions and automation points that exist after the Edit Point
Select All Before Edit Point	Same as above, but before the Edit point (i.e. to the left of it)
Select All After Playhead	Same as above, but considering the Playhead, regardless of the Edit Point
Select All Before Playhead	Same as above, with the Playhead
Select All Between Playhead and Edit Point	Selects all the regions between the Playhead and the Edit Point
Select All Within Playhead and Edit Point	Same as above, but the regions must be totally included between the Playhead and the Edit Point
Select Range Between Playhead and Edit Point	Creates a Range between the Playhead and the Edit Point
Edit	
Cut	Deletes the current selection, but puts it in memory ready to be pasted
Copy	Copies the current selection to memory
Paste	Pastes the memory at the <i>Edit Point</i> , after a Cut or Copy operation
Align	Aligns the sync point of all selected regions to the Edit Point
Align Relative	Same as above, but considers multiple regions as a block and aligns the block relative to the Edit Point
Insert Selected Region	If a region is selected in the <i>Region List</i> , inserts it in the track
Insert Existing Media	Inserts an external media file in the track, same as <i>the Session &gt; Insert Existing Media</i>
Nudge	
Nudge Entire Track Later	Moves all the region to the right by the amount shown in the <i>nudge time</i> preference
Nudge Track After Edit Point Later	Same as above, but only for regions that begin after the Edit Point
Nudge Entire Track Earlier	Same as above, to the left
Nudge Track After Edit Point Earlier	Same as above, to the left
(un)Freeze	Consolidates all the regions in the track into one <i>frozen</i> region which can't be edited

## 33 - Controlling Track Appearance

Ardour offers many options for controlling the appearance of tracks, including color, height, waveform style and more. These can all be found in the *Edit > Preferences > Editor* menu.

### 33.1 - Layering Display

Ardour allows arbitrary layering of regions—there can be any number of regions at a given position. By default, the regions are overlaid in the editor window, to save vertical space.

However, this display mode can be confusing for tracks with many overdubs, because it's not obvious in which order the overdubs are layered. Although there are other methods of moving particular regions to the top of an overlapping set, and although Ardour also has playlists to manage *takes* a bit more efficiently than just continually layering, there are times when being able to clearly see all regions in a track without any overlaps is reassuring and useful.

The example below shows a track with a rather drastic overdub situation, viewed in normal overlaid mode:

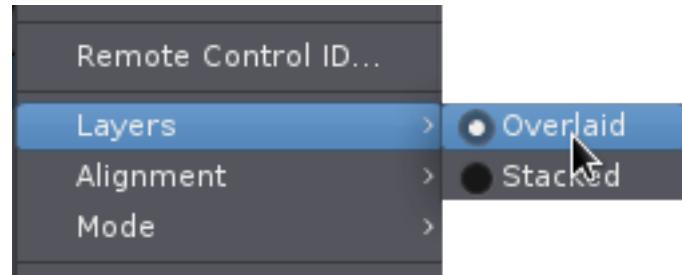


Fig. 57: The Track layering menu

Fig. 58: Overlapping regions in *overlaid* mode

This display can be changed by right clicking on the track header, showing the menu displayed above. There are two choices for layers, and overlaid is currently selected. Clicking on stacked, the track display changes to: .. figure:: /images/layers\_stacked.png

**alt** Overlapping regions in *stacked* mode

Overlapping regions in *stacked* mode

Regions can still be moved around as usual, and can be dragged so that they overlay each other again, but when the mouse button is released, things will flip back to them all being stacked cleanly. The number of lanes for the track is determined by the maximum number of regions existing in any one spot throughout the track, so if a track has 10 overdubs stacked up in one spot, it will end up with 10 lanes. Obviously, using a large track height works much better for this than a small one.

### 33.2 - Track Color

New tracks in Ardour are assigned a random color from a pastel color palette, so they should never end up being particularly bright or particularly dark.

#### Changing the color of specific tracks

Changing the color of a track is done by selecting the track(s) and right clicking on the track header of one of them then from the context menu, selecting Color and picking a hue in the color dialog. Every selected track will be re-colored.

If only one track is changed, right clicking on that track's header will be enough to select it, saving the extra mouse click.

#### Changing the color of all tracks in a group

Tracks that belong to a *track/bus group* can share a common color by enabling the Color option for the group. With this enabled, any color change will be propagated to all group members.

The group color can also be explicitly changed by context-clicking on the group tab in the Mixer, selecting Edit Group... and then clicking on the Color selector in the displayed dialog.

### 33.3 - Track Height

At some stage of the production, a quick overview over as many tracks as possible may be required, or a detailed view into just a few, or a combination of the two. To facilitate this, the height can be configured individually for each track in the editor window, or globally.

#### Resizing one or a few tracks

A right click on a track header will display the Height menu, and allow to choose from a list of standard sizes. All selected tracks will be redrawn using that height.

Alternatively, moving the pointer to the bottom edge of a track header will change the cursor to a two-way vertical arrow shape. Left-dragging dynamically resizes all selected tracks.

#### Resizing all the tracks

The three rightmost items of the *Zoom Controls*, in the toolbar, allow to quickly resize multiple tracks' heights at once, or to display a selected number of tracks in the editor, or all the selected ones, etc.

#### Fitting to the Editor Window

Fitting one or many tracks to the Editor window can be done by selecting the tracks to display and choosing Track > Height > Fit Selection (Vertical) or using the keyboard shortcut, f. Ardour adjusts the track heights and view so that the selected tracks completely fill the vertical space available, unless the tracks cannot be fitted even at the smallest possible size.

The Visual Undo (default shortcut: Z) can be used to revert this operation.

### 33.4 - Waveform display

The display of waveforms (or, more correctly, peak envelopes, since the actual waveform is only visible at the highest zoom levels) is configurable via the Edit > Preferences > Appearance > Editor dialog, to support different use cases and user preferences. The following options are available:

Show waveforms in regions

By default, Ardour draws waveforms within audio regions. Disable this option to hide them.

Waveform scale

Linear

This is the traditional linear (1:1) display of the peak envelope, or, at higher zoom levels, the individual samples.

Logarithmic

Alternatively, a logarithmic display of the peak envelope can be used. This will give a better idea of program loudness (it is similar to dBs) and plot soft passages more clearly, which is useful for soft recordings or small track height.

Waveform shape

Traditional

The zero line appears in the middle of the display and waveforms appear as positive and negative peaks above and below.

Rectified

The zero line appears at the bottom of the display and waveforms appear as absolute peaks *above* the line only.

---

CHAPTER  
**ONE**

---

**34 - IMPORTING AND EXPORTING SESSION DATA**



## 34.1 - IMPORT DIALOG

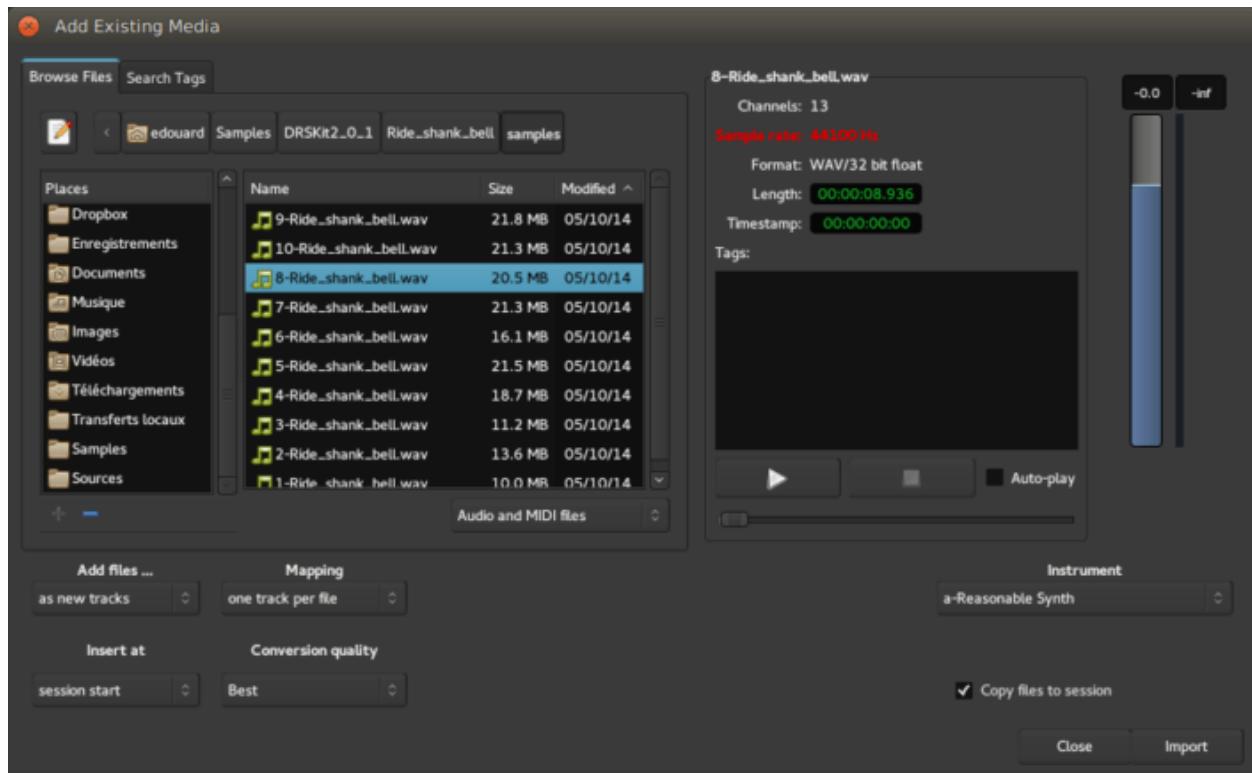


Fig. 1: The import window.

Many sessions will require the use of existing material, whether it consists of audio and/or MIDI data. Using existing samples, loops and riffs from files stored on the system can be the basis for a new session, or a way to deepen and improve one that is already underway.

Importing audio and MIDI data into the session is done with the Add Existing Media dialog, accessed by the Session > Import menu.

### 2.1 The Soundfile Information Box

This box will display information about the currently selected file:

- number of channels,

- sample rate,
- file format,
- length,
- embedded timestamp (applies to some professional formats such as Broadcast WAVE), and
- *tags* (attached metadata to help categorize files in a library).

If the sample rate differs from the current session rate, it is displayed in red, which indicates that the file must be resampled before importing.

Resampling is controlled by the Conversion quality option described below.

## 2.2 Auditor

Files can be auditioned before importing. The slider under the play and stop buttons allows to scrub around, a fader on the right side allows to control the playback volume.

Auditioning MIDI files requires a MIDI instrument to be chosen in the Instrument dropdown list.

## 2.3 Importing options

Through the Add files... option, imported files can be inserted in the session:

as new tracks	automatically creates new tracks and import the files in it
to region list	adds the files to the <i>region list</i> , from where then can be manually dragged into a track
as new tape tracks	adds the files as <i>Tape tracks</i> .

The Insert at option chooses where in time the file will be imported, amongst:

the file timestamp (if available, zero by default)

at the *edit point*

at the playhead

at the session start.

The Channel Mapping option is only available for multi-channel files (i.e. all but mono ones). It is either

one track/region per file	Creates a multi channel track for each imported file
one track/region per channel	Creates only mono channels, as many as there are channels in the imported files
sequence files	If multiple files are imported, they can be sequenced into a single track in the order of selection

The Conversion quality drop-down controls the quality of the resampling process, if the sampling rate of the source file differs from the session rate.

Finally, and most importantly, the files can be linked or copied to the session with the Copy files to session checkbox. Please read [Copying versus Linking](#) for details.

## 34.2 - SUPPORTED FILE FORMATS

The list of audio file formats that Ardour can understand is quite long. It is based on the functionality offered by `libsndfile`, an excellent and widely used software library by Australian programmer Erik de Castro Lopo. As `libsndfile`'s capabilities expand, so will Ardour's abilities to import (and export) new formats. Ardour supports all common audio file formats, including WAV, AIFF, AIFC, CAF, W64 and BWF, with all typical sample formats (8-, 16-, 24-, 32-bit integer, floating point, and more).

A full list of `libsndfile`'s supported formats is available on the [libsndfile website](#).

For MIDI import, Ardour will read any Standard MIDI Format (SMF) file.



### 34.3 - ADDING PRE-EXISTING MATERIAL

There are several ways to importing an audio or MIDI file into a session:

- Session > Import
- Region List context menu: Import To Region List
- Track context menu: Import Existing Media

These methods are all equivalent: they open the *Add Existing Media* dialog.

Finally, files can also easily be imported into a project by dragging and dropping a file from some other application (e.g. the system's file manager). Files can either be dragged onto the Region List, into the desired track or into an empty space in the editor track display.

The file will be imported and copied into the session, and placed at the position where the drag ended.



## 34.4 - COPYING VERSUS LINKING

Copying and linking are two different methods of using existing audio files on the computer (or network file system) within a session. They differ in one key aspect:

### 5.1 Copying

An existing media file is copied to the session's audio folder, and if necessary converted into the session's native format.

For audio files, the format can be chosen (e.g. WAVE or Broadcast WAVE). Audio files will also be converted to the session sample rate if necessary (which can take several minutes for larger files).

MIDI files will already be in SMF format, and are simply copied into the session's MIDI folder.

### 5.2 Linking

A link to an existing media file somewhere on the disk is used as the source for a region, but the data is **not copied or modified** in any way.

While linking is handy to conserve disk space, it means that the session is no longer self-contained. If the external file moves, it will become unavailable, and any changes to it from elsewhere will affect the session. A backup of the session directory will miss linked files.

The Copy file to session option in the Import dialog window allows to choose to copy or link files into the session:

Copy file to session	This file will be imported in the audio/MIDI folder of the session.
Copy file to session	This file won't be copied.

There is a global preference Edit > Preferences > General > Session > Always copy imported files. If it is enabled, linking a file will not be possible.



## **34.5 - SEARCHING FOR FILES USING TAGS**

A tag is bit of information, or metadata, that is associated with a data file. Specifically, tags are texts, keywords or terms that have some relevance to a particular sound file. Ardour can store these tags in a searchable database so that they can quickly be searched for to retrieve sounds based on the tags that have been assigned to them.

For example if the term 120bpm has been assigned to a sound, search later for this tag will make the file appear in the search list. Tags are independent of the filename or anything else about the file. Tags, and the file paths that they are associated with, are stored in a file called sfdb in the Ardour user folder.

### **6.1 Creating and adding tags**

Adding tags to a given file is done by opening the Session > Import dialog, selecting the file in the browser, and typing new tags into the tag area in the soundfile information box on the right.

Tags are stored when the input box loses focus, there is no need to explicitly save them.

To have more than one tag for a file, new tags can either be added on new lines (meaning the Enter key is pressed between two tags) or they can be separated from the previous ones by a comma (,), with or without spaces.

### **6.2 Searching for files by tag**

Searching for specific tags is done in the Search Tags tab of the same dialog. Files which have been tagged with the relevant terms will appear in the results window. Selected files can be auditioned and marked with additional tags if required.



## 34.6 - STEM EXPORTS

Stem exports are covered fully in the *Export* chapter. A stem export creates one file per track, starting at the beginning of the session. Each track can then be imported into another DAW.

All data will be lost except the actual audio/MIDI (no plugins, no automation). This is one of the most common methods of interchange because it works between all DAWs.



---

CHAPTER  
**EIGHT**

---

**35 - FILE AND SESSION MANAGEMENT AND COMPATIBILITY**



## 35.1 - SESSION TEMPLATES

Session templates are a way to store the setup of a session for future use. They do not store any *audio* data but can store:

- The number of tracks and busses, along with their names
- The plugins present on each track or bus (if any)
- All I/O connections

### 9.1 Creating a Session Template

The Session > Save Template shows a dialog asking for the name of the new template, and a description.

### 9.2 Using a Session Template

In the *New Session dialog*, a panel lists the different template (factory and user-created).

### 9.3 Managing Templates

Both Session templates and Track Templates can be managed through the Manage Templates window, which can perform the following actions:

- Renaming a template
- Removing one
- Adding/modifying its description
- Exporting the templates (e.g. to be used in another Ardour instance)
- Importing templates (from e.g. another Ardour instance).

See also *Adding Tracks and Busses* for information on templates for individual tracks or busses.

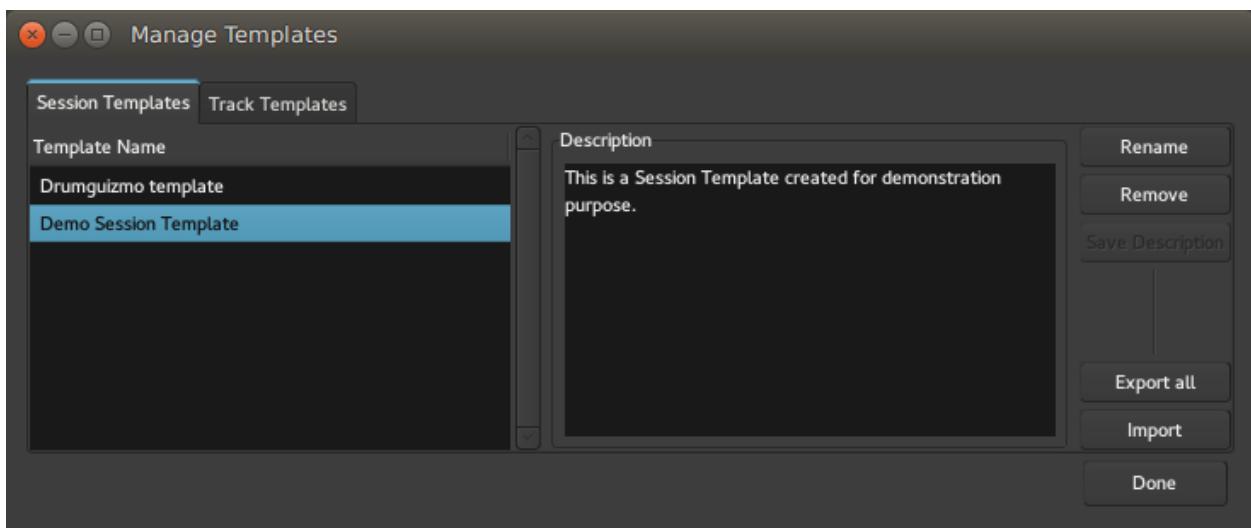


Fig. 1: The Manage Templates window

## 35.2 - SNAPSHOTS

A snapshot is a backup of the current state of a session. It differs from a simple save by allowing branching. It is a “frozen” version of the session at a certain point in time.

For example, creating a snapshot before changing the entire arrangement of a piece, or drastically altering the signal processing provides a reference to come back to, should that not work out.

This is accomplished by using either of the Session > Snapshot menus. A small dialog will appear, allowing to enter a name for the snapshot. The default name is based on the current date and time.

The difference between the two snapshot menus is:

Snapshot (& keep working on current version)...	Saves a snapshot of the session, but keeps the current session active, i.e. any subsequent Session > Save will overwrite the original session, and the snapshot will remain unchanged.
Snapshot (& switch to new version)...	Saves a snapshot of the session, and uses this snapshot as the current active session, i.e. any subsequent Session > Save will overwrite the snapshot, and the original session will remain unchanged.

Any number of snapshots can be created.

Creating a snapshot does **not** modify the session, nor does it save the session. Instead, it saves an alternate version of the session, within the session folder. The snapshot shares all data present in the session.

### 10.1 Switching to a Snapshot

Switching to an existing snapshot is done by navigating the *Snapshot List* and clicking the the name of the desired snapshot. Ardour will switch to the snapshot, and, if there are unsaved changes in the current session, offer to save them.

### 10.2 Starting Ardour With a Snapshot

Since a snapshot is just another session file stored within the session folder, that “version” can be chosen when loading an existing session. The browser in the “Open Session” dialog will show an expander arrow for sessions that have more than one session file (i.e. snapshots) present. Clicking on it shows the list, and then clicking on the name of the snapshot loads it.



## 35.3 - CLEANING UP SESSIONS

Recording and editing any serious session might leave the session with some unused or misplaced files here and there. Ardour can help deal with this clutter thanks to the tools located in the Session > Clean-up menu.

### 11.1 Bring all media into session folder

When *importing media files*, if the Copy files to session has not been checked, Ardour uses the source file from its original destination, which can help avoiding file duplication. Nevertheless, when the session needs to be archived or transferred to another computer, moving the session folder will not move those *external* files as they are not in the folder, as seen in *Backup and sharing of sessions*.

Using the Bring all media into session folder menu ensures that all media files used in the session are located inside the session's folder, hence avoiding any missing files when copied.

### 11.2 Reset Peak Files

Ardour represents audio waveforms with peak files, that are graphical images generated from the sound files. This generation can be time and CPU consuming, so it uses a cache of the generated images to speed up the display process. To watch for files modification, Ardour relies on the file-modification time. If an external file is embedded in the session and that file changes, but the system-clock is skewed or it is stored on an external USB disk (VFAT), Ardour can't know the change happened, and will still use its deprecated peak files.

Using the Reset Peak Files menu allows to reset this cache, which frees up disk space, and forces the re-creation of the peak files used in the session. It can prove useful if some waveforms are not used anymore, or if a graphical or time glitch happens.

### 11.3 Clean-up Unused Sources...

Recording usually leaves a lot of unused takes behind, be it in midi or audio form, that can clutter the Region List, and eat up a lot of hard drive space. While its generally a good practice to keep as many things as possible while recording, when transferring or archiving the session, some clean up can help a lot in reducing the sessions clutter and size.

Selecting Clean-up Unused Sources... will force Ardour to detect those unused waveforms by looking for unused regions, and (through a prompt) for unused playlists. The media files will not be destroyed, though. At this stage, they are just copied in a particular place of the session path (namely, in the `dead sounds/` sub-folder).

## 11.4 Flush Wastebasket

Although Ardour is a *non-destructive* audio-editor, it allows for a very careful destruction of unused media materials. This function is closely linked to the previous one. When the unused sources have been cleaned up and quarantined, the Flush Wastebasket menu will allow for their physical destruction.

As a safeguarding mechanism though, Flushing the wastebasket is impossible in the same working session as the Cleaning up of unused sources: the user needs to close the session and reload it before flushing. It allows to test the playback of the session and ensure both that Ardour did not commit any mistake (unlikely, but better safe than sorry), and that the user is absolutely sure of what he does.

All media destroyed this way is not sent to the system's *trash can* but permanently deleted. If a file is mistakenly destroyed this way, the user will have to rely on data recovery techniques to try getting it back.

## 35.4 - INTERCHANGE WITH OTHER DAWs

It has never been particularly easy to move sessions or projects from one DAW to another. There are two interchange standards that have reasonably widespread support:

- OMF (Open Media Framework), also known as OMFI. Developed and controlled by Avid, never standardized
- AAF (Advanced Authoring Format). Developed by a consortium of media-related corporations.

In practice both of these standards have such complex and/or incomplete specifications that different DAWs support them only partially, differently, or not at all.

### 12.1 Transferring an Ardour session from / to another DAW

To move a session from another DAW to Ardour, or from Ardour to another DAW, there are two basic choices:

- *Stem exports*
- *Using AATranslator*

### 12.2 Importing ProTools® files

Ardour provides a basic import tool for ProTools® sessions, in the Session > Import PT Session menu.

Though incomplete, this import is intended for ptf and ptx files. Protools® 5, 8, 9, 10, 11 have been tested with varying degrees of success. (versions 6 and 7 are not supported).

The elements of the files that are imported are:

- Audio regions data & position
- MIDI notes (fused to a unique region).

### 12.3 Using AATranslator

AATranslator is a Windows application that can convert sessions/projects from many different DAWs into other formats. At the present time (December 2016), it can read and write Ardour 2.X sessions, and can read Ardour 3 sessions.

The program runs very well on Linux using [Wine](#) (a Windows environment for Linux). There are equivalent solutions for running Windows applications on OS X, but we have no experience with them at this time. Ardour users have reported great results using AATranslator on Ardour 2.X sessions.

The [AATranslator website](#) has full details on supported formats and DAWs. The list includes ProTools, Live, Reaper, OMF, AAF and many more.

AATranslator is closed-source, non-free software (as of this writing, June 2017, the cost is 59 USD for the “Standard” version, and 199 USD for the “Enhanced” version).

---

CHAPTER  
**THIRTEEN**

---

**PART V - PLAYBACK & RECORDING**



---

CHAPTER  
**FOURTEEN**

---

**36 - PLAYING BACK TRACK MATERIAL**



## 36.1 - CONTROLLING PLAYBACK

The playhead is a red vertical line that indicates the current position of playback.

### 15.1 Positioning the Playhead

#### 15.1.1 Positioning the playhead at the current pointer position

Pressing P will set the playhead to the current position of the mouse pointer, if it is within the editor track area.

#### 15.1.2 Positioning the playhead on the timeline

A Left click anywhere on the *Ruler* will move the playhead to that position.

#### 15.1.3 Positioning the playhead with the transport clocks

Clicking on either the primary or secondary transport clock and *editing their value* moves the playhead to a specific position.

#### 15.1.4 Positioning the playhead at a marker

Right clicking on the marker and selecting either Locate to Here or Play from Here will place the playhead at the marker's position.

Alternatively, placing the mouse pointer on the marker and pressing P sets the playhead precisely on the marker location.

### 15.2 Looping the Transport

When the *loop transport* button is pressed, the playhead will jump the start of the loop range, and continue to the end of that range before returning to the start and repeating.

While looping, a light green area is displayed in the Ruler over the tracks to show the loop range.

By default, looping is bound to the l key.



Fig. 1: The playhead

---

CHAPTER  
**SIXTEEN**

---

## 36.2 - USING KEY BINDINGS

Ardour has many available commands for playback control that can be bound to keys. Many of them have *default bindings*. Some of the most used are found below.

Those keybindings are shown in the corresponding menus. Memorizing at least the most frequently used can be a great time saver.

Space	switch between playback and stop.
Home	Move playhead to session start marker
End	Move playhead to session end marker
→	Playhead to Next Grid
←	Playhead to Previous Grid
0	Move playhead to start of the timeline
space	Start recording
space	Stop and forget capture



---

CHAPTER  
**SEVENTEEN**

---

**37 - AUDIO RECORDING**



---

**CHAPTER  
EIGHTEEN**

---

## **37.1 - MONITORING**

When recording, it is important that performers hear themselves, and any pre-recorded tracks they are performing with.

Audio recorders typically allow monitoring (i.e. listening to) the input signal of all tracks that are armed for recording, and playing back the unarmed tracks.



---

CHAPTER  
**NINETEEN**

---

## **37.2 - LATENCY CONSIDERATIONS**

In the days of analog tape recording, the routing of monitor signals was performed with relays and other analog audio switching devices. Digital recorders have the same feature, but may impart some *latency* (delay) between the time a noise is made and the time that it will come back from the recorder.

The latency of *any* conversion from analog to digital and back to analog is about 1.5–2 ms. Some musicians claim that even the basic A/D/A conversion time is objectionable. However even acoustic instruments such as the piano can have approximately 3 ms of latency, due to the time the sound takes to travel from the instrument to the musician's ears. Latency below 5 ms should be suitable for a professional recording setup. Because 2 ms are already used in the A/D/A process, extremely low buffer sizes must be used in the workstation I/O setup to keep the overall latency below 5ms. Not all *computer audio systems* are able to work reliably at such low buffer sizes.

For this reason it is sometimes best to route the monitor signal through an external mixing console while recording, an approach taken by most if not all professional recording studios. Many computer I/O devices have a hardware mixer built in which can route the monitor signal “around” the computer, avoiding the system latency.

In either case, the monitoring hardware may be digital or analog. And in the digital case there will still be the A-D-A conversion latency of 1–2 ms.



## 37.3 - MONITOR SIGNAL FLOW

There are three basic ways to approach monitoring:

### 20.1 External Monitoring

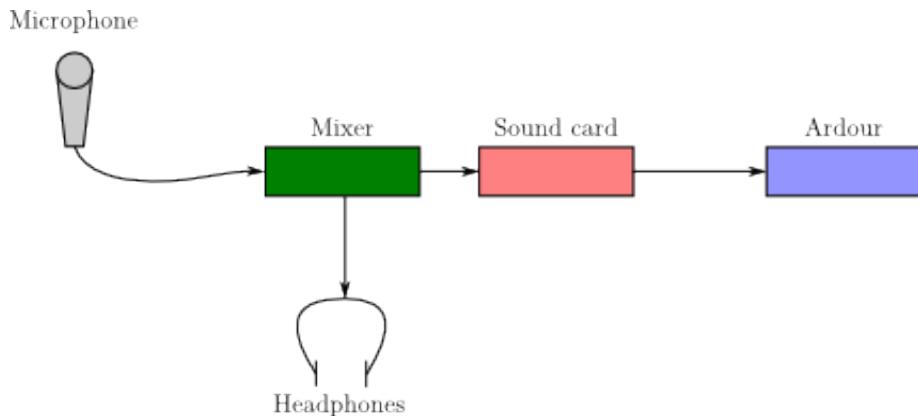


Fig. 1: External monitoring

When using external monitoring, Ardour plays no role in monitoring at all. Perhaps the recording set-up has an external mixer which can be used to set up monitor mixes, or perhaps the sound-card being used has a “listen to the input” feature. This approach yields zero or near-zero latency. On the other hand it requires external hardware, and the monitoring settings are less flexible and not saved with the session.

### 20.2 JACK-Based Hardware Monitoring

Some sound cards have the ability to mix signals from their inputs to their outputs with very low or even zero latency, a feature called hardware monitoring. Furthermore, on some cards this function can be controlled by JACK. This is a nice arrangement, if the sound card supports it, as it combines the convenience of having the monitoring controlled by Ardour with the low latency operation of doing it externally.

### 20.3 Software Monitoring

With the software monitoring approach, all monitoring is performed by Ardour—it makes track inputs available at track outputs, governed by various controls. This approach will almost always have more routing

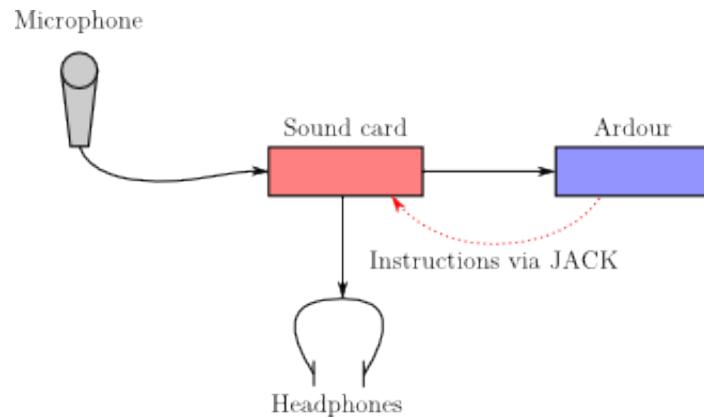


Fig. 2: Hardware Monitoring

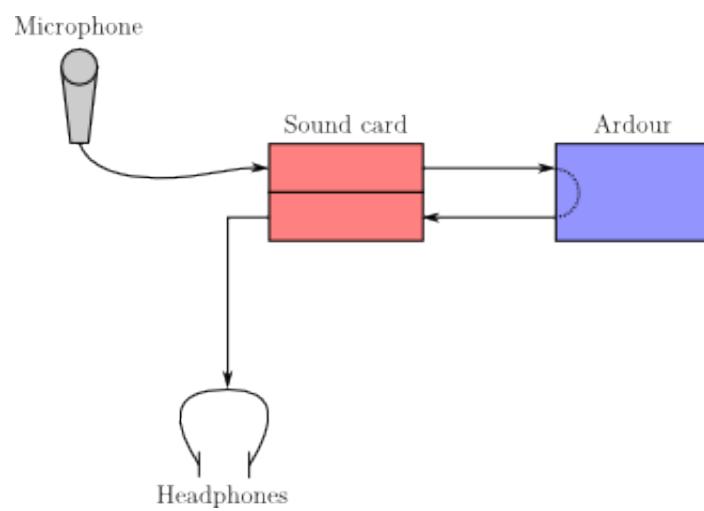


Fig. 3: Software Monitoring

flexibility than JACK-based monitoring. The disadvantage is that there will be some latency between the input and the output, which depends for the most part on the JACK buffer size that is being used.



---

CHAPTER  
**TWENTYONE**

---

**38 - PUNCH RECORDING MODES**



---

CHAPTER  
**TWENTYTWO**

---

## **38.1 - WORKING WITH MARKERS**

It is very useful to be able to tag different locations in a session for later use when editing and mixing. Ardour supports both locations, which define specific positions in time, and ranges which define a start and end position in time.

In addition to the standard location markers, there are three kinds of special markers:

- CD markers are locations that are restricted to legal CD sector boundaries. They can be used to add track index markers to compact disc images.
- The Loop range defines the start and end points for Looping.
- The punch range defines the in and out points for punch recording.



---

CHAPTER  
**TWENTYTHREE**

---

## 38.2 - CREATING LOCATION MARKERS

Location Markers appear in the *Locations Markers ruler* at the top of the timeline. The green begin and end markers, that define the length of the session, appear automatically, but yellow custom markers can be created at any position in a session.

There are multiple ways to create custom markers at the current playhead position:

- using the keyboard shortcut (default is Num-, the Enter key on the numeric keypad)
- using the Transport > Markers > Add Mark from Playhead menu.

Adding a marker at an arbitrary location on the timeline (i.e. not at the playhead position) can also be done either by:

- right clicking in the Location Marker ruler, and selecting New location marker
- using the *Ranges & Marks List*, clicking New Marker and using the *clock widget* to set its position.



### **38.3 - CREATING RANGE MARKERS**

Range markers are essentially two *location markers* that appear in the *Range Markers ruler*, and are grouped together to mark the beginning and end of a section in the timeline.

#### **24.1 Creating a Range on the timeline**

Creating a new range can be done by right clicking on the Range Markers ruler at the top of the timeline, then selecting New Range. Two markers with the same name and opposing arrows will appear along the ruler.

It is also possible to create range markers from a selected range or region in the Editor window, or to using the Ranges & Marks List in the Editor list.

#### **24.2 Editing a Range**

Both markers of a range can be independently moved along the timeline by clicking and dragging them to the desired location.

They can also be moved together by dragging one marker.

See *Moving markers* for more information.



---

CHAPTER  
**TWENTYFIVE**

---

## **38.4 - MOVING MARKERS**

### **25.1 Single marker**

Left-clicking and dragging moves a single marker to a new location on the timeline.

### **25.2 Multiple markers**

It is possible to move multiple markers by the same distance. Left-clicking each discreet marker, or Left-clicking the first and last markers of a range of markers selects them, then dragging one to a new location will move all selected markers together.

The markers are bounded by the zero point on the timeline. In other words, the first marker in the selection cannot move to the left of zero on the timeline.

### **25.3 Both ends of a range marker**

By left-dragging either end of the range marker, the other end will move by the same distance.



---

CHAPTER  
**TWENTYSIX**

---

## **38.5 - THE LOOP RANGE**

The loop range is a special range that defines the start and end points for loop play, which can be enabled in the transport bar.

It can be defined via the timeline or the *Ranges & Marks list*.



---

CHAPTER  
**TWENTYSEVEN**

---

## 38.6 - MARKER CONTEXT MENU

Right-clicking a marker in the timeline opens the marker context menu:

Locate to Here	Move the playhead to this marker's position.
Play from Here	start playback from this marker's position.
Move Mark to Playhead	Move this marker to the current playhead position.
Create Range to Next Marker	Create a range marker between this location and the next one along on the timeline.
Hide	Hide this marker from the view. It can be made visible again from the Window > Locations window or the ' <b>Ranges &amp; Marks list &lt;#the-ranges-and-marks-lists &gt;'</b> .
Rename	Change the name of the marker.
Lock	If this is ticked, it will be impossible to drag the marker's position; useful if you want to prevent accidental movements.
Glue to Bars and Beats	If this is ticked, the marker will maintain its position in bars and beats even if there are changes in tempo and meter.
Remove	Removes the marker.

There are also a few options in Transport > Active Mark. These options apply to the currently selected location marker, and move it to a nearby region boundary, region sync point, or to the playhead or mouse.



---

CHAPTER  
**TWENTYEIGHT**

---

## **38.7 - PUNCH RANGE**

The punch range is a special range used to define where recording will start and/or stop during a punch. It can be defined on the timeline or in the *Ranges & Marks* list.



---

CHAPTER  
**TWENTYNINE**

---

**PART VI - EDITING**



## 39 - NAVIGATING THE EDITOR

Navigating the Editor window is obviously a very frequent operation. Ardour sticks with a lot of the usual conventions in this regard, to allow for a quick learning. As those operations are so common, it is worth taking the time to learn most of the keyboard and mouse shortcuts in order for these to become fast and natural.

The keyboard shortcuts can, as always, be edited, so the defaults are shown here.

### 30.1 Scrolling

Scrolling can be done on-canvas, or with the *Summary*.

#### 30.1.1 On Canvas

Action	Mouse	Keyboard
Scrolling up		↑
Scrolling down		↓
Scrolling up one page		
Scrolling down one page		
Scrolling left		
Scrolling right		

Moving the playhead outside the view may scroll the screen accordingly, so using ← or →, while not *scrolling* per se, will result in scrolling if Transport > Follow playhead is checked. This is also true with the *Navigation Timeline*, and anything that moves the Playhead.

#### 30.1.2 In the Summary

Clicking and dragging in the Summary will scroll the view left and right. If the screen view is clicked (the white rectangle) and dragged, the view can also be scrolled vertically.

Additionally, on the left of the Summary, the two < and > arrows buttons allow to scroll one screen either left or right, while at the right of the Summary, the two ↑ and ↓ arrows buttons allow to scroll one screen either up or down.

## 30.2 Zooming

Zooming (on time) can be done on-canvas (which will always be centered around the mouse cursor), with the Summary, or with the *Zoom Controls*.

### 30.2.1 On Canvas

Zooming in	
Zooming out	

### 30.2.2 In the Summary

Resizing the screen view in the Summary (the white rectangle) changes the zoom accordingly.

### 30.2.3 With the Zoom Controls

With the Zoom Focus set, the – and + buttons will zoom out or in around this focus. The [ ] button zooms to the whole session as defined by the start and end markers.

These controls are bound to the keyboard – and = respectively by default.

## 30.3 Height of the tracks

Changing the height of the tracks results in more or less tracks on screen. This can be done on canvas, with the Summary or with the Zoom Controls.

### 30.3.1 On canvas

Using – or + while hovering over a track reduces or enhances its height, i.e. zooms on the hovered track, regardless of the selection.

The F key resizes the tracks so that only the selected one(s) are displayed. If some unselected tracks are in-between those selected, their *visibility* will be toggled off.

### 30.3.2 In the Summary

Resizing the screen view in the Summary (the white rectangle) changes the number of tracks displayed (hence their heights) accordingly. It behaves more like a zoom as the relative height of the tracks are kept.

### 30.3.3 With the Zoom Controls

The three rightmost buttons of the Zoom Control bar, while not zoom buttons, act upon the height of the tracks:

- The first selector directly selects how many tracks are currently on screen.

- The second one reduces the height of the selected track(s). If none are selected, all the tracks are affected, while maintaining (as long as it is possible) their relative heights.
- The third one enlarges the tracks, and is the counterpart of the previous one.



---

CHAPTER  
**THIRTYONE**

---

**40 - EDITING BASICS**



## 40.1 - WORKING WITH REGIONS

### 32.1 Working With Regions

Regions are the basic elements of editing and composing in Ardour. In most cases, a region represents a single contiguous section of one or more media files. Regions are defined by a fixed set of attributes:

- the audio or MIDI source file(s) they represent,
- an offset (the “start point”) in the audio or MIDI file(s), and
- a length.

When placed into a playlist, they gain additional attributes:

- a position along the timeline, and
- a layer.

There are other attributes as well, but they do not *define* the region. Things to know about regions:

#### 32.1.1 Regions Are Cheap

By themselves, regions consume very little in terms of computer’s resources. Each region requires a small amount of memory, and represents a rather small amount of CPU work if placed into an active track. So, multiplying regions creation whenever needed should not be much of an issue CPU wise.

#### 32.1.2 Regions Are Not Files

Although a region can represent an entire audio file, they are never equivalent to an audio file. Most regions represent just parts of an audio file(s) on disk, and removing a region from a track has nothing to do with removing the audio file(s) from the disk (the Destroy operation, one of Ardour’s few destructive operations, can affect this). Changing the length of a region has no effect on the audio file(s) on disk. Splitting and copying regions does not alter the audio file in any way, nor does it create new audio files (only recording, and the Export, Bounce and Reverse operations create new audio files).



## 40.2 - REGION NAMING

Region names are initially derived from either:

- the name of the track for which they were recorded, or
- the name of the embedded/imported file they represent.

### 33.1 Whole File Region Names

These are not audio files, but regions that represent the full extent of an audio file. Every time a new recording is done, or a new file is imported to the session, a new region is created that represents the entire audio file. This region will have the name of the track/playlist/original file, followed by a “-“, then a number plus a dot and then a number.

For recorded regions, the number will increase each time a new recording is made. So, for example, if there is a track called Didgeridoo, the first recorded whole file region for that playlist will be called Didgeridoo-1. The next one will be Didgeridoo-2 and so on.

For imported regions, the region name will be based on the original file name, but with any final suffix (e.g. “.wav” or “.aiff”) removed.

Normally, whole file regions are not inserted into tracks or playlists, but regions derived from them are. The whole-file versions live in the *Editor’s region list* where they act as an organizing mechanism for regions that are derived from them.

### 33.2 Normal Region Names

When a region is inserted into a track and playlist, its initial name will end in a version number, such as .1. For a recorded region, if the whole file region was Hang drum-1, then the region in the track will appear with the name Hang drum-1.1. For an imported region, if the whole file region was Bach:Invention3, then the region in the track will appear with the name Bach:Invention3.1.

### 33.3 Copied Region Names

Duplicating or splitting a region creates new region(s) that are based on the same original files. Hence, they share the same base name (in the example above, Hang drum-1), but their version number will be incremented each time. Duplicating Hang drum-1.4 by left dragging it will create a new region called Hang drum-1.5. Splitting Hang drum-1.5 by hitting the S key will remove the Hang drum-1.5 region and create two shorter regions named Hang drum-1.6 and Hang drum-1.7.

## 33.4 Renaming Regions

Regions can be renamed at any time using the region context menu : right click > *name\_of\_the\_region* > Rename... . The new name does not need to have a version number in it (in fact, it probably should not). Ardour will add a version number in the future if needed (e.g. if the region is copied or sliced).

---

CHAPTER  
**THIRTYFOUR**

---

## 40.3 - CORRESPONDING REGIONS SELECTION

*Track Groups* have a property titled Select which, if enabled, cause Ardour to propagate a region selection in one track of a group to the corresponding regions of the other tracks in that group.

This can be particularly useful when an instrument has been recorded using multiple microphones (e.g. a drum kit): by enabling the Select property for the group, selecting a region in one of the tracks, Ardour will select the corresponding region in every other track of the group, which in turn means that a subsequent edit operation will affect all the tracks together.

### 34.1 How Ardour Decides Which Regions are “Corresponding”

Regions in different tracks are considered to be corresponding for the purposes of sharing selection if they satisfy *all* the following criteria:

1. each region starts at the same offset within its source file,
2. each region is located at the same position on the timeline, and
3. each region has the same length.

### 34.2 Overlap Correspondence

Sometimes, the rules outlined above are too strict to get Ardour to consider regions as corresponding. Regions may have been trimmed to slightly different lengths, or positioned slightly differently, and this will cause Ardour to not select regions in other grouped tracks.

In this case, changing Edit > Preferences > Editor > Regions in active edit groups are edited together: to whenever they overlap in time will allow regions in different tracks to be considered equivalent for the purposes of selection if they overlap. This is much more flexible and will cover almost all of the cases that the fixed rules above might make cumbersome.



---

CHAPTER  
**THIRTYFIVE**

---

## 40.4 - REGION CONTEXT MENU

In the editor window, right clicking (context clicking) on a region displays a menu with track and region operations. The menu begins with the name of the region, or Selected Regions if multiple regions are selected. Selecting it will display the *Region menu* for operations on this(these) region(s).

Some items may not be in the exact same order as in the main menu.

If there is more than one region layered at the point clicked, the menu will also contain a Choose Top item. This dialog allows to select which region should be on the top layer. See *Adjusting Region Layering* for more details.

Below these items is the rest of the *Track Context Menu*, which provides access to track-level operations.



## 40.5 - COMMON REGION EDIT OPERATIONS

This section covers a set of region editing operations that are likely to be used often while working on a session. Depending on work habits (and experience of other DAWs), some of these operations will be critical while others are used only rarely.

All of these operations can be carried out from the keyboard (see *Default Keyboard Shortcuts* for a list). Equivalent operations can be performed with the mouse in most cases.

Some of these operations make use of *the edit point/range* and *affect specific regions*.

Spot (Align)	Move selected regions to the edit point.
Split	Split selected regions at the edit point.
Trim Start	Adjust the start of selected regions to the edit point (or as close as possible).
Trim End	Adjust the end of selected regions to the edit point (or as close as possible).
Duplicate	Make a copy of each selected region and position it immediately after the original.
Crop	Truncate selected regions to the edit range.
Separate	Split selected regions at both ends of the edit range.
Set Fade In	Adjust selected audio regions' fade in to end at the edit point.
Set Fade Out	Adjust selected audio regions' fade out to end at the edit point.
Toggle Fade In	Turn selected audio regions' fade in on or off.
Toggle Fade Out	Turn selected audio regions' fade out on or off.
Play Region	Play session from the start of the earliest selected region.
Zoom To Region	Zoom horizontally so that the selected regions span the editor track view.
Set Sync Point	Set the sync point of all selected regions to the edit point.
Insert	Inserts the currently selected regions in the Region List at the edit point.



---

CHAPTER  
**THIRTYSEVEN**

---

## 40.6 - COPY REGIONS

### 37.1 Copy a Single Region

Copying a region is done using the *Grab mouse mode*, by moving the mouse pointer into the region and left-dragging. Ardour creates a new region and follows the mouse pointer as it moves. See [Move Regions](#) for more details on moving the copied region.

### 37.2 Copy Multiple Regions

Copying multiple regions requires them to be selected before copying. Then left-dragging one of the selected regions will copy the regions as they move. The copied regions will keep their positions relative to each other.

### 37.3 Fixed-Time Copying

Copying region(s) to other track(s) while keeping the copies at the same exact position on the timeline as the originals is done by simply using a Middle-drag instead.



---

CHAPTER  
**THIRTYEIGHT**

---

## 40.7 - MOVE REGIONS WITH THE MOUSE

Moving or copying a region is done using the *Grab mouse mode*, or the Smart mode with the pointer in the lower half of the region to begin a move or *copy* operation.

With the pointer in the region, using a Left-drag will make the region follow the pointer as it is moved around. By default, the region can move freely along the timeline.

To move a region from one track to another, the move must be started as described above, but the pointer should end in the desired track. The region will follow the pointer.

If some other kinds of tracks are visible, the region will remain where it is as the pointer moves across them, and will then jump to the new track. This serves as a visual reminder that an audio region cannot be dragged into an automation track or a bus, for example.

### 38.1 Move Multiple Regions

In order to move multiple regions, they should be selected before moving. Then Left-dragging one of the selected regions will move all the regions, keeping their positions relative to each other.

### 38.2 Fixed-Time Motion

Moving region(s) to other track(s) while keeping its at the same exact position on the timeline is done by simply using a Middle-drag instead.



---

CHAPTER  
**THIRTYNINE**

---

## 40.8 - ALIGN (SPOT) REGIONS

Aligning regions (sometimes called “spotting”) means moving one or more regions based on a defined location, which in Ardour is always the *edit point*. An alignment operation moves the region(s) so that some part of the region is positioned at the edit point. Available alignment commands include:

Align Region starts a	Selected region(s) are moved so that their start is located at the current edit point
Align Region ends a	Selected region(s) are moved so that the end is located at the current edit point
Align Region sync points a	Selected region(s) are moved so that their sync point is located at the current edit point
Align Region starts relative a	Selected region(s) are moved so that the start of the earliest region is located at the current edit point, and all others maintain their relative position relative to that region



---

CHAPTER  
**FORTY**

---

**41 - EDIT MODE AND TOOLS**



## 41.1 - EDITING CLOCKS

### 41.1 Clock Modes

Every clock in Ardour has four different, selectable clock modes. Each mode displays time using different units. The clock mode can be changed by Right-clicking on the clock and selecting the desired mode from the menu. Some clocks are entirely independent of any other clock's mode; others are linked so that changing one changes all clocks in that group. The different modes are:

Time code	Time is shown as SMPTE timecode in Hours:Minutes:Seconds:Frames, measured from the timecode zero point on the timeline (which may not correspond to the session start and/or absolute zero on the timeline, depending on configurable timecode offsets). The frames value is dictated by either the session FPS setting, or, if slaved to an external timecode master, the master's setting. In the transport clocks, the FPS value is shown below the time display, along with an indication of the current timecode source (INT means that Ardour is its own timecode source).
BBT	Time is shown as Bars:Beats:Ticks, indicating musical time measured from the start of the session. The transport clocks show the current tempo in bpm and meter below the time display.
Minutes:Seconds:Millisecond	Time is shown as Hours:Minutes:Seconds.Millisecond s, measured from the absolute start of the <b>Timeline</b> (ignoring the session start and any timecode offsets).
Samples	Time is shown as a sample count from the absolute start of the timeline (ignoring the session start and any timecode offsets). The number of samples per second is given by the current sample rate, and in the transport clocks, this rate is shown below the time display along with any pullup/pulldown adjustment.

### 41.2 Changing clock values with the keyboard

New values for the clock can be typed in after clicking on the relevant clock. Clicking on the clock will show a thin vertical cursor bar just to the right of the next character to be overwritten. Time should be typed in the same order as the current clock mode—if the clock is in Timecode mode, it should be hours, minutes, seconds, frames. So, to change to a time of 12:15:20:15 one would type 12152015. Each number typed will appear in a different color, from right to left, overwriting the existing value. Mid-edit, after typing 3222 the clock might look like this:



Fig. 1: A clock being edited in Ardour

Finishing the edit is done by pressing **Enter** or **Tab**. The **ESC** key allows to exit an edit without changing the clock. If an entry is mis-typed so that the new value would be illegal (for example, resulting in more than

30 frames when Timecode is set to 30 frames per second), the clock will reset at the end of the edit, and move the cursor back to the start to allow for another try.

## 41.3 Avoiding the mouse entirely

There is a shortcut available to edit the transport clocks entirely without the mouse. It can be found in Window > Key Bindings > Transport > Focus On Clock. If bound to a key (÷ on the numerical keypad is the default), then pressing that key is equivalent to clicking on the primary (left) transport clock, and editing can begin immediately.

## 41.4 Entering Partial Times

One detail of the editing design that is not immediately obvious is that it is possible to enter part of a full time value.

As an example, supposing that the clock is in BBT mode, displaying 024|03|0029, altering the value to the first beat of the current bar can be done by clicking on the clock and typing 010000. Similarly, if it is in Minutes:Seconds mode, displaying 02:03:04.456, getting to exactly 2 hours can be achieved by clicking on the clock and typing 0000000 to reset the minutes, seconds and milliseconds fields.

## 41.5 Entering Delta Times

Values can also be typed into the clock that are intended as a relative change, rather than a new absolute value, by ending the edit by pressing + or - (the ones on any keypad will also work). The plus key will add the entered value to the current value of the clock, minus will subtract it. For example, if the clock is in Samples mode and displays 2917839, moving it back 2000 samples is done by typing 2000 and -, rather than ending with Enter or Tab.

## 41.6 Changing clock values with the mouse

### 41.6.1 Using a scroll wheel

With the mouse pointer over the clock, moving the scroll wheel changes the clock values. Moving the scroll wheel up ( ) increases the value shown on the clock, moving it down ( ) decreases it. The step size is equal to the unit of the field hovered over (seconds, hours, etc.).

### 41.6.2 Dragging the mouse

With the mouse pointer over the clock, pressing the left mouse button and dragging also affects the clocks: dragging upwards increases the value shown on the clock, dragging downwards decreases it, again with a step size equal to the unit of the field where the drag began on.

---

CHAPTER  
**FORTYTWO**

---

## 41.2 - WHICH REGIONS ARE AFFECTED?

This section explains the rules used to decide which regions are affected by editing operations. They don't really have to be understood—hopefully things will Just Work—but it may be useful eventually.

Editing operations in Ardour either operate on a single point in time (Split being the obvious example) or on two points (which can also be considered to be a range of sorts), Separate is a good example of this.

Most operations will operate on the currently selected region(s), but if no regions are selected, the region that the mouse is in will be used instead. Single-point operations will generally pick a set of regions to use based on the following rules:

- If the *Edit Point* is mouse, then
  - if the mouse is over a selected region, or no region, use all selected regions, or
  - if the mouse is over an unselected region, use just that region.
- For all other Edit Points
  - use the selected regions *and* those that are both under the edit position *and* on a selected track, or on a track which is in the same active edit-enabled route group as a selected region.

The rationale here for the two different rules is that the mouse Edit Point is special in that its position indicates both a time and a track; the other edit points (Playhead,Marker) indicate a time only.



## 42 - MAKING SELECTIONS

Many editing operations in Ardour require to first select one or more regions to change them in some way. A single region, or multiple regions can be selected, including regions in different tracks. When a region is selected, it will appear in a darker color than unselected regions.

If a track is a member of a group that is active and has the Select property enabled, then Ardour will attempt to match whatever selections is made in one track across every other track of the group. See [Corresponding Regions Selection](#) for more information on precisely how selections will be propagated to other tracks.

### 43.1 Track Selection

Tracks are selected by clicking on the Track header at the left of the Editor window. Multiple tracks can be selected with Left clicks, or a range of consecutive tracks with Left.

### 43.2 Region Selection

Using the *Grab Mode tool*, left clicking on a region selects it. If *Smart mode* is enabled, the lower half of the region must be clicked.

### 43.3 Deselecting a Region

Still using the *Grab Mode tool*, Left-clicking the region deselects it. If *Smart mode* is enabled, the lower half of the region must be clicked.

Note that a left click simply toggles the selected status of an object, so it can be used to select unselected regions too.

### 43.4 Selecting Multiple Regions in a Track

This can be achieved in different ways:

- Left-clicking each region.
- Dragging a rubberband box from an empty point in a track before the first region to select to a point within or after the last region to select. Using left-drag allows doing it multiple times.
- If the regions are all adjacent to one another, clicking the first region to select, then Left-clicking the last one.

## 43.5 Selecting All Regions in a Track

The Select > Select All In Track option in the context menu (Right click on the track) selects all the regions in the track at once.

See the *Track Context Menu* for more information on other per-track selection operations that are available.

## 43.6 Selecting Multiple Regions Across Different Tracks

This can be achieved by a left-click or Left-click on the regions to select.

## 43.7 Selecting a Region From the Region List

Clicking the name of the region in the *Region List*. selects it. This will do nothing for whole-file regions, since they do not exist anywhere in a playlist or track.

---

CHAPTER  
**FORTYFOUR**

---

**43 - EDITING REGIONS AND SELECTIONS**



## 43.1 - TRIMMING REGIONS

Changing the length of a region is a very common editing operation, often known as trimming. There are several ways to accomplish this with Ardour, and some very useful specialized trimming operations.

### 45.1 Drag-Trimming With the Mouse

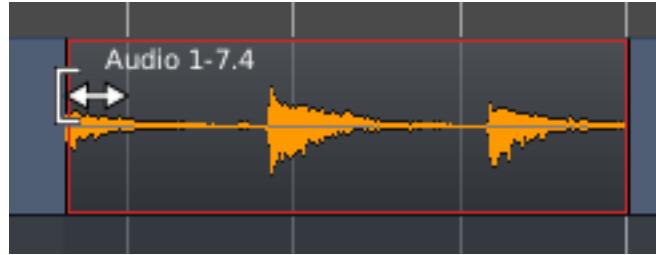


Fig. 1: Trimming region - before and after

In Grab mode, moving the pointer near the beginning or end of the region changes the cursor to indicate that trimming is possible, and the edge of the region can then be Left-dragged in both directions.

Trimming will obey *Snap settings*.

### 45.2 Other Trimming operations

There are several commands for region trimming. Some use the *edit point* to determine where to trim to. Some are not bound to any keys by default (but could be via the *Keybindings Editor*).

These command are both in the Region > Trim main menu (with a region selected) or in the context menu of a region, right click on a region > *Name\_of\_The\_Region* > Trim

Trim Start at Edit Point (j)	Trim selected region(s) start to edit point.
Trim End at Edit Point (k)	Trim selected region(s) end to edit point.
Trim to Loop/Punch	Trim selected region(s) beginning and end to the loop/punch boundaries (if it exists).
Trim to Previous (j)	Trim the start of selected region(s) to the end of the previous region. If the region is too short, it is extended to its maximum to the left.
Trim to Next (k)	Trim the end of selected region(s) to the start of the following region. If the region is too short, it is extended to its maximum to the right.

---

CHAPTER  
**FORTYSIX**

---

## **43.2 - PUSH/PULL TRIMMING**

Normally, when trimming regions by dragging with the mouse, it affects only the selected regions. Their lengths are directly affected by the trim operation, but nothing else is. Sometimes though, when trimming a region that directly adjoins another, the desired result is to move the boundary between the regions and not to make these regions overlap. This requires trimming both regions on either side of the junction, in opposite directions. Push/Pull trim, activated by pressing key before starting the drag, will do just that.

The following pictures show the difference in the results of a normal trim and a push/pull trim:

In the initial situation, before trimming, two adjacent regions are present, the rightmost-one being selected. The simple trim, obtained by dragging the selected region's starting position earlier, overlaps the earlier region. A crossfade has been manually created between them, so their sound will fade from the leftmost region to the rightmost one.

If the same trim is done, but by Left-dragging to turn it into a push-pull trim instead, there is no overlap, and the end of the earlier region has been moved along with the start of the later region, so that they still directly adjoin each other. In effect, it is like doing a simple trim to reduce the leftmost region, then doing a simple trim to extend the rightmost one to fill the gap.



---

CHAPTER  
FORTYSEVEN

---

### 43.3 - STRETCHING

The Stretch Mode tool can be switched to by selecting it in the *Toolbox*, or simply by hitting the T key.

It allows to extend or reduce the duration of a region, optionally maintaining its pitch. This is one of the few operations in Ardour that affect the underlying audio data from a region, even if the original audio is kept safely—no data is lost in the process.

This operation is usually used to fit an audio sequence with a different rhythm into a session, but can be used in a wide area of cases, due to its ability to maintain or alter the pitch.

The Stretch Mode tool is very similar in use to doing a trim in grab mode: the boundary (start or end) is left-clicked and dragged to its wanted position. Notice a timer appearing, showing the new duration of the region using the same *clock mode* as in the *primary transport clock*.

Stretching is a complex operation (phase vocoding), involving resampling, frequency analysis and synthesis. The parameters used to transform the audio data are user tweakable, and exposed to the user as the left mouse button is released:

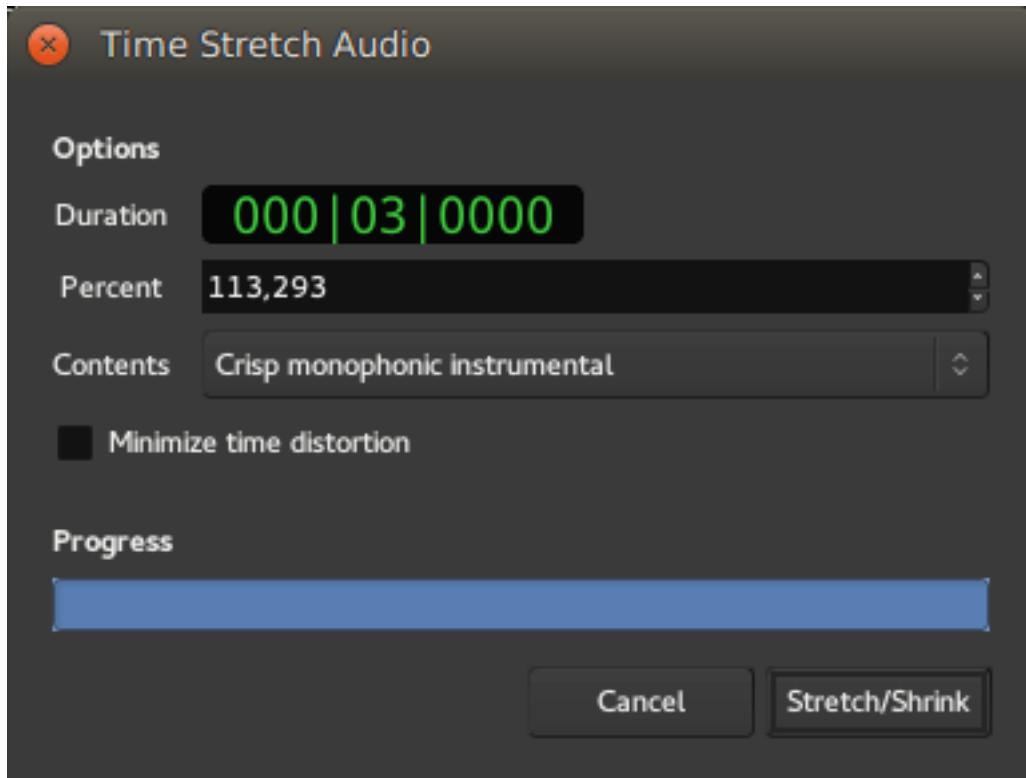


Fig. 1: The Time Stretch Audio window

The Time Stretch Audio window is made of:

Duration	The target duration of the region, expressed using the primary transport clock's mode
Percent	The target duration of the region, expressed as a percentage of the region's original length. Can be either higher than 100% (to expand the region) or lower (to shrink it)
Contents	The type of audio the region is made of. Ardour will fine-tune its algorithm based on this content, see below
Minimize time distortion	Tries to reduce the smearing of the audio created by the phase vocoding process
a Progress bar	showing the operation in progress.

The *Contents* should be selected to best fit the actual content of the region, amongst:

Content

Disable phase resynchronisation at transients

Band-limit phase resync to extreme frequencies

Disable phase locking to peak frequencies

Use longer processing window (actual size may vary)

Use shorter processing window

Mushy

X

X

X

Smooth

X

X

Balanced multitimbral mixture

X

Unpitched percussion with stable notes

X

Crisp monophonic instrumental (*default*)

Unpitched solo percussion

X

X

Resample without preserving pitch

*see below*

While the table above details *how* the different kinds of audio material alter the fine-tuning of the DSP, from an user point of view, the operation often consists in trying different settings and listening to the result.

The best way to start experimenting is to consider the material itself:

- If the material doesn't need its pitch to be preserved, the best choice is *Resample without preserving pitch*
- For drum-type material, the best choice is (depending on the transients crispness, stretching factor...) one of the two *percussion* types
- For melodic mono-tonal material (bass, winds,...), the best (and default) choice is *Crisp monophonic instrumental*
- For multi-tonal material (chords,...), either one of the three first choice, or the default *Crisp*.



## 43.4 - SEPARATE UNDER

### 48.1 Separate Under

When one region is over another, and the lower region has to be cut so that it directly adjoins both ends of the overlapping one, with no overlaps, the Separate Under tool can be a very efficient time-saver. With the upper region selected, the Edit > Separate > Separate Under menu will split the lower region so that it no longer overlaps the upper region at all.



Fig. 1: Region arrangement before and after ‘Separate Under’

If the upper region covers only one end of the lower region, then this operation is equivalent to *Trim to Next* or *Trim to Previous*, depending on which end is covered.

### 48.2 Separate Using Range

A loop or punch range can also be used to slice a region. By using the Edit > Separate > Separate Using Loop/Punch Range, any selected regions that are covered by the range at both ends of the range, or just one if the range only covers part of the region. This makes it easy to generate regions that correspond precisely to a range.



Fig. 2: Region arrangement before and after ‘Separate Using Loop Range’

## 43.5 - STRIP SILENCE FROM AUDIO REGIONS

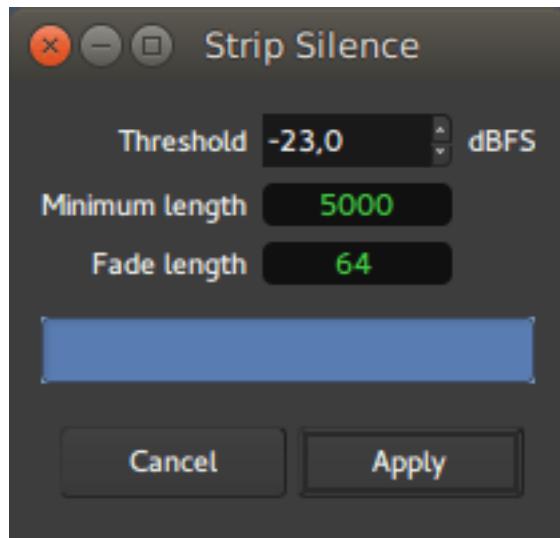


Fig. 1: The Strip Silence window

The Strip Silence tool allows to remove the parts of one or multiple regions that are below a user-defined silence threshold. It does *not* destroy the underlying audio, but trims the regions according to the silence threshold parameter. The edit applies to all selected regions, allowing batch processing.

The window, accessible either through the Region > Edit > Strip Silence menu or right click on a region > *Name\_Of\_TheRegion* > Edit > Strip Silence is made of:

Threshold	The audio level under which the audio is considered silent (in dBFS)
Minimum length	A minimum number of samples for Ardour to create a split. Under this number, the region won't be sliced
Fade length	Ardour adds fades, both in and out, to the trimmed regions, to the created region (so the sliced regions are longer by both the in and out fades duration, expressed in samples)
A progress bar	showing the time Ardour takes to compute the trimming based on the current parameters

Changing any parameter in the window is reflected in the main editor: the silent segments are highlighted and the number and durations of the shortest segments is displayed, helping fine-tune the parameters.

The minimum length for silence can be useful when editing very percussive material and just needing to automatically trim the ends of a region.

## 43.6 - INSERT/REMOVE TIME

### 50.1 Insert Time

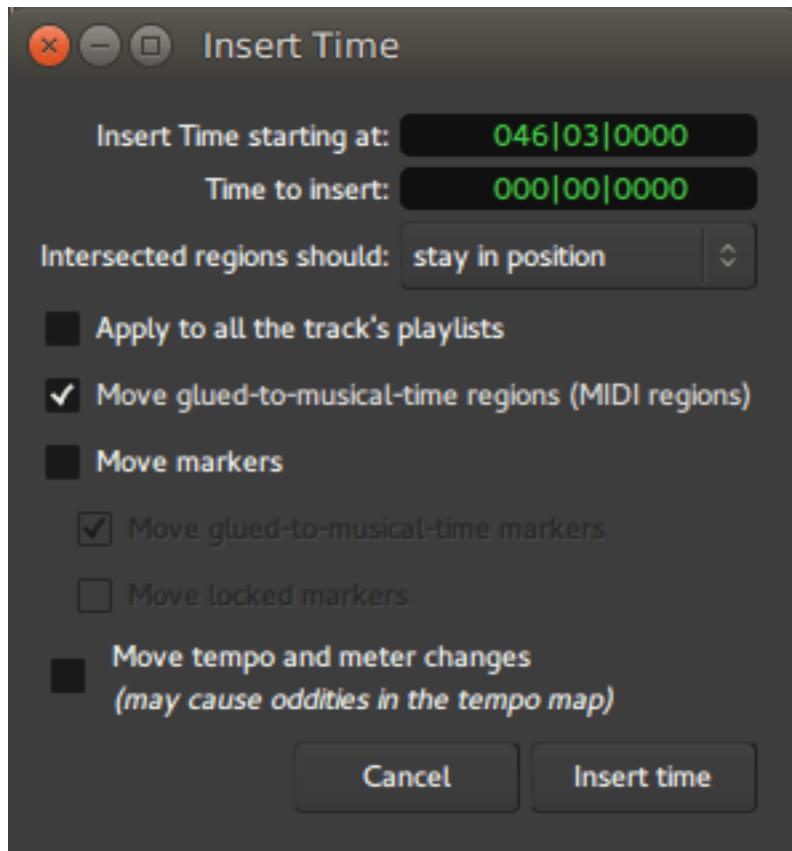


Fig. 1: The Insert Time window

The Insert Time window allows to insert blank space in the timeline, on the selected track(s). It is accessed through the Track > Insert Time menu.

The Insert Time window not only allows to set the time inserted, but also some fine tuning options:

Insert Time starting at:	Sets the point in the session where the time will be added. By default, it is the playhead's position
Time to insert:	Duration of the blank space inserted
Intersected regions should:	A choice as to what happens to regions that exists at the Insert Time set above. See below.
Apply to all the track's playlists	As a track can have multiple <i>playlists</i> , the insertion can happen either only on the active playlist or on all the playlists of this track
Move glued-to-musical-time regions (MIDI regions)	Defines if MIDI regions in selected tracks are affected by the operation
Move markers	As a <i>marker</i> can be locked or glued to bars/beats, this option and the two subjacent ones allow to shift the time position of those markers
Move tempo and meter changes	The <i>tempo and meter</i> markers, that can be used to change the tempo along the session, can also be shifted in the process. Though, moving the tempo markers while e.g. keeping the MIDI regions unaffected can create oddities.

Both the two time fields have a useful context menu, that allows to copy/paste the time and change the display among one of the *clock modes*. The Insert Time field also has two options to Set from Playhead (to get the insertion time from the playhead, which is what happens by default) and Locate to This Time which moves the playhead to the time field value, useful if the field has been manually edited to better visualize the insertion point.

The “Intersected regions should” dropdown allows to select what happens to regions that cross the insertion position:

stay in position ( <i>default</i> )	The crossed regions are not affected by the time insertion. Only regions after the insertion point are moved.
move	The crossed regions are shifted in time.
be split	The crossed regions are split, and the section after the time insertion point is shifted in time.

This last mode allows a *brute force* insertion, creating a blank space in all the selected tracks, and replacing multiple operations.

Note: One interesting option is, if a range selection (created with in *Range Mode*) exists, it is used as the default parameters in the window, instead of the playhead, for both the start time and duration.

## 50.2 Remove Time

The Remove Time window, accessed through the Track > Remove Time menu, is very similar to the previous, and its options are very similar. Only the “*Intersected regions should*” option is not present.

The range selection note above can be especially useful in this context.

## 43.7 - REGION PROPERTIES

### 51.1 Region properties

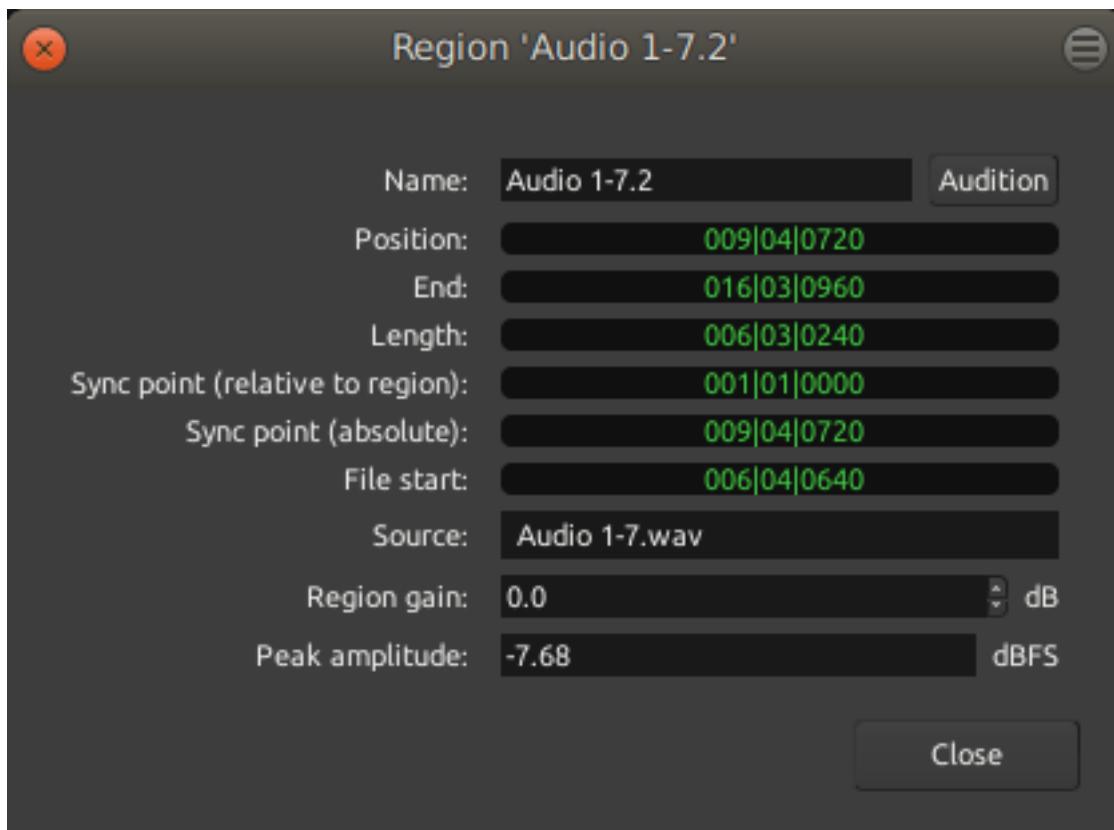


Fig. 1: The Region Properties window

The Region properties window brings information about the selected regions, and allows to fine tune its sequencing. It is accessed through the Region > Properties... menu, or by right clicking the region, *Name\_of\_the\_Region* > Properties....

This window also allows to manually set the different values.

Field

Meaning

Editable

Name:

The *name of the region* in the editor

X

Audition

This button allows to listen to the region and only the region, dry (with no effects, regardless of the processors applied to the track). For MIDI, the default MIDI synth, set in the *Preferences*, is used to audition the region.

Position:

Position in time of the left-hand side of the region

X

End:

Position in time of the right-hand side of the region

X

Length:

Duration of the region (=End–Position)

X

Sync point (relative to region):

Position in time of the *Sync Point*, relative to the beginning of the region. No manual sync point means the sync point is the beginning of the region (=Position), so will show as “0” in time and 001|01|0000 in Bars:Beats notation.

X

Sync point (absolute):

Position in time of the *Sync Point*, relative to the beginning of the session. No manual sync point means the sync point is the beginning of the region, and will be equal to the Position.

X

File start:

Position in time of the beginning of the region relative to the source file start. If the region has not been trimmed on the left, then the regions start is the file start and this value is 0.

Source:

Name of the source audio/MIDI file the region is extracted from. A region can be a part of or a whole audio/MIDI file, and multiple regions can be based on the same source file.

X

Region gain:

(*Audio files only*) Manual gain, in dB, applied constantly to the whole region, regardless of the global track's gain, automation, etc...

X

Peak amplitude:

(*Audio files only*) Maximum level the signal reaches inside the region. Expressed in dBFS (Full Scale), where 0 is the numeric maximum a signal can reach.

All the fields marked as “editable” in the table above allow the user to manually enter a value in the field, to manually set this value.

Right-clicking on a field allow to switch between the different *clock modes*.

The context menu that pops up also allows the relevant fields (Position, End, and both Sync points) to be set from the playhead location, which can be used to e.g. trim a region to the playhead or place a sync point exactly on a beat.



## 44 - FADES AND CROSSFADES

Every Region has a fade-in and fade-out. By default, the region fade is very short, and serves to de-click the transitions at the start and end of the region. By adjusting the regions fade length, a more gradual transition can be accomplished.

### 52.1 Region Fades

Region fades are possible at the beginning and end of all audio regions. In object mode, a grip appears at the top left and top right of an audio region when the cursor hovers over it. Placing the cursor over the top of the grip displays the region fade cursor tip. Clicking and dragging the grip left or right in the timeline adjusts the length of the fade.

### 52.2 Crossfades

Crossfades refer to the behavior of two audio regions transitioning smoothly (mixing) from one to another on the same track. Historically, this was done by splicing two pieces of analog tape together, and this concept was carried forward into digital editing. Each track is a sequence of sound files (regions). If two regions are butted against each other, there needs to be a method to splice them smoothly together. The crossfade allows one region to fade smoothly out, while the next region fades smoothly in, like two pieces of tape that have been cut at an angle, and overlapped.

But Ardour uses a more refined “layered” editing model, and therefore it is possible for multiple regions to be stacked on a single location with arbitrary overlaps between different layers. For this reason, crossfades must be implemented differently. It can’t be assumed that a crossfade is an entity that exists between two regions; instead each region must have its own associated crossfades at each end, and the topmost region must always crossfade down to the underlying region(s), if any.

Ardour solves this problem by putting a crossfade at the beginning and end of every region. The fades of the bottom-most region are first rendered, and then each region is rendered on top of the one below it, with fades at the end of each region providing a crossfade to the region(s) beneath it.

It is important to understand that region fades *are* crossfades. When one region has another region or multiple regions beneath its fade area, then what will be heard is the topmost region fade-out mirrored as a fade-in on the underlying region(s). The grip for the topmost region will allow changing the length and type of the crossfade into the underlying region(s). In this way complicated series of crossfades can be created, and then another region layered atop the others, and faded into a complicated series.

If a region doesn’t have any region(s) under it, then the region is crossfaded to silence; for convenience this is called a “fade” rather than a crossfade.

## 52.3 Fade Shapes

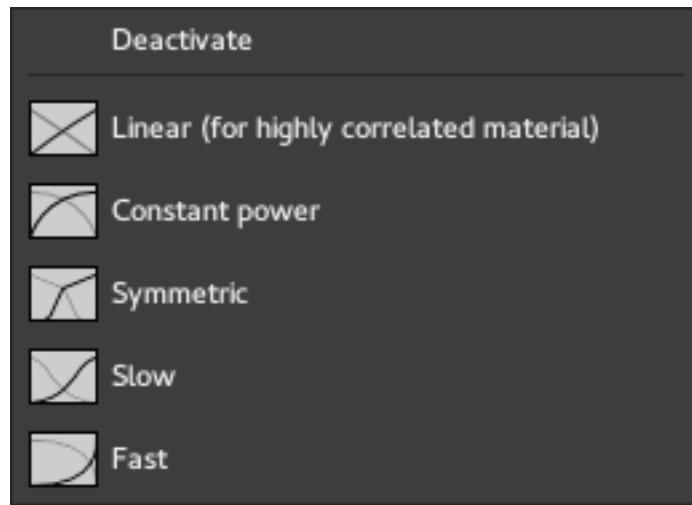


Fig. 1: The fade shape context menu.

To activate/deactivate or change the shape of a region's fadein or fade-out, the cursor has to be hovered over the region fade grip until the cursor tip indicates region fade editing, then right clicked to bring up a context menu. In the context menu is a list of options for the region fade. Activate/Deactivate enables and disables the region fade.

Because each fade is also a crossfade, it has an inverse fade shape for the audio beneath the fade. It is important to know how the shapes differ, and which are most suitable for various editing tasks.

The different types of fades are:

Linear	A simple linear coefficient decrease, and its mathematical inverse. A Linear fade starts attenuating quickly, and then cuts off even more abruptly at lower levels. When used as a crossfade, the signals are each -6dB attenuated at the midpoint. This is the correct crossfade to use with highly-correlated signals for a smooth transition.
Constant Power	The constant power curve starts fading slowly and then cuts off abruptly. When used as a crossfade between 2 audio regions, the signals are symmetrically attenuated, and they each reach -3dB at the midpoint. This is the correct crossfade to use when splicing audio in the general (uncorrelated) case.
Symmetric	The Symmetric fade starts slowly, then attenuates significantly before transitioning to a slower fade-out near the end of the fade. When used as a crossfade, the Symmetric curve is not mathematically correct like the Constant Power or Linear curves, but it provides a slower fade-out at low volumes. This is sometimes useful when editing 2 entire music works together so that the transition is more gradual.
Slow	The Slow curve is a modified linear decibel fade. The initial curve starts more gradually so that it has a less abrupt transition near unity. After that, it sounds like a perfectly smooth fader or knob moved to silence. This shape is excellent as a general-purpose fade-out. When used as a crossfade, the inverse fade curve maintains constant power but is therefore non-symmetric; so its use is limited to those cases where the user finds it appropriate.
Fast	The Fast curve is a linear decibel fade; It sounds like a perfectly smooth fader or knob moved to silence. This shape is excellent as a general-purpose fade-in. When used as a crossfade, the inverse fade curve maintains constant power but is therefore non-symmetric; so its use is limited to those cases where the user finds it appropriate.

Although these fade shapes serve specific purposes, any of the shapes is usable in any situation, so the final decision is mostly an artistic choice.

These fade curves are developed to provide a range of common uses, and are developed with the least possible amount of changes in the “slope” of the line. This provides artefact-free crossfades. Some DAWs provide complicated fade editors with parametric “spline” controls of the fade curves. While it might be interesting to develop a fade curve with a faster cutoff, the mathematical difference between this and simply shortening the fade is vanishingly small; and the amount of effort to shorten the fade is much easier than messing with a crossfade editor dialog.



---

CHAPTER  
FIFTYTHREE

---

## 45 - GAIN ENVELOPES

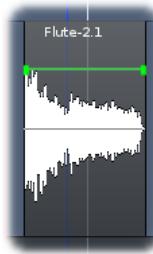


Fig. 1: A gain envelope (in green).

In Ardour, every region has a gain envelope, which is normally hidden. Clicking on the Draw tool will cause all the gain envelopes on all regions to show themselves; these will appear as green lines with square dots (control points) at the beginning and end of each region. The vertical axis represents gain, with the top of the region representing +6dB and the bottom representing approximately -170dB. By default, the line starts and ends at 0dB; the control points can be moved up and down to change the amount of gain at that point.

Gain follows the line between control points continuously during playback, and adjusts the gain for that region accordingly. It is completely automatic, unlike channel *automation*.

### 53.1 Manipulating Gain Envelopes

The default gain curve, by itself, is not very useful; in order to have more control over the shape of the gain envelope it is necessary to add extra control points. Clicking anywhere in the region where there are no existing control points adds a control point to the envelope; it will appear *on the line* at the X-axis of the mouse's current position in the region.

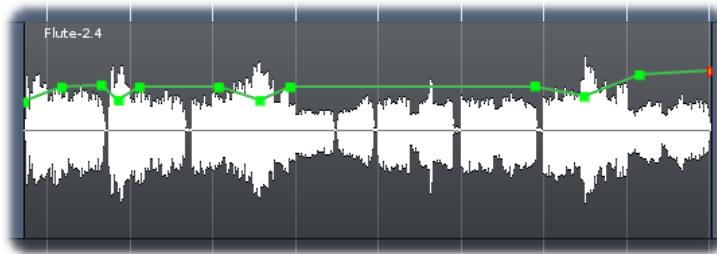


Fig. 2: A more complex gain envelope.

Once added, a control point can be Left clicked and dragged to the desired location. Hovering over a control point will show its current level in dB. Left clicking a control point and pressing Delete, or Right clicking a point deletes it.

---

CHAPTER  
**FIFTYFOUR**

---

**46 - PLAYLISTS**



## 46.1 - UNDERSTANDING PLAYLISTS

A playlist is a list of regions ordered in time. It defines which parts of which source files should be played and when. Playlists are a fairly advanced topic, and can be safely ignored for many types of audio production. However, the use of playlists allows the audio engineer more flexibility for tasks like multiple takes of a single instrument, alternate edits of a given recording, parallel effects such as reverb or compression, and other tasks.

Each audio track in Ardour is really just a mechanism for taking a playlist and generating the audio stream that it represents. As a result, editing a track really means modifying its playlist in some way. Since a playlist is a list of regions, most of the modifications involve manipulating regions: their position, length and so forth. This is covered in the chapter *Working With Regions*.

This page covers some of the things that can be done with playlists as objects in their own right.

### 55.1 Tracks are not Playlists

It is important to understand that a track *is not* a playlist. A track *has* a playlist. A track is a mechanism for generating the audio stream represented by the playlist and passing it through a signal processing pathway. At any point in time, a track has a single playlist associated with it. When the track is used to record, that playlist will have one or more new regions added to it. When the track is used for playback, the contents of the playlist will be heard. The playlist associated with a track can be changed at (almost) any time, and tracks can even share playlists.

Some other DAWs use the term “virtual track” to define a track that isn’t actually playing or doing anything, but can be mapped/assigned to a real track. This concept is functionally identical to Ardour’s playlists. We just like to be little more clear about what is actually happening rather than mixing old and new terminology (“virtual” and “track”), which might be confusing.

### 55.2 Playlists are Cheap

One thing to bear in mind is that playlists are cheap. They do not cost anything in terms of CPU consumption, and they have very minimal efforts on memory use. So generating new playlists whenever needed is recommended. They are not equivalent to tracks, which require extra CPU time and significant memory space, or audio files, which use disk space, or plugins that require extra CPU time. If a playlist is not in use, it occupies a small amount of memory, and nothing more.



## 46.2 - PLAYLIST OPERATIONS

In the *track header* (editor window, left pane) is a button labelled p (for “Playlist”). A click on this button displays the following menu:

(Local Playlists)	Shows all of the playlists associated with this track, and indicates the currently selected playlist
Rename...	Displays a dialog to rename the current playlist
New...	Creates a new empty playlist, and the track switches to the new playlist
New Copy...	Creates a new playlist that is a copy of the current playlist; the track switches to the new playlist
Clear Current	Removes all regions from the current playlist
Select From All...	Displays a playlist browser to manually choose which playlist this track should use. (from this track or another one)

### 56.1 Renaming Playlists

Playlists are created by default with the name of the active playlist, plus a version number, and the first playlist is named after the track with which it is associated. So, the first playlist for a track called “Cowbell” will be called “Cowbell.1”, the next one “Cowbell.2”, etc. This name can be changed at any time, to anything: Ardour does not require playlist names to be unique, although it will make the user’s life easier if they are. Suggested examples of user-assigned names for a playlist might include Lead Guitar, 2nd take, vocals (quiet), and downbeat cuica. These might be different from the associated track names, which for these examples might be Lead Guitar, Vocals and Cuica. The playlist name provides more information because it is about a specific version of the material that may (or may not) end up in the final version of the track.

Using the fact that playlist names are based on the active one with an incremented version number, one can rename a playlist “Cowbell take.1” so that the next playlist created is automatically named “Cowbell take.2” etc. This allows for a quick way to label different takes.

### 56.2 Sharing Playlists

It is entirely possible to share playlists between tracks. The only slightly unusual thing that should be noted when sharing is that edits to the playlist made in one track will magically appear in the other. It is an obvious consequence of sharing. One application of this attribute is parallel processing, described in *Playlist Use Cases*.

To avoid this kind of behaviour, and nevertheless use the same (or substantially the same) playlist on two tracks, the desired playlist must be chosen in the second track, and then the New Copy... button clicked. This generates an independent copy of it for that track, which can then be edited without affecting the original.

## 46.3 - PLAYLIST USECASES

### 57.1 Using Playlists for Parallel Processing

One of the uses of playlists is to apply multiple effects to the same audio stream. For example, applying two different non-linear effects such as distortion or compression to the same audio source (linear effects can be just applied one after the other in the same track) can be done by creating a new track, applying the original track's playlist, and then applying effects to both tracks independently.

The same result could be achieved by feeding the track to multiple busses which then contain the processing, but this increases the overall latency, complicates routing and uses more space in the Mixer window.

### 57.2 Using Playlists for “Takes”

Using Playlists for takes is a good solution when one needs the ability to edit individual takes, and select between them.

Each time a new take is started, a new playlist should be created with  $p > \text{New}$ . Thus, later, any previous or later takes can be selected as desired.

Creating a composite edit from multiple takes, can be achieved either:

- by creating a new track to assemble the final version, and “cherry picking” from the playlists in the original track by copying regions over as required
- by recording each successive take on top of the others in “layers” and then editing them using the layer tools.

### 57.3 Using Playlists for Multi-Language Productions

The same approach as for takes is useful when recording or editing content in multiple versions, such as dubbed movie dialog in several languages: having all versions on the same track allows to apply the same processing, making it easy to switch language before exporting the session.



---

CHAPTER  
FIFTYEIGHT

---

## 47 - RHYTHM FERRET

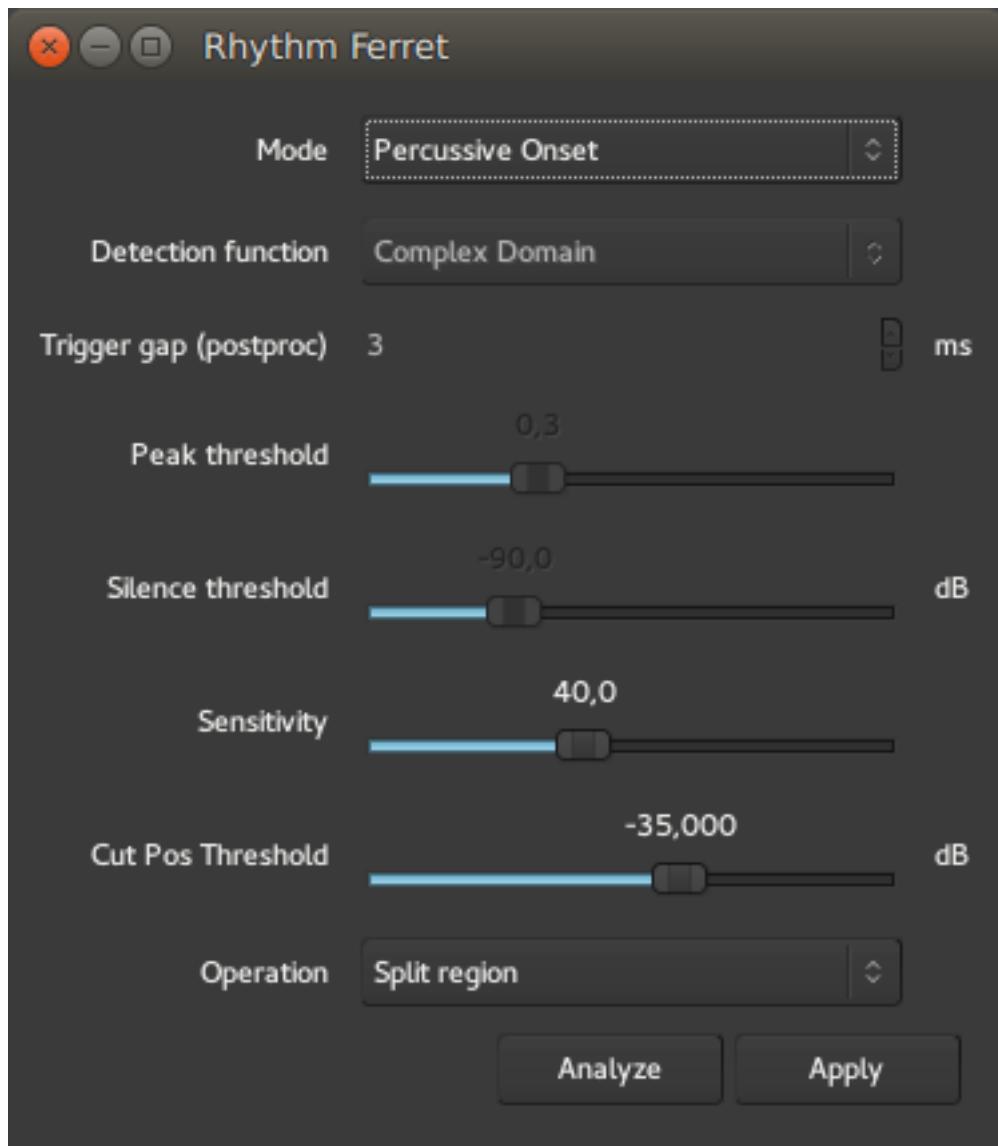


Fig. 1: The Rhythm Ferret window

The Rhythm Ferret is a dedicated tool to speed up the usually labor intensive task of slicing and adjusting a sound region to match a specific time grid. It is especially useful for drum tracks, either to match a different

tempo, or to adjust a slightly out of tempo performance.

It is not limited to this use though, as it supports both percussive and note type detection, and can be used on melodic material too.

## 58.1 Accessing the Rhythm Ferret

The Rhythm Ferret window can be accessed by right clicking any audio region, then *Name\_Of\_The\_Region* > Edit > Rhythm Ferret.

Once the window is open, selecting any region will make it the focus of the Rhythm Ferret's detection, hence allowing to process multiple regions sequentially without reopening the window each time.

The window itself is made of:

- a “mode” selection
- some parameters for this mode
- an operation selection, that for now only allows to Split regions.

## 58.2 The “Mode” selection

As the Rhythm Ferret is able to detect both percussive hits and melodic notes, it is important to choose the best suited mode for the considered material, so that Ardour can perform the detection with the greatest accuracy :

- Percussive Onset will detect the start of each hit based on the sudden change in energy (= volume) of the waveform
- Note Onset will detect the start of each note based on the changes in the frequency domain.

## 58.3 The Percussive Onset mode

In this mode, only two parameters are active:

Sensitivity (%)	The proportion of the samples that must exceed the energy rise threshold in order for an onset to be detected (at frames in which the detection function peaks). This roughly corresponds to how “noisy” a percussive sound must be in order to be detected.
Cut Pos Thresh-old (dB)	The rise in energy amongst a group of samples that is required for that to be counted toward the detection function’s count. This roughly corresponds to how “loud” a percussive sound must be in order to be detected.

As those parameters are very material-related, there is no recipe for a perfect match, and a good peak detection is a matter of adjusting those two parameters by trial and error, and trying using the Analyze button after each try.

Vertical grey markers will appear on the selected region, showing where Ardour detects onsets as per the parameters. These markers can be manually adjusted, see below.

## 58.4 The Note Onset Mode

In the Note Onset mode, more parameters are active:

Detection function	The method used to detect note changes. More on this below.
Trigger gap (postproc) (ms)	Set the minimum inter-onset interval, in milliseconds, i.e. the shortest interval between two consecutive onsets.
Peak threshold	Set the threshold value for the onset peak picking. Lower threshold values imply more onsets detected. Increasing this threshold should reduce the number of incorrect detections.
Silence threshold (dB)	Set the silence threshold, in dB, under which the onset will not be detected. A value of -20.0 would eliminate most onsets but the loudest ones. A value of -90.0 would select all onsets.

The Detection function, used in Note Onset mode to choose the mathematical strategy used to detect the note changes, is user-selectable:

Energy based	This function calculates the local energy of the input spectral frame
Spectral Difference	Spectral difference onset detection function based on Jonathan Foote and Shingo Uchihashi's "The beat spectrum: a new approach to rhythm analysis" (2001)
High-Frequency Content	This method computes the High Frequency Content (HFC) of the input spectral frame. The resulting function is efficient at detecting percussive onsets. Based on Paul Masri's "Computer modeling of Sound for Transformation and Synthesis of Musical Signal" (1996)
Complex Domain	This function uses information both in frequency and in phase to determine changes in the spectral content that might correspond to musical onsets. It is best suited for complex signals such as polyphonic recordings.
Phase Deviation	This function uses information both energy and in phase to determine musical onsets.
Kullback-Liebler	Kulback-Liebler onset detection function based on Stephen Hainsworth and Malcom Macleod's "Onset detection in music audio signals" (2003)
Modified Kullback-Liebler	Modified Kulback-Liebler onset detection function based on Paul Brossier's "Automatic annotation of musical audio for interactive systems" (2006)

Ardour defaults to Complex Domain, which usually gives good result for harmonic material.

## 58.5 Manual adjustment

Using the Rhythm Ferret consists usually in finding the right parameters to split the audio, by adjusting them and clicking the Analyze button. Each time an analysis is run, Ardour erases the previous results, and creates grey markers on the region according to the parameters. Those markers can be manually dragged with the LEFT mouse button to adjust their positions.

Once the markers are suitably placed, the second button in the down hand side of the Rhythm Ferret window allows to Apply the operation. At the moment of writing, only the Split Region is available, which will split the region at the markers.

Those regions can then be manually aligned, or have their sync points set to the closest grid (as per the *Grid settings* in effect), by selecting all the regions, and using the right click then Selected Regions > Position > Snap position to grid.

---

CHAPTER  
**FIFTYNINE**

---

**PART VII - MIDI**



## 48 - MIDI EDITING

Ardour's handling of MIDI editing differs from most other DAWs and MIDI sequencers.

### 60.1 Key features of Ardour MIDI handling

- All editing is done in-place, in-window. There is no separate piano roll window or pane. Edit notes right where they appear.
- MIDI, just like audio, exists in regions. MIDI regions behave like audio regions: they can be moved, trimmed, copied (cloned), or deleted. Ardour allows either editing MIDI (or audio) regions, or MIDI region content (the notes), but never both at the same time. The e key (by default) toggles between region level and note level editing
- Editing note information in Ardour occurs in only a single region. There is no way currently to edit in note data for multiple regions at the same time, so for example notes cannot be selected in several regions and then all deleted, nor can they be copied-and-pasted from one region to another. Region(s), though, can be copy-pasted just as with audio.
- All MIDI I/O is done via JACK for sample accurate timing and maximal efficiency when communicating with external software synthesizers.
- Every MIDI track has its own JACK MIDI port for input; it may have an arbitrary combination of audio and MIDI outputs, depending on the signal processing in the track; the full flexibility of JACK connectivity is present for MIDI just as it is for audio.
- Full automation for MIDI tracks, integrated with the handling of all MIDI CC data for each track.
- Controllers (CC data) can be set to discrete or continuous modes (the latter will interpolate between control points and send additional data).
- There is a Normal and a Percussive mode for note data editing.
- The scroomer is a combination scroll/zoom tool for altering the zoom level and range of visible MIDI data.

### 60.2 Notable Differences

- Fader (volume) control currently operates on transmitted MIDI data, not by sending CC #7.
- All note/data editing is per-region. There are no cross-region operations at this time.
- By default, copying a MIDI region creates a deep link—both regions share the same data source, and edits to the contents of one will affect the other. Breaking this link is done by selecting MIDI > Unlink from other copies from the region context menu, after which the selected region(s) will have their own

copies of *only* the data that they visually display on screen. The region will no longer be trimmable back to its original length after an Unlink operation, and the operation cannot be undone.

---

CHAPTER  
**SIXTYONE**

---

## 49 - CREATING MIDI TRACKS

Creating new MIDI track(s) can be done using the Session > Add Track/Bus menu. In the *Add Track/Bus dialog*, MIDI Tracks must be picked from the combo selector at the upper left.

A track template can be used, by selecting it in the *Configuration* drop down menu.

One singularity of the MIDI track creation in the ability to select right at creation time the instrument that will be used in the track. The instrument in this context is any plugin that will generate audio in response to receiving MIDI.



---

CHAPTER  
**SIXTYTWO**

---

## 50 - CREATING MIDI REGIONS

Although recording MIDI is a common way to create new MIDI regions, it is often desirable to do so as part of editing/arranging.

Create a new MIDI region is as simple as entering draw mode (press “d” or click on the pencil tool) and then Left-clicking in a MIDI track. A region will be created that is one bar long. Alternatively, use the left button to drag-create a new region of the desired length.

Once the region exists, to trim it to a different length, switch back to grabber/object/select mode (press “g” or click on the grabber tool). It can then be *trimmed* to any length desired.

Once a region has been created, *some notes* should be added to it.



---

CHAPTER  
SIXTYTHREE

---

## 51 - ADDING NEW NOTES

### 63.1 Adding new notes

In general, most MIDI editing will probably be done with the mouse in object mode. This allows to select notes, copy, move or delete them and alter their properties (see below). But at some point, *adding* notes to a MIDI region using the mouse will mean dragging with the mouse. Since this would normally be a selection operation if the mouse is in object mode, there needs to be some way to tell Ardour that the user is trying to draw new notes within a MIDI region. Ardour provides two ways to do this. One is to leave the mouse in object mode and Left-drag. The other, useful to enter a lot of notes for a while, is to switch the mouse into Draw Notes mode, which will now interpret any drags and clicks as requests to add a new note. For obvious reasons, Draw Notes mode can not be used while using region-level editing.

So, to summarize:

Selecting, moving, copying, trimming, deleting <i>regions</i>	Note Level Editing disabled, using object, range or other mouse modes
Selecting, moving, copying trimming, deleting <i>notes</i>	Note Level Editing enabled, and using mouse object mode
Adding new notes	enabling “Note Level Editing” and then either <ul style="list-style-type: none"><li>• using mouse object mode and Left-drag, or</li><li>• using mouse draw mode.</li></ul>

Note that there is also *a step entry editor* allowing to enter notes from a virtual keyboard and lots more besides.



---

CHAPTER  
**SIXTYFOUR**

---

## 52 - CHANGING NOTE PROPERTIES

All the details about a selected note can be viewed by context-clicking on it. The dialog that pops up will also allow to modify all the properties of the selected note(s). Individual properties can also be modified more efficiently using the techniques described below.

Moving notes	Right arrow and Left arrow move the selected note(s) early and later in time.
Changing pitch values	↑ increases the pitch of the selected notes. ↓ reduces the pitch of the selected notes. If any of the selected notes are already at the maximum or minimum value, no changes will be made to any of the notes, to preserve relative pitches. This can be overrode with . The default shift distance is one semitone. alters this to one octave.
Changing velocity values	↑ increases the velocity of the selected notes. ↓ reduces the velocity of the selected notes. If any of the selected notes are already at the maximum or minimum value, no changes will be made to any of the notes, to preserve relative velocities. This can be overrode with . Also, pressing v pops up a dialog that will allow to set the absolute velocity value of each selected note. Finally, the scroll wheel will also adjust notes in the same way as the arrow keys (note that like the arrow keys it only affects selected notes, not the note the pointer is over).
Changing channel	Pressing c brings up a dialog that allows to see and alter the MIDI channel of the selected notes. If the selected notes use different channels, they will all be forced to the newly selected channel.
Changing start/end times	, (comma) will alter the start time of the note. . (period) will alter the end time of the note. Both keys will by default make the note longer (either by moving the start earlier or the end later). For the opposite effect, ,/. can be used. The note will be altered by the current grid setting. To change the start/end positions by 1/128th of a beat, the modifier must be added to these shortcuts.
Quantization	q will quantize the selected notes using the current quantize settings. If the quantize settings have not been set for this session yet, the quantize dialog will appear. q will display the quantize dialog to allow to reset the quantize settings, and then quantizes the selected notes. The default quantize settings are: quantize note starts to the current grid setting, no swing, no threshold, full strength.
Step Entry, Quantize etc.	Refer to the <i>Step Entry</i> , <i>Quantizing MIDI</i> , etc. specific pages.



---

CHAPTER  
**SIXTYFIVE**

---

## 53 - HANDLING OVERLAPPING NOTES

Every MIDI note consists of two messages, a NoteOn and a NoteOff. Each one has a note number and a channel (also a velocity, but that isn't relevant here). The MIDI standard stresses that it is invalid to send a second NoteOn for the same note number on the same channel before a NoteOff for the first NoteOn. It is more or less impossible to do this with a physical MIDI controller such as a keyboard, but remarkably easy to trigger when editing in a DAW—simply overlapping two instances of the same note will do it.

Ardour offers many options for how to deal with instances where two instances of the same note overlap. Which one to use is a per-session property and can be modified from Session > Properties > Misc > MIDI Options.

never allow them	Edits that would create note overlaps are not allowed
don't do anything in particular	Ardour leaves overlapping notes alone—the behaviour of a MIDI receiver (plugin or hardware) is undefined
replace any overlapped existing note	When one note is moved to overlap another, remove the one that wasn't being moved
shorten the overlapped existing note	When one note is moved to overlap another, shorten the one that wasn't moved so that there is no overlap
shorten the overlapping new note	When one note is moved to overlap another, shorten the one that was moved so that there is no overlap
replace both overlapping notes with a single note	When one note is moved to overlap another, merge them both to form one (longer) note

Changing the option in use will not retroactively make changes—it will only affect new note overlaps created while the option remains chosen.

Ardour does not check for note overlaps across tracks or even across regions. Dealing with the consequences is up to the user.



---

CHAPTER  
**SIXTYSIX**

---

## 54 - NOTE CUT, COPY AND PASTE

While in *note edit* mode, selected notes can be cut and paste as in most software, that is:

- cut: x
- copy: c
- delete: Delete
- paste: v.

These operations use the same keyboard shortcuts as most editing software does, and as Ardour uses for regions. Obviously, the paste operation only works if a cut or copy operation has happened beforehand.



## 55 - NOTE SELECTION

### 67.1 Selecting/Navigating note-by-note

Tab selects the next note. Tab selects the previous note. Tab or Tab adds the next/previous note to the selection.

### 67.2 Selecting notes with the mouse

While in mouse object mode, any note can be clicked to select it. Once a note has been selected, Left-clicking on another selects all notes between them. Adding or removing a note to/from the selection is done by Left clicking it. Clicking and dragging outside of a note rubberband selects a series of notes.

Three different selection operations are possible while in mouse range mode:

- Vertical drags within the MIDI region will select all notes within the spanned note range.
- Clicks on the piano header of the track (if visible—the track must be tall enough to display it) will select all occurrences of that note.
- Drags on the piano header of the track will select all notes within the spanned note range.

### 67.3 Listening to Selected Notes

If Edit > Preferences > MIDI > Sound MIDI notes as they are selected is enabled, Ardour will send a pair of NoteOn/NoteOff messages through the track, which will typically allow to hear each note as it is selected.



---

CHAPTER  
SIXTYEIGHT

---

## 56 - QUANTIZING MIDI

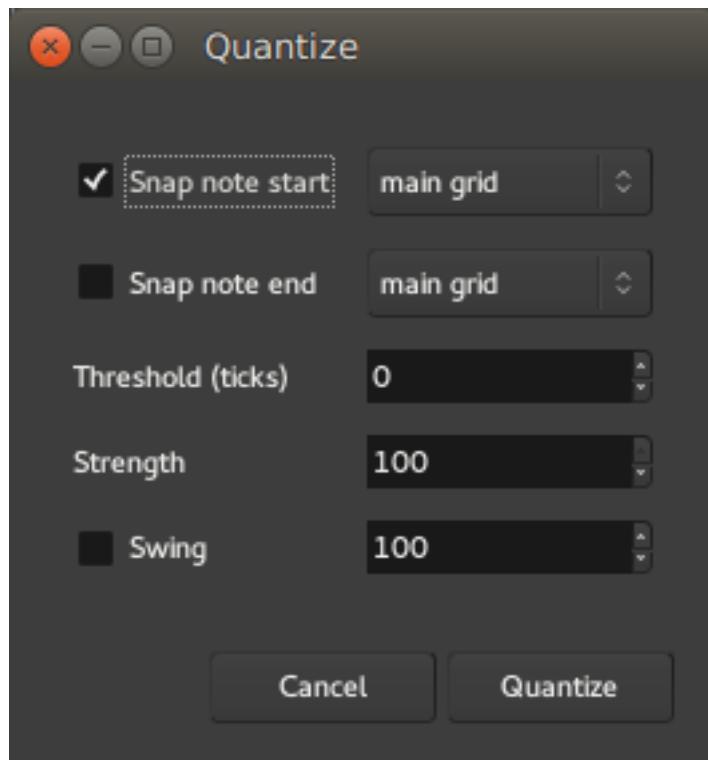


Fig. 1: The Quantize dialog

Quantizing a MIDI region, usually one recorded from a MIDI instrument, consists in perfectly aligning the notes with the grid by shifting the notes positions to the closest grid line. The result is a perfectly timed MIDI region, allowing to correct rhythmically poor performance.

This dialog is accessed via the Region > MIDI > Quantize... while having a MIDI region selected, or by right clicking a MIDI region, *Name\_Of\_The\_Region* > MIDI > Quantize... or with the default 5 shortcut and includes:

Snap note start	If checked, the start of the notes will be aligned to the grid as defined in the following combo-box (see below)
Snap note end	If checked, the end of the notes will be aligned to the grid as defined in the following combo-box (see below)
Threshold (ticks)	Defines how close from a grid point a note must be in order to be quantized. Notes farther than this number of <i>ticks</i> will not be affected.
Strength	Defines how close to its new position the note must be moved, as a percentage of the nominal distance (allowing for a non-perfect quantization, i.e. just making the performance rhythmically better without giving it a machine-generated feel)
Swing	Applies a <i>swing</i> to the midi notes, i.e. delays every 2nd note by this amount, to e.g. simulate a groovy drummer

The grid selection combo boxes allow a choice between the current *main grid*, or many beat subdivisions.

Both note start and note end can be selected at once, resulting in a 2-pass quantization: the note starts are aligned to the grid (with or without the swing and Strength parameters), then their ends are aligned.

The swing is a value between 0 and 130, and is relative to the user-selected grid type: every note which is considered a second note (i.e. close enough to an odd grid line as per the threshold value) will be delayed by this number of ticks.

---

CHAPTER  
**SIXTYNINE**

---

## 57 - STEP ENTRY

Editing MIDI can be a tedious task. Ardour allows using a connected MIDI device like a music keyboard or pad controller, or use the mouse. A third option, providing fine-grain control, precision and speed of entry comes from using a custom note entry dialog.

The step entry dialog is accessed via a right click context menu on the rec-enable button, because step entry is related to *recording* MIDI data. Step editing and recording MIDI via the track's MIDI port cannot happen simultaneously.



Fig. 1: Ardour's Step Entry dialog

The dialog (quite closely modelled on Logic's) contains:

- Chord entry switch (successive notes are stacked in a chord until it is released)
- Note length selectors
- Triplet toggle
- Normal, single, double and triple dotted note selectors
- Sustain button
- Buttons to:
  - Insert a rest of the current selected note duration
  - Insert a rest of the current grid step size
  - Move back to the last inserted note
  - Move forward to the next beat, or bar
  - Move forward to the edit point
- Dynamics controls from pianississimo to fortississimo
- Channel selector
- Explicit numerical velocity selector, for more precise control than the dynamics selectors offer
- Octave selector

- Buttons to add bank or program change events
- a full 10 octave virtual keyboard

More or less all actions in the step entry dialog can be driven directly from the keyboard, so that moving back and forth from the keyboard to the mouse is not necessary even for complex data insertion.

The default key bindings for this (configured in `step_editing.bindings`) are:

grave	octave 0
1 to 9	octave 1 to 9
0	octave 10
F1	note length whole
F2	note length half
F3	note length third
F4 to F8	note length quarter to sixtyfourth
a	insert C
w	insert C
s	insert D
e	insert D
d	insert E
f	insert F
t	insert F
g	insert G
y	insert G
h	insert A
u	insert A
j	insert B
Tab	insert rest
Primary Tab	insert snap rest
BackSpace	back
z	note velocity
x	note velocity
c	note velocity
v	note velocity
b	note velocity
n	note velocity
m	note velocity
comma	note velocity
Up	next note velocity
Down	prev note velocity
Primary Up	next note length
Primary Down	prev note length
apostrophe	toggle triplet
period	toggle dotted
Primary period	no dotted
bar	toggle chord

## 58 - PATCH CHANGE



Fig. 1: A patch change in a MIDI region

A patch change is Ardour’s description for a combination of MIDI program change and bank select messages, that (typically) instruct a synthesizer or sampler to select a different sound to use on a particular channel.

Patch changes are shown within MIDI regions as small rectangles or flags, with a vertical line showing where in the region (hence “when”) this patch change happens.

### 70.1 Inserting Patch Changes

To insert a patch change, the *edit point* should be located where the patch change should be (within an existing MIDI region). When right clicking, and from the MIDI region’s context menu, selecting MIDI > Insert Patch Change, a dialog appears allowing to set the bank and program values.

### 70.2 Modifying Patch Changes

Context-clicking on a patch change will bring up the same dialog that was used to create it, allowing to modify the program and/or bank numbers.

The mouse wheel can also be used: / on the patch change will alter the program number, / will modify the bank number.

### 70.3 Moving Patch Changes

Just Left-dragging the patch change moves it around.

## 70.4 Removing Patch Changes

Pressing Del with the mouse pointer into the rectangular area, or using the delete mouse button operation will remove the patch change (the operation can be undone).

## 70.5 Names for Patch Numbers: MIDNAM files

MIDNAM files assign human-readable names to the “coordinates” (MSB, LSB, pc) of instruments and controls of MIDI-devices. A number of MIDNAM files come already pre-bundled with Ardour. Should this not be the case for your device, you can add your own (see below).

### 70.5.1 Selecting a device

For the proper names to show up in the “Patch Selector”-dialog, you have to assign a device to your current track. To do so, hover the lower border of the tracks header (the mouse-cursor will change to a “resize-cursor”) and expand it. You’ll see dropdown menus. Select your device in the menu.

### 70.5.2 Adding a custom MIDNAM-file

MIDNAM-files are XML-Files. You can edit them using your favorite text-editor. When doing so, please ensure to change the “Model” of the device, as Ardour will only load each model once (i.e. it will skip files, if there are clashes).

After you have done modifications to a file, it is a good idea to validate it. This can be done using the tool `xmllint` as shown below:

```
$ xmllint --valid --noout myfile.midnam
$ wget http://www.midi.org/dtds/MIDINameDocument10.dtd
$ xmllint --dtdvalid MIDINameDocument10.dtd myfile.midnam
```

Once you are satisfied with your file, you have to put it at a location where Ardour picks it up. The best place would be the (hidden) directory *Ardour configuration directory* subdirectory *patchfiles*. in your home-folder. Should the sub-directory *patchfiles* not exist yet, just create it. The path and file-names are case-sensitive. The file should end with “*.midnam*”.

After restarting Ardour, hit the small Log-button in the upper right corner of the main window. It should say something like (this is Linux, Macos or Windows will be different):

```
[INFO]: Loading 3 MIDI patches from /home/username/.config/ardour5/patchfiles
```

The added device should now show up in the dropdown mentioned in the previous paragraph.

Should the MIDNAM-file be useful for the general public, it would be nice to share it: Fork the Ardour-project on [GitHub](#) by hitting the “Fork”-Button. Go to the *patchfiles*-directory (and read the README).

You can upload the file using the Web-Interface. Be sure to select “*Create a new branch for this commit and start a pull request*”.

## **59 - INDEPENDENT AND DEPENDENT MIDI REGION COPIES**

When copying a MIDI region, Ardour has to decide whether to make the copy refer to the same data as the original or not. If it does refer to the same data, then editing either the copy or the original will affect the both of them. If it refers to an independent copy of the data then each one can be edited without affecting the other.

### **71.1 Changing dependent/independent copying for the entire session**

Session > Properties > Misc > MIDI region copies are independent can be used to control the default behaviour when making a copy of a MIDI region.

When enabled, every new copy of a MIDI region results in a copy being made of the MIDI data used by the region, and the new copy of the region will refer to that data.

When disabled, every new copy of a MIDI region will refer to the same MIDI data, and thus editing any copy will change the contents of all of them.

Changing the status of this option has no effect on the existing dependent/independent status of existing region copies.

### **71.2 Making an existing copy of a MIDI region independent**

Right clicking on the MIDI region to be independent then selecting MIDI > Unlink From Other Copies makes it independent: the copy is now using its own version of the data, and edits to the copy will affect only the copy. Other copies will continue to share data.

Note that the copied data only covers the extent of the region when the copy is made. If the region was already trimmed and then a copy is made, an independent copy will have no access to data that is earlier or later than the bounds of the region it was copied from. Put differently, making an independent copy of a trimmed MIDI region only retains the visible part of it.



---

CHAPTER  
SEVENTYTWO

---

## 60 - TRANSPOSING MIDI



Fig. 1: The Transpose dialog

A whole region, or multiple regions, can be transposed at once, with the help of the Transpose MIDI dialog, accessed by right clicking a region > *name\_of\_the\_midi\_region* > MIDI > Transpose... .

This very simple dialog allows to choose either a number of semitones to add or subtract to all the notes inside the region(s), and/or for more significant changes, octaves (12 semitones).



---

CHAPTER  
SEVENTYTHREE

---

## 61 - AUTOMATING MIDI-PITCH BENDING AND AFTERTOUCH

Adding pitch bending or aftertouch can add a lot of subtlety to an otherwise plain sounding midi region and help humanize it.

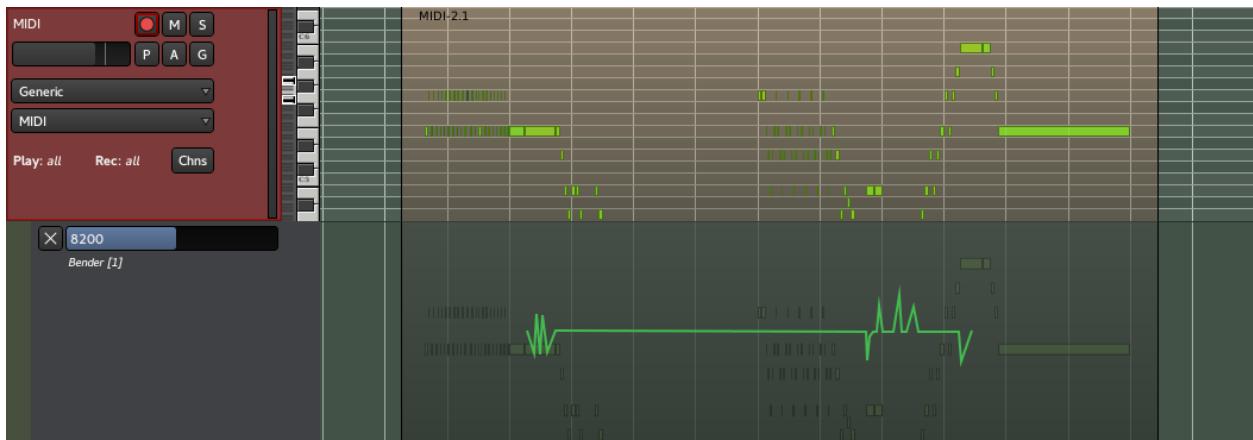


Fig. 1: Automation: pitch bending

Pitch bending and aftertouch both work the same way, through automation, by right clicking the MIDI track's header > Automation > Bender (or Pressure) > the channel to bend.

Using the Draw tool, as for all the automation, allows to create a gradual change from one drawn point to another. A line in the center produces no change to the pitch, while a line above the center will bend the pitch to a higher note (up to 4 semitones) and a line going under the middle will bend the pitch to a lower note.

The values can be anything between 0 (-4 semitones) to 16383 (+4 semitones). No automation or a value of 8192 means no pitch shifting.

Aftertouch works very similarly, though the values are between 0 and 127. It should be noted that aftertouch differs from velocity, as aftertouch allows to slightly change the timbre or create a vibrato, while the velocity sets the power with which the note is played (e.g. on a keyboard, the key is hit).



## 62 - TRANSFORMING MIDI-MATHEMATICAL OPERATIONS

Considering the numerical nature of MIDI events, it can be tempting to apply mathematical transformations to our MIDI regions by using mathematical operations. Ardour makes it very easy and powerful with the Transform tool.

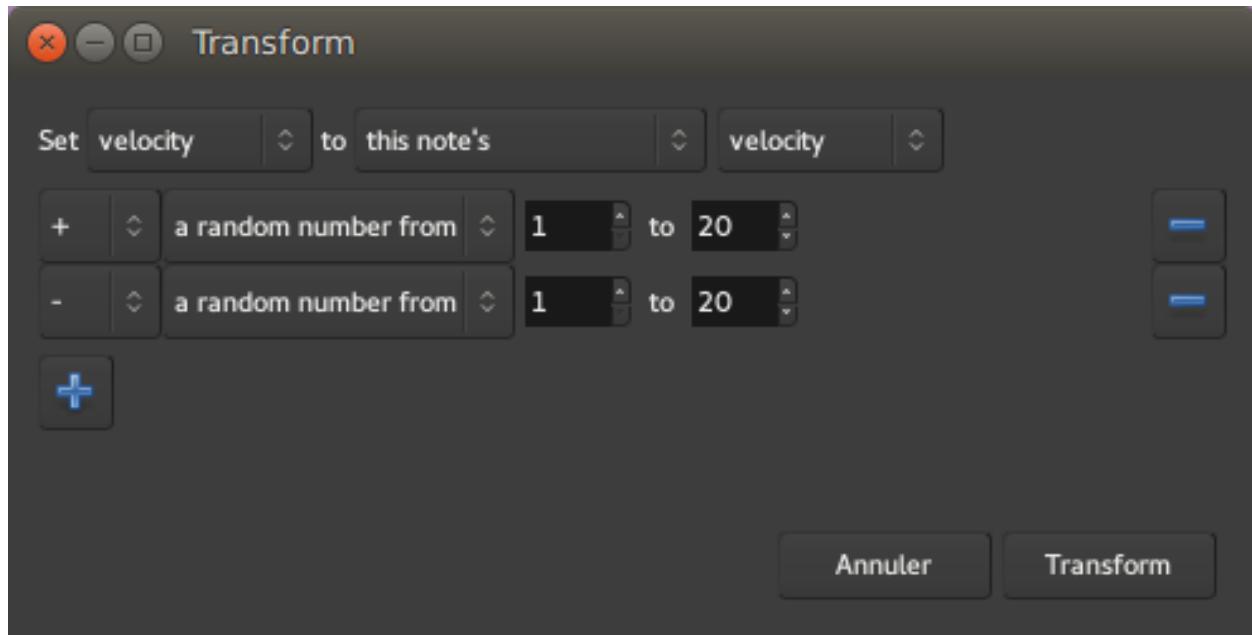


Fig. 1: MIDI transformation

To access the Transform tool, right click the MIDI region > *name\_of\_the\_region* > MIDI > Transform...

First, select the property you want to modify in the 'Set' field, then change the target value using the 2 following fields. If you want to add more operands, click the "+" sign to create new lines. You can remove a superfluous line using the "-" sign on the right of the newly created line.

In the picture above, the Transform tool has been used to add a bit of humanisation, by slightly changing the velocity of each note of the region, by a random number between -19 and +19 from its original velocity. So three operations are applied:

- Set velocity to this note's velocity
- – a random number from 1 to 20
- – a random number from 1 to 20

Each note will trigger a calculation of its own, so its velocity will be increased by a random number between 1 and 20, then decreased by a random number between 1 and 20.

The properties that can be computed are:

- note number (e.g. C2 is note number 24, C#2 is 25 and *so on*)
- velocity (the global intensity of the note, between 0 and 127)
- start time (in beats)
- length (in beats)
- channel

and the calculation may be based on the following properties:

- this note's
- the previous note's
- this note's index (number of the note, i.e. the first one is 0, the second is 1, etc.)
- exactly (for a constant value, between 1 and 127)
- a random number from *lower* to *higher* (*lower* and *higher* being constant values between 1 and 127)
- equal steps from *lower* to *higher* (*lower* and *higher* being constant values between 1 and 127)

The mathematical operators can be:

- – (addition)
- – (subtraction)
- \* (multiplication)
- / (euclidian division)
- mod (remainder of the euclidian division).

All these operations can be very handy, as long as there is a mathematical way to achieve the targeted goal. Beware though of odd “border cases”: division by zero (which does nothing), using the note's index and forgetting it starts at 0 and not 1, etc.

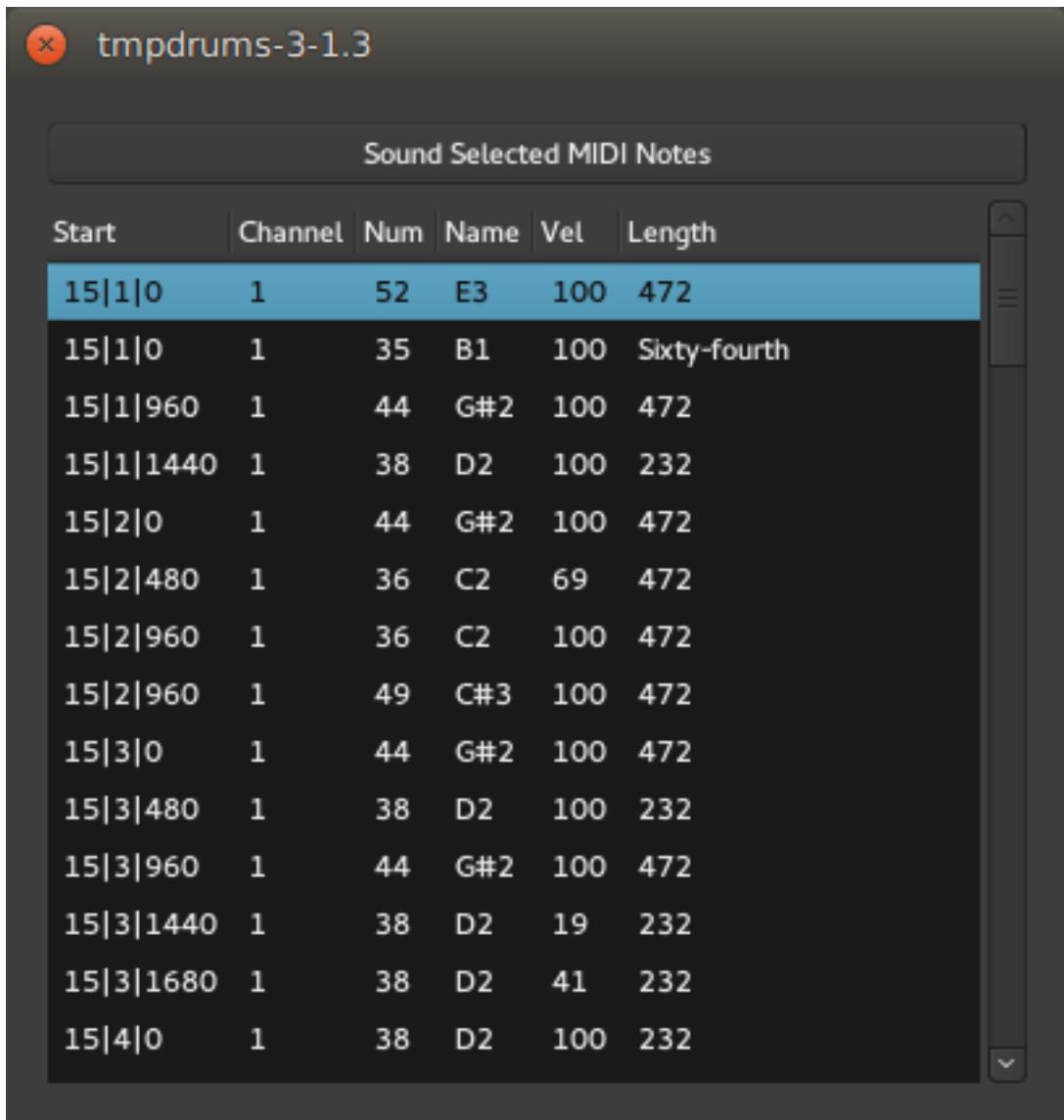
Very interesting results can nevertheless be created, like humanizing (randomizing the velocity, start time and duration of all the notes), creating arpeggios, automating tedious tasks, transposing, etc.

---

CHAPTER  
SEVENTYFIVE

---

## 63 - MIDI LIST EDITOR



The screenshot shows a software window titled "tmpdrums-3-1.3". The main area is a table titled "Sound Selected MIDI Notes" with columns: Start, Channel, Num, Name, Vel, and Length. The first row is highlighted in blue. The data in the table is as follows:

Start	Channel	Num	Name	Vel	Length
15 1 0	1	52	E3	100	472
15 1 0	1	35	B1	100	Sixty-fourth
15 1 960	1	44	G#2	100	472
15 1 1440	1	38	D2	100	232
15 2 0	1	44	G#2	100	472
15 2 480	1	36	C2	69	472
15 2 960	1	36	C2	100	472
15 2 960	1	49	C#3	100	472
15 3 0	1	44	G#2	100	472
15 3 480	1	38	D2	100	232
15 3 960	1	44	G#2	100	472
15 3 1440	1	38	D2	19	232
15 3 1680	1	38	D2	41	232
15 4 0	1	38	D2	100	232

Fig. 1: The MIDI List Editor window

The List Editor is a way to look at the MIDI data of a region, not graphically as they are displayed in the Editor, but in a tabular form. This way of seeing the MIDI data allows for a quicker “debugging” of a MIDI

region, and for a fast *non-graphical* (i.e. no mouse involved) editing. This list has a vertical flow, i.e. the first events (in time) are on the top of the window, and the latest are at its bottom.

It is accessed by selecting the Region > MIDI > List Editor... menu while having one MIDI region selected, or by Right clicking the MIDI region and choosing *Name\_Of\_TheRegion* > MIDI > List Editor...

The window displays the following MIDI data:

Start	the timestamp of the start of the note
Chan- nel	the MIDI channel of the event
Num	The <i>MIDI number</i> of the note
Name	The MIDI name of the note, made of its English name and octave (e.g. “C4”)
Vel	the velocity of the note (i.e. its intensity, between 0 and 127)
Length	duration of the note, either expressed as a number (in ticks, related to the tempo) or as a text (fraction of a beat, also related to the tempo)

At the top of the window is a Sound Selected MIDI Notes, which is a toggle button allowing to listen to a note as it is selected.

Each value can be manually modified, by Left clicking it. The Name field is related to the Number one, and cannot be edited. To change a note, its number must be changed, which will be reflected in the Name field.

---

CHAPTER  
SEVENTYSIX

---

## 64 - MIDI TRACER

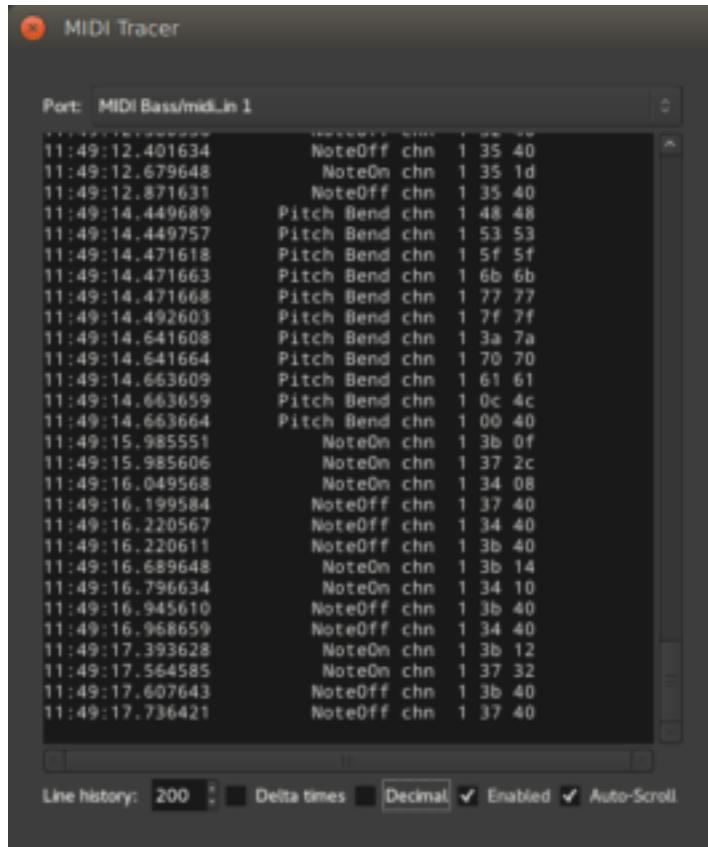


Fig. 1: The MIDI Tracer window

The MIDI Tracer is similar to the *MIDI List Editor*, in the way that it displays MIDI information as a tabular text view, and has a vertical flow, i.e. the events follow a top to bottom time order.

Its use is different though, as it is *not* bound to a specific region or track: the MIDI that is monitored is any global input or output Ardour presents to the system. It is hence a very useful option to monitor a MIDI port, be it an external controller/device or the in/output of any track.

It is accessed by selecting the Window > MIDI Tracer menu.

The window is made of:

Port	a list of all the MIDI ports Ardour presents to the system. They are both internal and external and are the same ports Ardour presents to JACK, if enabled.
the events list	where all the events for this port are listed, see below
Line history	how many lines should be kept in the events list
Delta times	if checked, shows the times as the duration since the last event, instead of the <i>absolute time</i>
Decimal	if checked, shows the MIDI data as decimal values instead of the original hexadecimal values
Enabled	if checked, the events are displayed in the events list, otherwise stops the logging
Auto-Scroll	if checked, the events list scrolls as new events are logged, allowing to keep the newest events on screen

The events list displays the events as columns:

time of the event	either absolute or relative, based on the Delta times checkbox
MIDI status (event type)	what midi events happened (e.g. Note On, Note Off, Pitch Bend, ...)
MIDI channel	in which MIDI channel did the event happen
MIDI data bytes (event parameters)	parameters of the event, e.g. for a Note On: what note was it, and which was the note's velocity

Note: The MIDI Tracer can lists all kind of MIDI events, “audio” ones, but also *scene automation* or *timecodes* ones.

---

CHAPTER  
**SEVENTYSEVEN**

---

**65 - MIDI RECORDING**



## 66 - MIDI SCENE AUTOMATION

Ardour is capable of being used to both record and deliver MIDI “scene” automation. These are MIDI messages typically used to switch presets or “scenes” on a variety of external equipment (or software), including lighting and other audio/video tools. A common use case is to automatically change presets between songs or to change lighting conditions based on a specific position on the timeline.

Each change from one scene to another is represented by a marker in the “Marker” bar.

Technically, scene changes are delivered as a combination of bank and program change MIDI messages. MIDI allows for 16384 banks, each with 128 programs.

### 78.1 Recording Scene Changes

Ardour has a dedicated MIDI port named “Scene In”. Recording scene changes can be done by connecting this port to whatever source(s) of MIDI scene (bank/program change) messages should be recorded.

Whenever the global record enable button is engaged and Ardour’s transport is rolling, a new marker will be created for each scene change message received via the “Scene In” port.

If two different scene changes are received within a certain time period, only the later one will be recorded as a new marker. The default threshold for this is 1 millisecond.

If a scene change message is received while the playhead is close to an existing marker with an associated scene change, the recording process will alter the scene change in the existing marker rather than adding a new one. The default threshold for this “proximity” test is 1 millisecond.

### 78.2 Manually Creating Scene Changes

This feature is not currently implemented.

### 78.3 Playing back Scene Changes

Ardour has a dedicated MIDI port named “Scene Out”. Playing back scene changes can be done by connecting this port to whatever target(s) of MIDI scene (bank/program change) messages should be sent to.

When the global record enable button is *not* enabled, the relevant message(s) will be sent via the “Scene Out” port as the playhead rolls past each marker with a scene change associated with it.

## 78.4 Editing Scene Changes

This feature is not currently implemented.

## 78.5 Disabling Scene Changes

This feature is not currently implemented.

---

CHAPTER  
**SEVENTYNINE**

---

**PART VIII - ARRANGING**



---

CHAPTER  
**EIGHTY**

---

**67 - TIME, TEMPO AND METER**



## 67.1 - TEMPO AND METER

Tempo and meter belong together. Without both, there is no way to know where a beat lies in time.

Tempo provides a musical pulse, which is divided into beats and bars by a meter. When tempo is changed or an audio-locked meter is moved, all objects on the timeline that are glued to bars and beats (locations, regions) will move in sympathy.

When performing meter or tempo operations, it is advisable to use the BBT ruler (available by right-clicking an existing marker or ruler name), and ensure that the constraint modifier is set (in Preferences->User Interaction) so that no other modifiers share its key combination. The constraint modifier is the “Constrain drags using: ” setting under the “When Beginning a Drag” heading. One viable setting is .

### 81.1 Tempo

Tempo can be adjusted in several ways:

- by double clicking on a tempo marker. This opens the tempo dialog which allows entering the tempo directly into an entry box.
- by using the constraint modifier (which is set in Preferences->User Interaction) to drag the beat/bars in the BBT ruler or the tempo/meter lines. This is the preferred way to match the tempo to previously recorded material.
- by holding down the constraint modifier while dragging a tempo vertically. This is used for more complex tempo solving, as it allows changing of the position and tempo of a tempo marker in the same drag; it is, however, a useful way to adjust the first tempo for a quick result.

A tempo may be locked to audio or musical time. This can be changed by right-clicking on a tempo. If a tempo is locked to music, an entry will be available to lock it to audio. Similarly an audio-locked tempo may be locked to music by right-clicking it and selecting the “Lock to Music” entry.

Audio locked tempo marks stay in their frame position as their neighbour's positions are altered. Their pulse (musical) position will change as their neighbours move. Music locked tempo marks move their frame position as their neighbours are moved, but keep their pulse position (they will move as the music is moved).

A tempo may be constant or ramped:

- A constant tempo will keep the session tempo constant until the next tempo section, at which time it will jump instantly to the next tempo. These are mostly useful abrupt changes, and is the way in which traditional DAWs deal with tempo changes (abrupt jumps in tempo).
- A ramped tempo increases its tempo over time so that when the next tempo section has arrived, the session tempo is the same as the second one. This is useful for matching the session tempo to music which has been recorded without a metronome. Ramps may also be used as a compositional tool, but more on this later. Note that a ramp requires two points—a start and an end tempo. The first tempo

in a new session is ramped, but appears to be constant as it has no tempo to ramp to. It is only when a new tempo is added and one of them is adjusted that a ramp will be heard. The same applies to the last tempo in the session—it will always appear to be constant until a new last tempo is added and changed.

To add a new tempo, use the primary modifier and click on the tempo line at the desired position. The new tempo will be the same as the tempo at the position of the mouse click (it will not change the shape of the ramp).

To copy a tempo, hold down the primary modifier and drag the tempo to be copied.

## 81.2 Meter

Meter positions beats using the musical pulse of a tempo, and groups them into bars using its number of divisions per bar.

The first meter in a new session may be moved freely. It has an associated tempo which cannot be dragged by itself (although all others can). It can be moved freely and is locked to audio.

New meters are locked to music. They may only occur on a bar line if music locked.

An audio locked meter provides a way to cope with musical passages which have no meter (rubato, pause), or to allow a film composer to insert a break in music which cannot be counted in beats.

If a meter is audio-locked, its bar number is fixed from the point at which it left the main score. That bar number cannot be changed, nor can tempo motion allow the previous bar to overlap. If another bar is needed, lock the meter to music again (right click->”Lock to Music”), drag the meter to the desired bar and re-lock to audio. The new bar can be freely dragged again.

- To change a meter, double click it. A dialog will appear.
- To copy a meter, hold down and drag it.

---

CHAPTER  
EIGHTYTWO

---

## 67.2 - TECHNIQUES FOR WORKING WITH TEMPO AND METER

### 82.1 Matching a recorded tempo with a tempo ramp

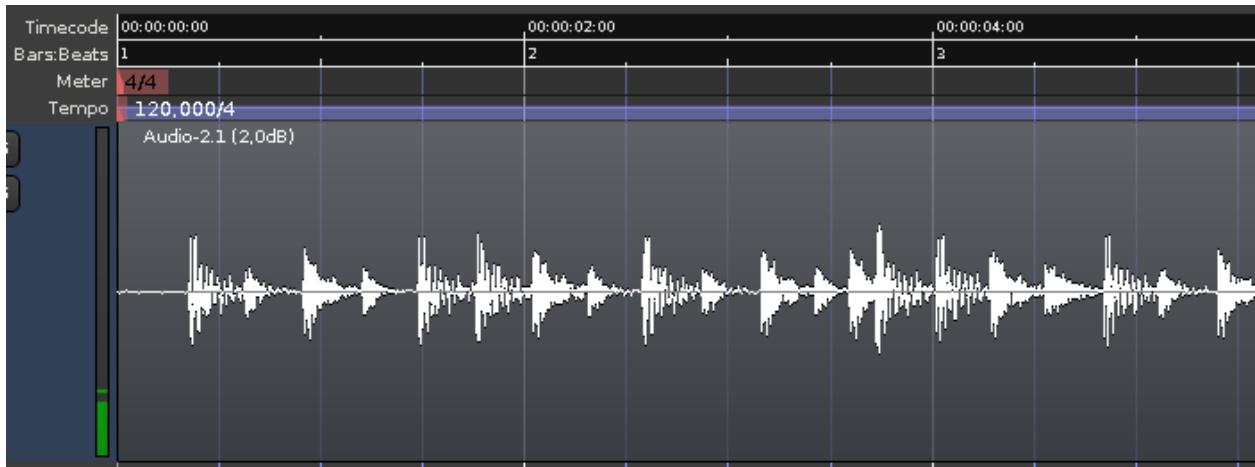


Fig. 1: Initial state.

As a general approach, the best way to control tempo ramps is to use them in pairs.

One typical use of tempo ramps is to match the click to a drum performance recorded in ‘free time’, like in the (admittedly bad) 4/4 example on the left.

#### 82.1.1 Step 1 : First meter

The first thing needed is determining where the first beat is in the recording and left dragging the first meter to that position.

#### 82.1.2 Step 2 : Locating the *n*th bar

Now the first click will be in time with the first beat. By listening to the recorded drums, the position of bar *n* (here, 9th beat, 3rd bar) is visually located (the playhead may be moved to this location to “pin” it).

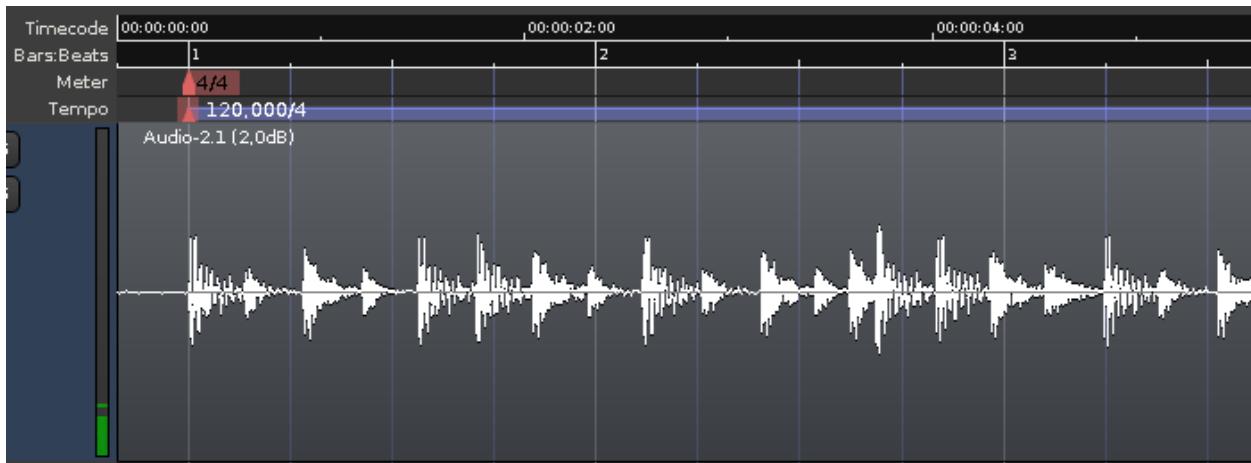


Fig. 2: Placing the first meter

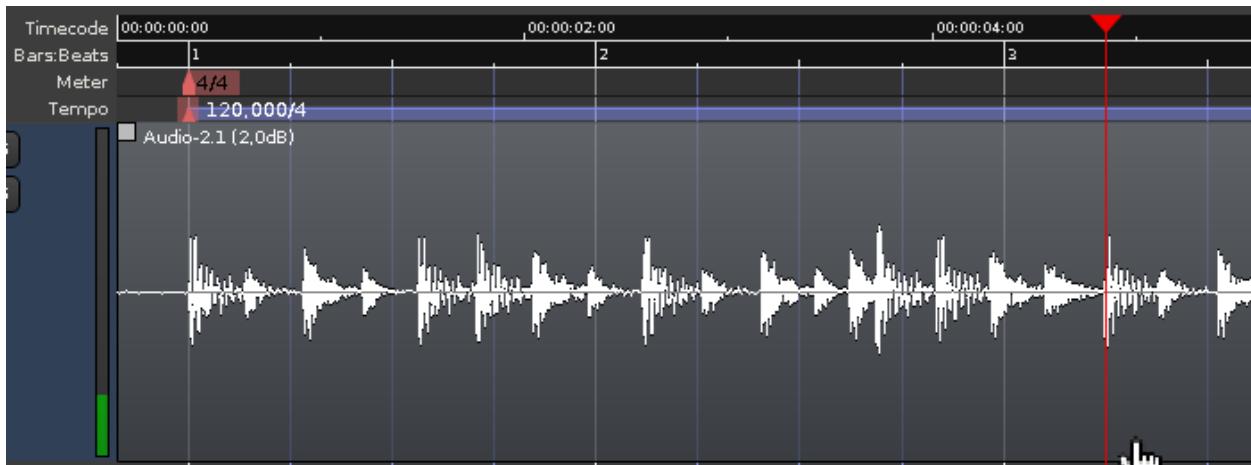


Fig. 3: Locating a known beat

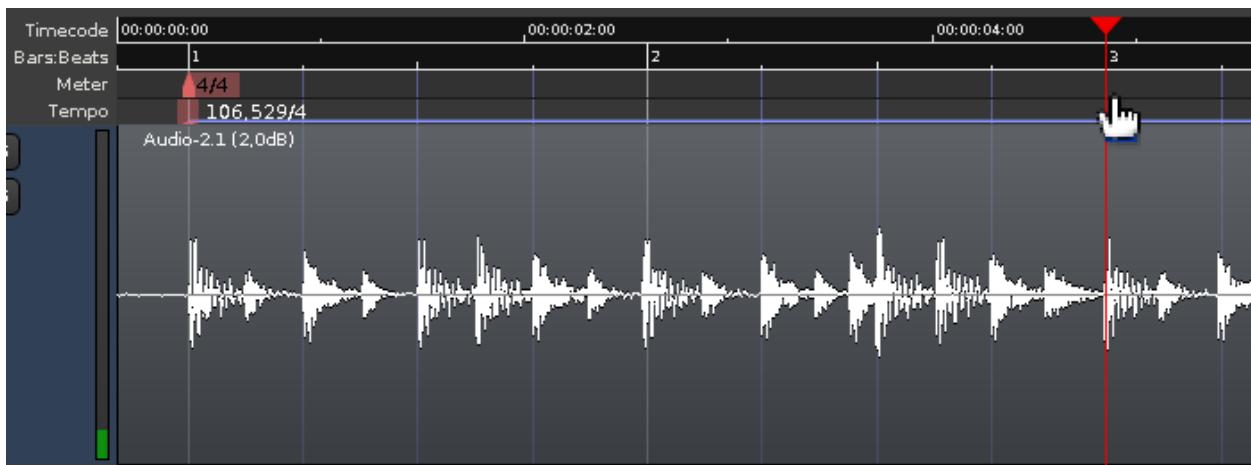


Fig. 4: Matching the tempi

### 82.1.3 Step 3 : Aligning the ruler with the tempo

Holding the constraint modifier ( by default), the third bar marker in the BBT ruler is dragged at the position of the third bar in the recording (where the playhead is located). This drag can be done either in the Meter or in the Tempo rulers. The tempo (on the first and only tempo marker) reflects the new value based on this change.

The click now matches the first 8 beats, but after that it can wander off, which will be reflected in the tempo lines that won't quite match the drum hits.

### 82.1.4 Step 4 : Placing a new tempo marker



Fig. 5: Creating a tempo marker

A new tempo marker is placed on the last position where the click matches the recorded audio, by -clicking the Tempo ruler. This will “anchor” the value of the tempo at that position.

### 82.1.5 Step 5 : Placing another tempo marker at the *n*th beat

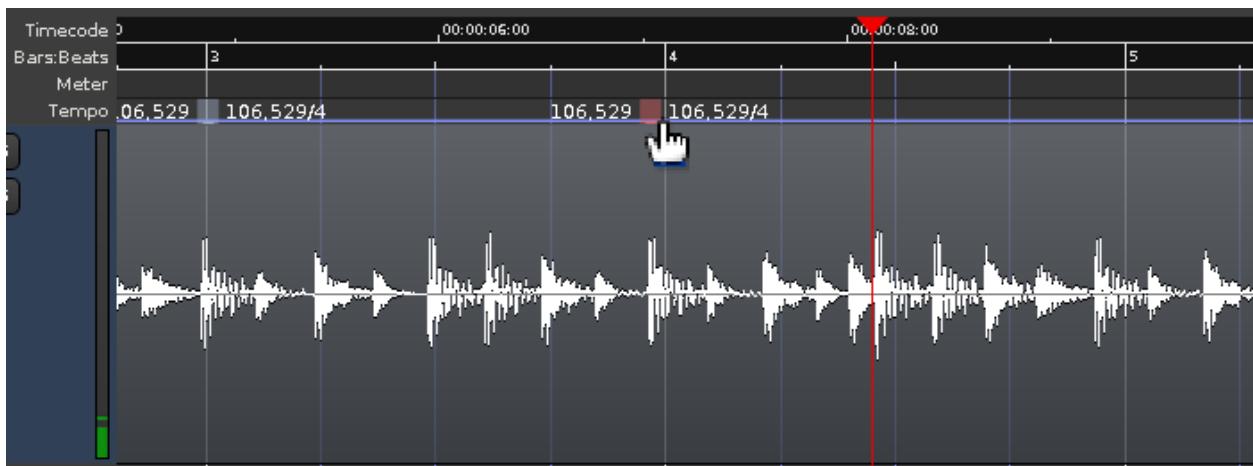


Fig. 6: Placing another marker

Another tempo marker is placed  $n$  beats after the previous marker (here, 4 beats, 1 bar).

### 82.1.6 Step 6 : Changing the tempo to a new value



Fig. 7: Adjusting the tempo

Now, -dragging any beat **after** the second new tempo marker will allow to align the drum audio and tempo after the second marker.

### 82.1.7 Step 7 : Ramping the tempo change



Fig. 8: Ramping the change

Although it may be unnecessary in some cases where the tempo changes abruptly, most of the time, the tempo change is progressive in time, like an instrumentist drifting in tempo. In those cases, the tempo change should be progressive too, and Ardour allows that by ramping the tempo change.

right clicking the first tempo marker, a menu appears, allowing to Ramp to Next. This will make the tempo between the two markers linearly change from the first marker's value to the second's.

Again, some time later the click will probably drift again, so the same technique has to be repeated: adding two new tempos and dragging the BBT ruler **after** the newest tempo so that the beats align with the audio again.

In a general sense, adding tempo markers in pairs allows to ‘pin’ the tempo at the marker’s location while moving further to the right.

## 82.2 Other use cases

Audio locked meters can be useful when composing, as they allow a continuous piece of music to be worked on in isolated segments, preventing the listening fatigue of a fixed form. Reassembly is left as an excercise for the reader.

Tempo ramps can also be used in a video context, e.g. for an accelerando, by snapping to TC frames and dragging the ruler so that a bar ends up on a significant video frame.



---

CHAPTER  
**EIGHTYTHREE**

---

**PART IX - MIXING**



---

CHAPTER  
**EIGHTYFOUR**

---

**68 - BASIC MIXING**



---

CHAPTER  
**EIGHTYFIVE**

---

## 68.1 - METERING IN ARDOUR

### 85.1 Introduction

An engineer reading and using audio level meters compares to a musician reading or writing sheet-music. Just like there are virtuoso musicians who can't read a single note, there are great sound-engineers who just go by their ears and produce great mixes and masters without ever looking at a single meter.

Yet, in order to work in or with the broadcast industry, it is usually unavoidable to use meters.

Audio level meters are very powerful tools that are useful in every part of the entire production chain:

- When tracking, meters are used to ensure that the input signal does not overload and maintains reasonable headroom.
- Meters offer a quick visual indication of an activity when working with a large number of tracks.
- During mixing, meters provide an rough estimate of the loudness of each track.
- At the mastering stage, meters are used to check compliance with upstream level and loudness standards and to optimize the loudness range for a given medium.

### 85.2 Meter Types

A general treatise on metering is beyond the scope of this manual. It is a complex subject with a history... For background information and further reading we recommend:

- [How To Make Better Recordings in the 21st Century—An Integrated Approach to Metering, Monitoring, and Leveling Practices](#) by Bob Katz. Has a good historic overview of meters and motivates the K-meter
- [Wikipedia: Peak programme meter](#)—overview of meter types.
- “[Audio Metering: Measurements, Standards and Practice: Measurements, Standards and Practice](#)”, by Eddy Brixen. ISBN: 0240814673
- “[Art of Digital Audio](#)”, by John Watkinson. ISBN: 0240515870

There are different metering standards, most of which are available in Ardour. In short:

Digital peak-meter	A Digital Peak Meter displays the absolute maximum signal of the raw audio PCM signal (for a given time). It is commonly used when tracking to make sure the recorded audio never clips. To that end, DPMs are always calibrated to 0 dBFS, or the maximum level that can be represented digitally in a given system. This value has no musical reason whatsoever and depends only on the properties of the signal chain or target medium. There are conventions for fall-off-time and peak-hold, but no exact specifications. Various conventions for DPM fall-off times and dBFS line-up level can be chosen in Edit > Preferences > Metering.
RMS meters	An RMS-type meter is an averaging meter that looks at the energy in the signal. It provides a general indication of loudness as perceived by humans. Ardour features three RMS meters, all of which offer additional peak indication. <ul style="list-style-type: none"> <li>• K20: A meter according to the K-system introduced by Bob Katz, scale aligned to -20 dBFS, rise/fall times and color schema according to spec.</li> <li>• K14: Same as K20 with scale aligned to -14 dBFS.</li> <li>• K12: Same as K20 with scale aligned to -12 dBFS (since 3.5.143).</li> <li>• Peak + RMS: standard RMS, customizable via Edit &gt; Preferences &gt; Metering</li> </ul>
IEC PPMs	IEC-type PPMs are a mix between DPMs and RMS meters, created mainly for the purpose of interoperability. Many national and institutional varieties exist (EBU, BBC, DIN). These loudness and metering standards provide a common point of reference which is used by broadcasters in particular so that the interchange of material is uniform across their sphere of influence, regardless of the equipment used to play it back. For home recording, there is no real need for this level of interoperability, and these meters are only strictly required when working in or with the broadcast industry. However, IEC-type meters have certain characteristics (rise-time, ballistics) that make them useful outside the context of broadcast. Their specification is very exact, and consequently, there are no customizable parameters.
VU meters	VU meters are the dinosaurs (1939) amongst the meters. They react very slowly, averaging out peaks. Their specification is very strict (300ms rise-time, 1–1.5% overshoot, flat frequency response). Ardour's VU meter adheres to that spec, but for visual consistency it is displayed as a bar-graph rather than needle-style (more below).

## 85.3 Ardour Specifics

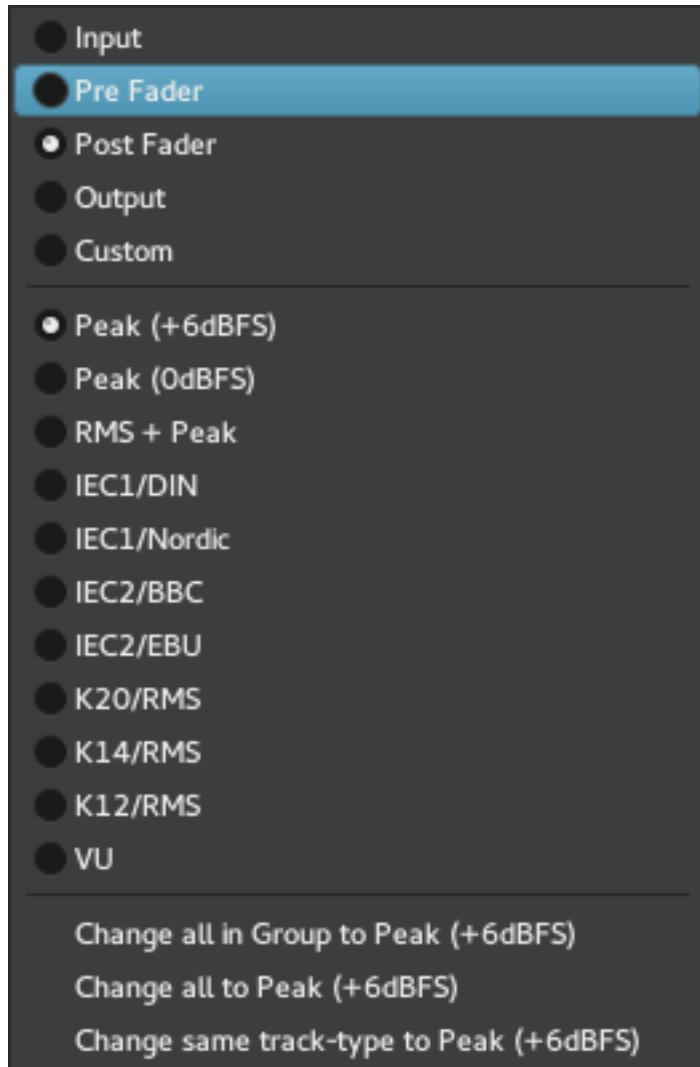


Fig. 1: Mixer strip meter context menu

Meters are available in various places in Ardour:

- The mixer window features fixed height meters for each channel strip.
- There are small (narrow) meters on each track-header in the editor window.
- There are variable height meters in the meterbridge window.
- Optionally, a fixed-size master meter can be displayed in the main toolbar.
- Various other locations (file import, sends) have level-meters.

They all share the same configuration and color-theme which is available in preferences and the theme-manager. Settings for the Peak and RMS+Peak meters as well as VU meter standards are found in Edit > Preferences > Metering.

The type of meter and the metering point (the place in the signal chain where the meter taps the signal) are configurable in the context menu of each meter. Depending on the Edit > Preferences > Mixer settings,

the metering point is also accessible via a button in each Mixer strip.

Regardless of meter type and standard the meter display will highlight red if the signal on the given channel exceeds the configured peak threshold.

Left clicking on the peak-indicator button resets the peak-hold indicator of a single channel.

Left clicking resets a whole group, and

Left clicking resets all meters.

## 85.4 Overview of meter types

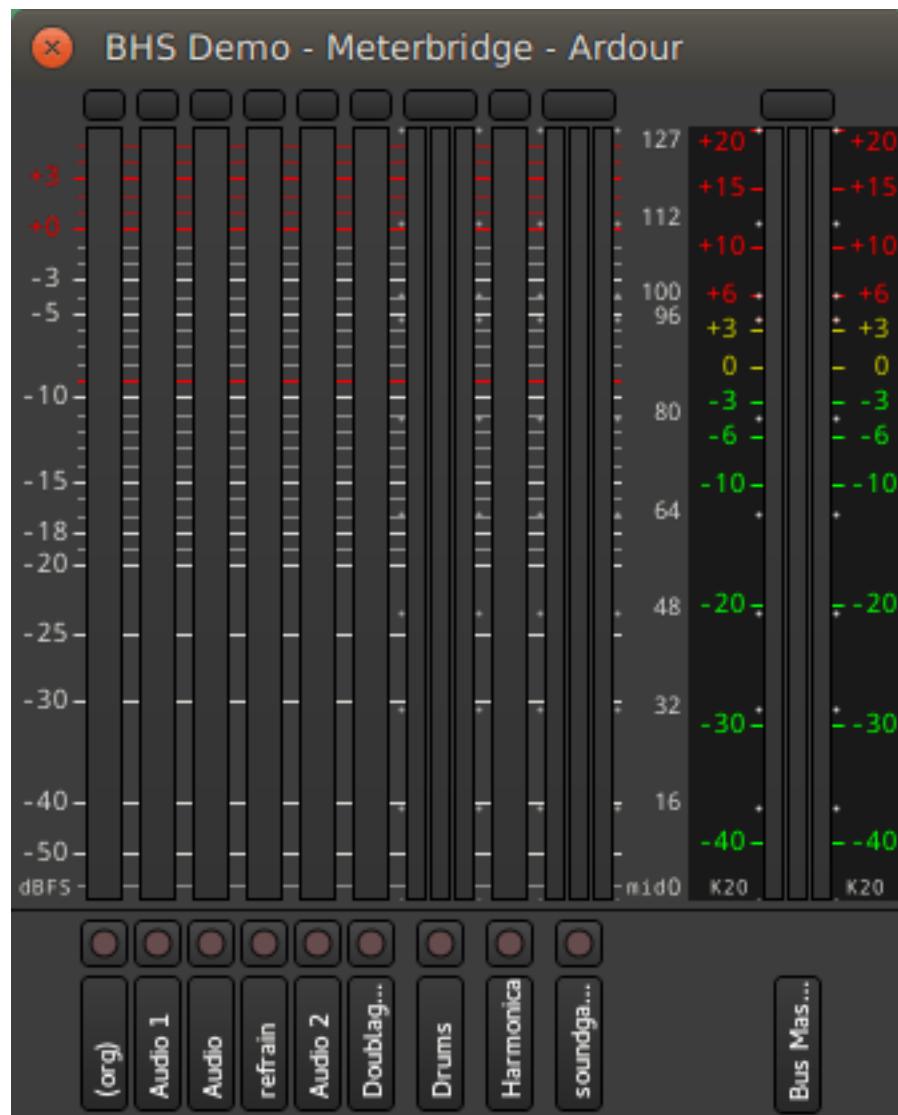


Fig. 2: Bar-graph meters in Ardour

The figure on the left shows all available meter-types in Ardour when fed with a -18 dBFS 1 kHz sine wave.

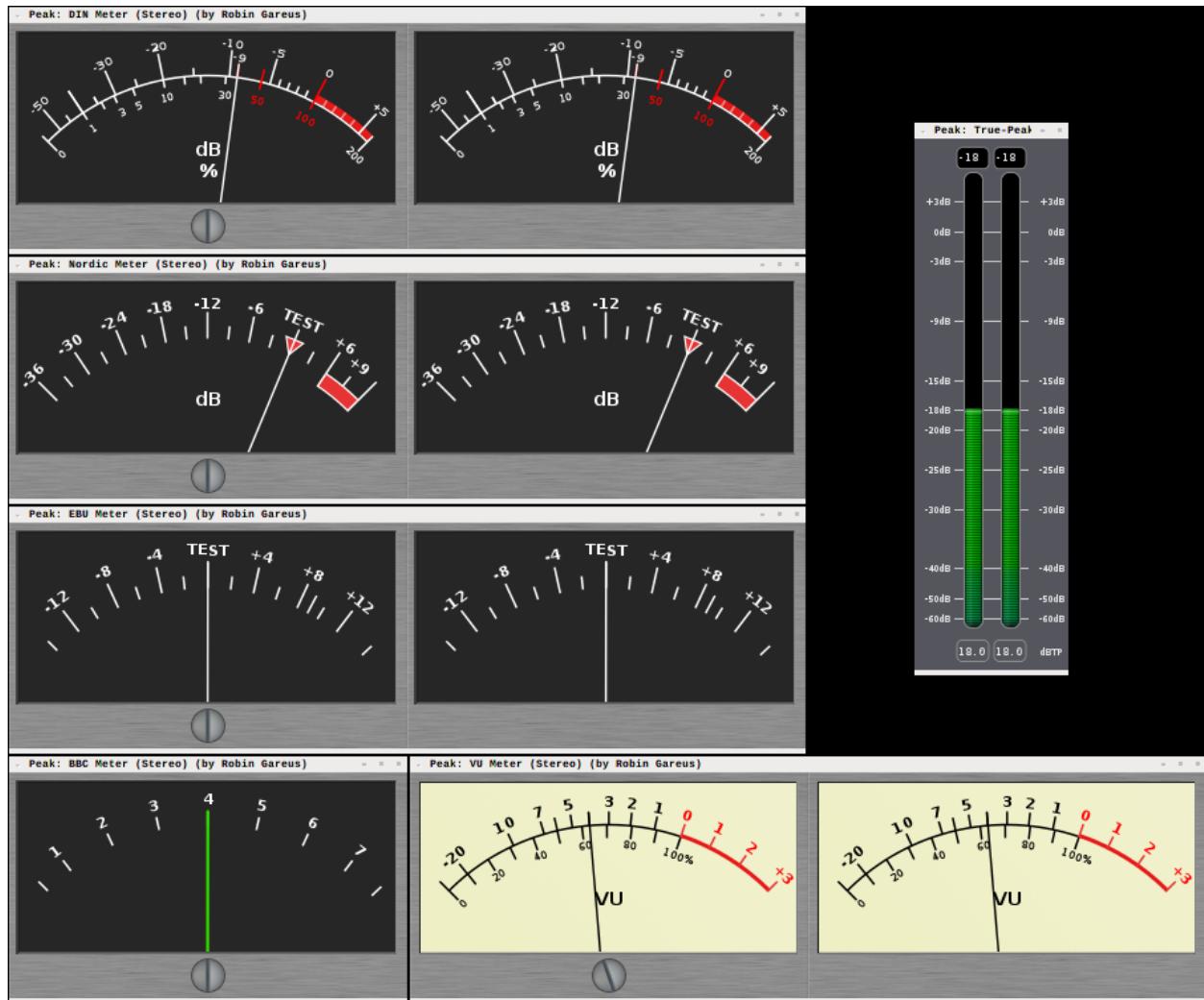


Fig. 3: Needle-style meters as external LV2 plugins

Due to layout concerns and consistent look and feel all meters available in Ardour itself are bar-graph type meters. Corresponding needle-style meters—which take up more visual screen space—are available as LV2 plugins (see image on the right): [meters.lv2](#).

## 68.2 - SIGNAL ROUTING

Ardour does most of its internal signal routing via JACK: all track and bus inputs and outputs are JACK ports, as are sends and inserts, which means they can be tapped into by other JACK clients. Only the signal flow inside a track or bus (i.e. from *processor to processor*) is handled internally.

By default, Ardour will automatically create the following connections:

- Track inputs are optionally auto-connected to hardware inputs, in round robin order, depending on the setting chosen in the *Session > New Session dialog*.
- Bus inputs are left disconnected.
- The number of track and bus outputs are equal to the number of inputs of the master bus.
- Track and bus outputs are always auto-connected to the master bus inputs.
- Master bus outputs are connected to hardware outputs.

This configuration is sufficient to do basic tracking and playback of many sessions without any adjustment by the user. Changing these connections is generally not necessary and often leads to problems.

However, for many workflows during mixing, more complicated signal routing is required. Ardour offers many possibilities for connecting things to fit any particular workflow.



## 68.3 - AUX SENDS

Auxiliary sends are simple *processors* in a bus or track channel strip. They tap the signal at a specific point in the signal flow (pre-fader, post-fader, before or after EQs and other plugins, etc.) and send a copy of that signal to a bus, without affecting the normal signal flow downwards to the channel fader.

Aux sends from several tracks are collectively sent to a bus in Ardour, to create a monitor mix for a musician, or to feed an effect unit. A bus used in this way is considered an auxiliary bus or Aux bus even though it is the same as any other bus. The output of such a bus might be routed to separate hardware outputs (in the case of headphone or monitor wedge mixes), or returned to the main mix (in the case of an effect).

Aux sends are not JACK ports, *External Sends* should be used to send audio to Jack ports. External Sends can send the tapped signal somewhere else directly, which is not usually possible on hardware mixers.

It may be useful to *compare and contrast* the use of aux sends with *subgrouping*.

### 87.1 Adding a new aux bus

New busses can be created using the Session > Add Track, Bus or VCA... menu, and selecting Audio Busses in the Template/Type selector on the left of the Add Track/Bus/VCA dialog.

### 87.2 Adding a send to an aux bus

Context-clicking on the processor box for the track to send to the bus, and choosing New Aux Send ... shows a submenu, listing the busses. Choosing one bus will add a send (which will be visible in the processor box). Note that if the only existing bus is the Master Bus, the menu will be grayed out.

#### 87.2.1 Pre-fader and Post-fader Aux Sends

Depending on whether the context-click happened above or below the fader in the processor box, the new aux send can be placed before or after the fader in the channel strip.

- Post-fader aux sends are typically used when using an aux for shared signal processing (FX), so that the amount of effect is always proportional to the main mix fader.
- Pre-fader sends ensure that the level sent to the bus is controlled *only* by the send, not the main fader—this is typical when constructing headphone and monitor wedge mixes.

The color of the processor will reflect this pre/post position (red for Pre, green for Post). Dragging and dropping the send inside the processor box before or after the Fader processor changes the type of fader accordingly.

## 87.3 Adding a new aux bus and sending a Track Group to it

All members of a group can be sent to a new aux bus at once with a single click. After creating the *track group* (and adding tracks to it), context-clicking on the group tab allows to choose either Add New Aux Bus (pre-fader) or Add New Aux Bus (post-fader). A new aux bus will be created, and a new aux send added to every member of the track group that connects to this aux bus.

## 87.4 Altering Send Levels

The amount of the signal received by a send that it delivers to the bus it connects to can be altered in two ways:

### 87.4.1 Using the Send Fader

Every send processor has a small horizontal fader that can be adjusted in the usual way. It is not very big and so this can be a little unsatisfactory if a very fine control over the send level is required.

### 87.4.2 Map Aux Sends To Main Faders

In Mixer mode, pressing the button marked Aux on a aux bus will alter the channel strip for every track or bus that feeds the aux bus. Many aspects of the strip will become insensitive and/or change their visual appearance. More importantly, the main fader of the affected channel strips will now control the send level and **not** the track gain. This gives a larger, more configurable control to alter the level. Clicking the Aux button of the aux bus again reverts the channel strips to their normal use.

## 87.5 Disabling Sends

Clicking on the small LED in the send display in the processor box of the channel strip will enable/disable the send. When disabled, only silence will be delivered to the aux bus by this track. When enabled, the signal arriving at the send will be delivered to the aux bus.

## 87.6 Send Panning

Send panners can be configured to either be independent of the main panner, or to follow it. The latter could be useful for Reverb effects, or for in-ear monitor mixes delivered in stereo.

## 68.4 - COMPARING AUX SENDS AND SUBGROUPS

Auxes and Subgroups share a common concept—they both provide a way for one or more tracks (or busses) to send their signal to a single bus so that common signal processing can be applied to the mix of their signals.

Aux sends leave the existing signal routing to the main mix in place, and are typically used to create a separate mix to send to (for example) monitors or headphones (for performer monitor mixes):

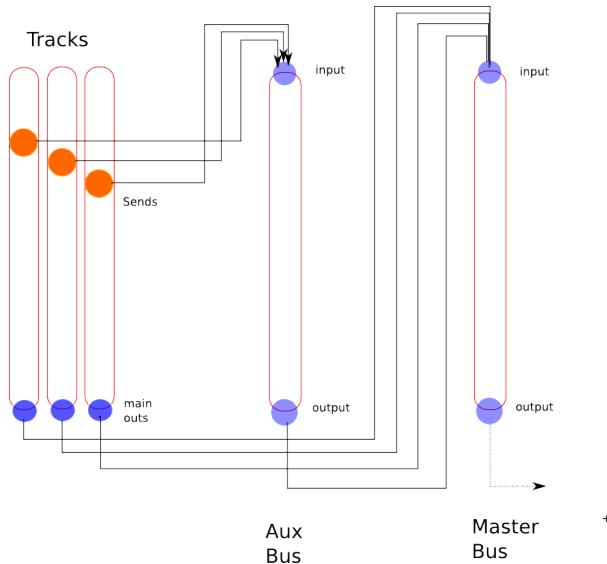


Fig. 1: Aux signal routing

Subgroups usually remove the original signal routing to the main mix and replace it with a new one that delivers the output of the subgroup bus to the main mix instead.

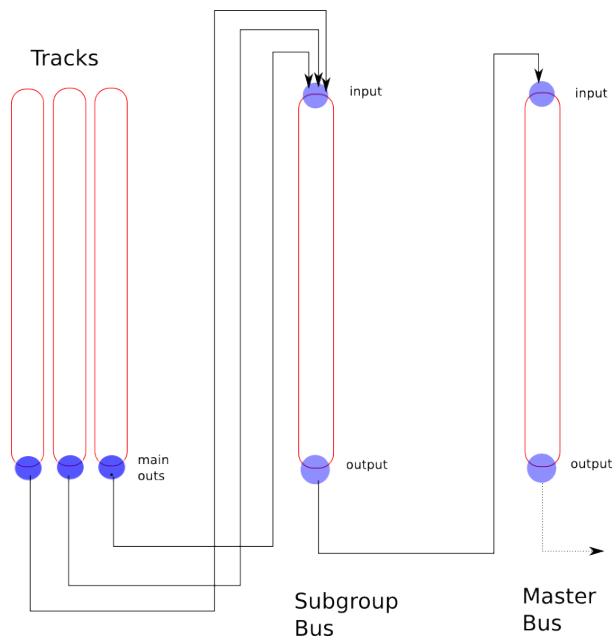


Fig. 2: Sub group signal routing

## 68.5 - EXTERNAL SENDS

Like a normal aux send, an external send taps the signal at a specific point within a channel strip, but delivers it to an external application or piece of hardware rather than an Ardour bus. By itself, an external send has no effect whatsoever on the audio signals within Ardour—it is a one-way signal routing that leaves all existing signal processing just as it was.

Most people will not have much use for this, but it can be useful to experiment with external applications or hardware signal processing applications.

### 89.1 Adding an External Send

Context-clicking on the *processor box* in a channel strip (at the desired location, pre or post fader) and choosing Add new External Send will show a dialog containing the standard Ardour *patchbay* to allow to connect the send to the desired destination.

### 89.2 Removing an External Send

An external send can be removed in several ways:

- Right-clicking the send in the processor box and choosing either Cut or Delete.
- Selecting the send (with a single left click) and pressing the Del key.

### 89.3 Altering Send Levels

Just below the send in the processor box is a small fader that can be used like all other faders in Ardour to control the gain applied to the signal delivered by the send. Dragging it alters the level, Shift-click restores to unity (0dB) gain.

### 89.4 Disabling Sends

Clicking the small LED in the send display within the processor box turns it on and off. When turned off, silence will be delivered to the send. When turned on, the signal within the channel strip will be delivered.

## 89.5 Editing Send Routing

Double-clicking on the send in the processor box will re-display the patchbay dialog that gives full control over the routing of the send.

---

CHAPTER  
**NINETY**

---

## **68.6 - INSERTS**

Inserts are signal tap points that can be placed anywhere inside a channel strip. Unlike Auxes, they will interrupt the signal flow, feeding the signal from before the insert point to its Insert send(s), and connecting the remainder of the channel strip to the Insert return(s), both of which are JACK ports which are visible to other JACK applications.

Inserts are the JACK equivalents of normalized switching jacks on an analog console.

An insert allows to either use a special external DSP JACK application that is not available as a plugin, or to splice an external analog piece of gear into a channel strip, such as a vintage compressor, tube equalizer, etc. In the latter case, the inserts would first be connected to a pair of hardware ports, which are in turn connected to the outboard gear.

Disabling (bypassing) an insert is done by clicking on its LED in the processor box.

When an insert is created, the signal will be interrupted until the relevant connections to the insert ports are made!

Inserts will incur an additional JACK period of latency, which can be measured and compensated for during mixing, but not during tracking!



---

CHAPTER  
**NINETYONE**

---

## 68.7 - SUBGROUPING

Subgrouping (sometimes known as “Grouping” or “Audio Grouping”) is a way to collect related signals together to apply some common treatment, before sending them on to the main mix. One standard application is to group several tracks belonging to the same instrument or section (such as a drum kit or horn section), to be able to adjust their volume with a single fader, after their inner balance has been set using the track faders.

Ardour also provides *VCA*s that is a very flexible way to adjust the volume of a group of tracks/busses when no additional processing is needed.

Create a subgroup from an existing Track/Bus group is done by right-clicking on the relevant *group tab*, and choosing Add new subgroup bus. A new bus will be created and every member of the track group will have its outputs disconnected from other destinations and then connected to the new bus inputs. The bus outputs will feed the master bus unless manual connections have been selected in the session preferences. The bus will be named after the track group name.

Alternatively, a group can be created manually, by first adding a new bus, then, for each track to be fed in the subgroup bus, disconnecting its outputs from the master and connecting it to the inputs of the subgroup bus instead. This can be done in the global audio patchbay or on a track by track basis via the output button of each track’s channel strip.

Remove a subgroup (bus) is done by right-clicking on the track group tab, and selecting Remove subgroup bus. Simply deleting the bus itself will **not** restore signal routing to the way it was before the addition of the subgroup bus—tracks that had been subgrouped will be left with their main outputs disconnected.



---

CHAPTER  
**NINETYTWO**

---

## **68.8 - PATCHBAY**

The patchbay is the main way to make connections to, from and within Ardour's mixer.

Notable exceptions are internal aux sends and connections to the monitor bus (when using one): these cannot be controlled from a patchbay, and are basically not under manual control at all.

The patchbay presents two groups of ports; one set of sources (which produce data), and one of destinations (which consume data). Depending on the relative number of each, the sources will be placed on the left or the top of the dialogue, and the destinations on the right or the bottom. Thus, in general, signal flow is from top or left to right or bottom.

Both sources and destinations are divided up into groups, with each group being given a tab:

Hardware	These are ports which are connected to a physical piece of hardware (a sound card or MIDI interface).
Ardour Busses	All ports belonging to busses.
Ardour Tracks	All ports belonging to tracks.
Ardour Misc	These are other ports that do not fit into the previous two categories; for example, the ports on which the metronome click is output, and MIDI ports for things like control surfaces and timecode.
Other	If there are other JACK clients running, their ports will be found here. If there are no such ports, the tab will not exist (on one or both axes of the grid).

The main part of the patchbay is a matrix grid. Within this grid, green dots represent connections, and any of the squares can be clicked on to make or break connections. Clicking and dragging draws a line of connections, which is sometimes useful for making many connections at once.

In the example patchbay shown above we can note various things. We are using the Ardour Tracks sources tab, so we see the output ports of the three tracks in our session: Fred, Jim and Foo. Our destinations are from the Ardour Busses tab, so we have the inputs of a session bus, Sheila, and the inputs of the master bus. Fred and Jim have stereo outputs, so have L and R connections. Foo is a MIDI track, so it only has one connection, and its squares in the grid are coloured light grey to indicate that no connection can be made between Foo (a MIDI output) and our busses (which are all audio-input).

The green dots in the example show that both Fred and Jim are connected to the master bus, left to left and right to right.

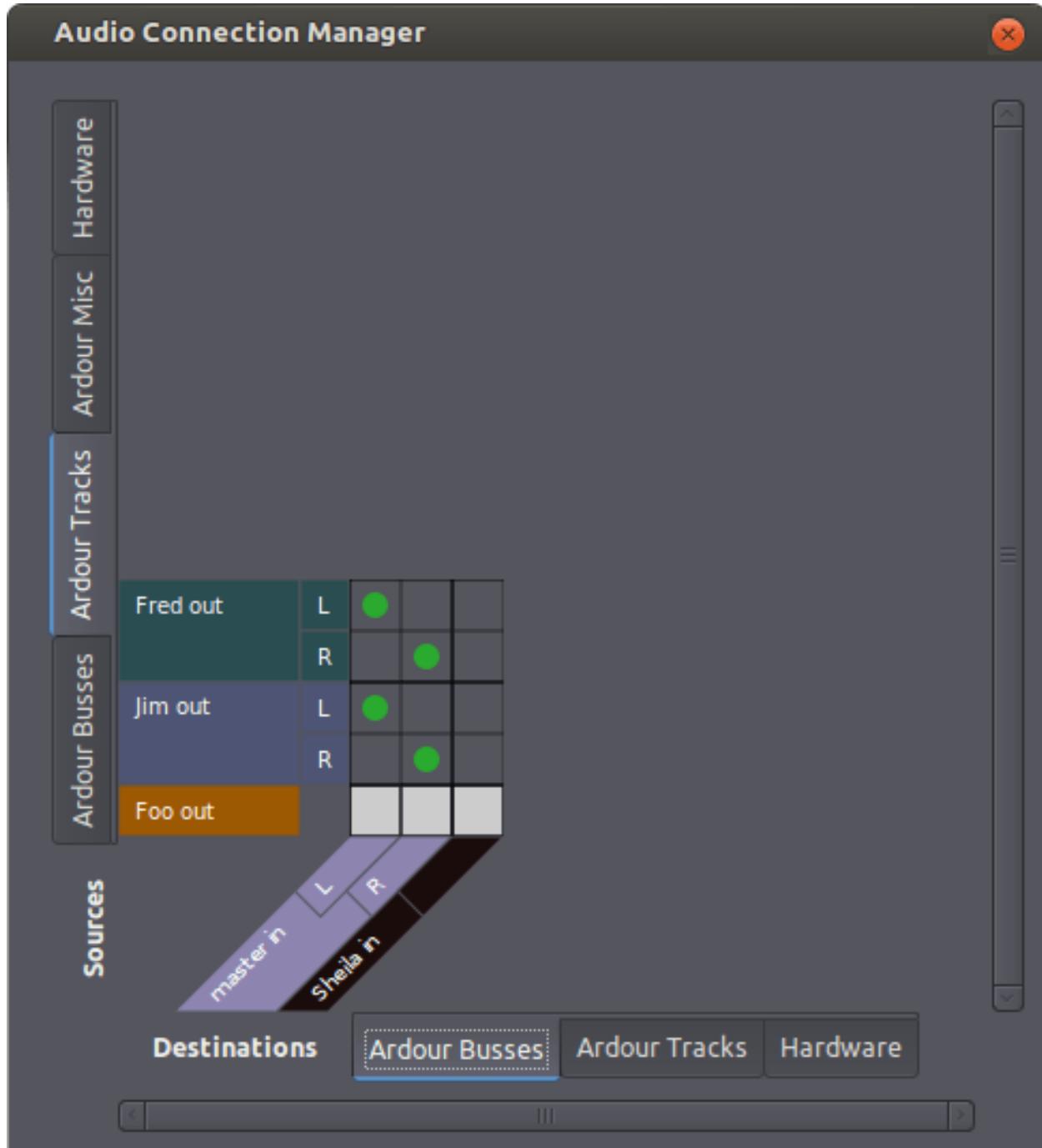


Fig. 1: An example patchbay

## 92.1 Variants on the Patchbay

Slightly different versions of the patchbay are available from different places in Ardour. A global view of all JACK audio connections is available, in Window > Audio Connections, or by pressing P. A corresponding MIDI Connection Manager can be opened using P.

There is also a patchbay available when connecting individual tracks; clicking on the input or output buttons of a mixer strip will open a connection manager which has the corresponding track input or output as the only destination or source, with all other ports available for connection to it.

## 92.2 Other patchbay features

right-clicking on a port name in the connection manager opens a context menu which provides a few handy options:

Add audio port and Add MIDI port	These options add audio or MIDI ports to the clicked source, if this is possible. In this way, for example, tracks and busses can be extended to have more inputs or outputs.
Remove <i>port name</i>	Removes the given port, if possible. Right-clicking a port will do the same.
Disconnect all from <i>port name</i>	Disconnects everything from the given port.
Rescan	Ardour will try to keep abreast of any changes to the JACK ports on the system, and reflect them in any connection managers which are open. If for some reason this fails, this can be used to re-scan the list of ports and update the manager.
Show individual ports	If a session has lots of multi-channel tracks or busses, it may be an unnecessary detail that left has to be connected to left and right to right every time a connection is made. This obviously gets worse with higher channel counts (such as for 5.1 or Ambisonics). To make life easier with such sessions, Show individual ports can be unticked. After that, the channels of tracks and busses will be hidden, and any green dots added in the connection manager will automatically connect each channel of the source to the corresponding channel of the destination (left to left, right to right and so on). In this mode, a half-circle in the connection grid indicates that some (but not all) of the source's ports are connected to the destination.
Flip	This will flip the visible ports on the vertical axis with those on the horizontal. If, for example, the top of the connection manager is showing Ardour Busses and the right is showing Hardware, flip will swap the view to the opposite. Flipping can also be done by pressing f. Note that if there are no matching tabs on both axes, flipping will be impossible.



## 68.9 - TRACK/BUS SIGNAL FLOW

### 93.1 Overview

In each individual Track or Bus the signal flow is top to bottom, as shown in the diagram on the right.

Trim, Fader and Panner are provided by Ardour. The Processor Box can hold third party plugins or host-provided redirects (insert, aux-send, etc.).

An important aspect is that the signal flow is multi-channel and not fixed throughout the track. For example, a track can have a mono input, a mono to stereo plugin (e.g. reverb) flowing into a surround panner with 6 outputs.

The design of Ardour is that the width of the signal flow is defined by the passage through plugins in the processor box, followed by panning. The number of inputs to the panner is defined by the number of outputs of the last plugin in the chain. The number of panner outputs is equal to the track's outputs ports, which can be added and removed dynamically. This schema is called *Flexible I/O*. It is very powerful and a distinctive feature of Ardour.

The golden rule of processor signal flow: The number of outputs of one link of the process chain defines the number of inputs of the next, until the panner.

Due to this rule there is one very common case that is hard to achieve: keeping a mono track mono. With *Flexible I/O*, if a stereo plugin is added on a mono track, the signal flow after that plugin becomes stereo.

### 93.2 Strict I/O

Strict I/O enforces a simple rule: plugins have the same number of inputs as they have outputs. By induction the track will have as many output ports as there are input ports.

Adding a plugin will not modify the signal flow. The number of plugin outputs is forced to the number of inputs present at the point of insertion. If a plugin pin is missing, it is ignored. If a plugin pin is not connected, it is fed with silence. Non-connected plugin outputs are ignored.

Strict I/O enforces the number of output ports. The number of inputs to the panner (outputs of last plugin) defines the number of track outputs (after panner). Required ports are automatically added, excess ports are removed. The user cannot manually add or remove output ports.

Strict I/O is set when creating the track and can later be enabled or disabled dynamically in the context menu of every mixer strip.

There are two exceptions to the above rule:

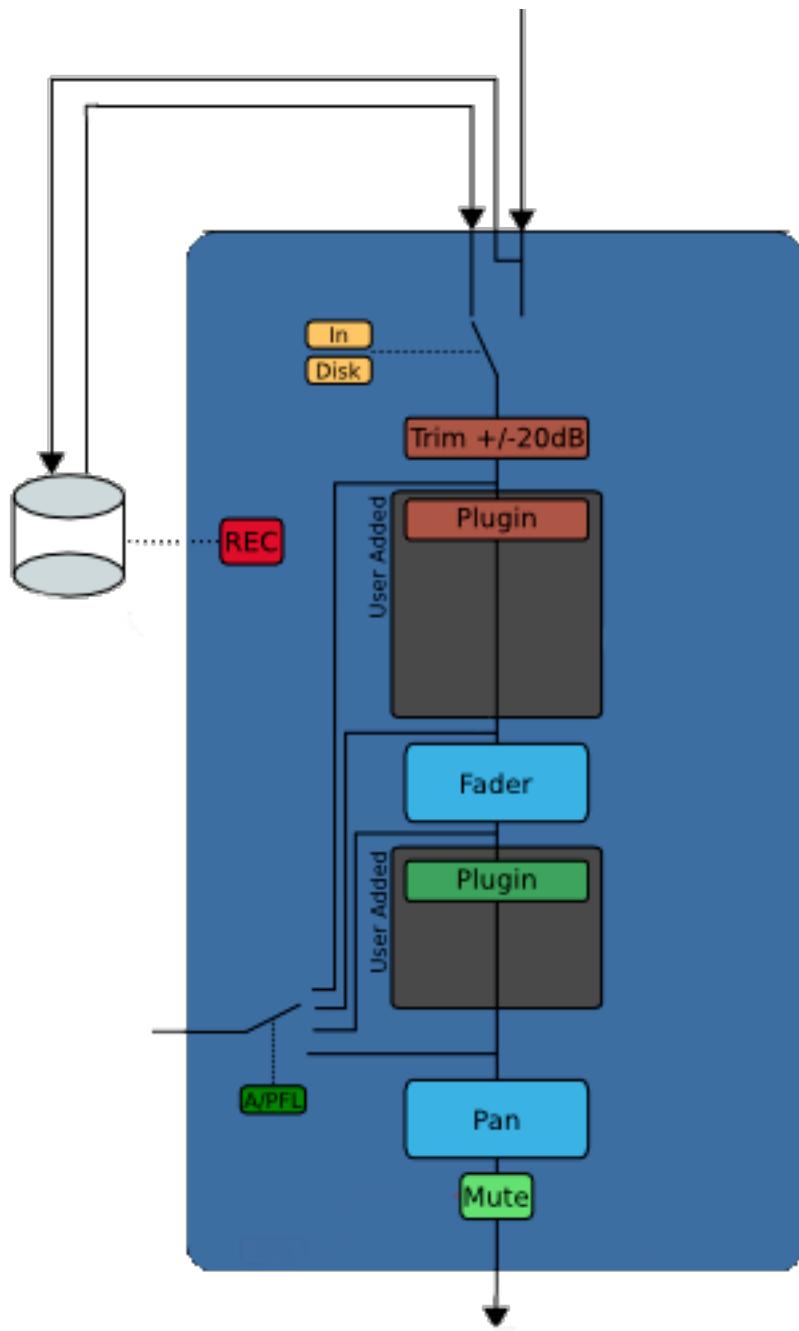


Fig. 1: Typical signal routing in a channel strip.



Fig. 2: Flexible vs. Strict I/O.

- Midi Synths. When adding a synth at a point where there is a Midi port only, the synthesizer plugin will add audio output ports, which trickle down the processor chain to all follow up plugins as inputs and in turn force their outputs to match
- Side chain inputs are not affected by Strict I/O

### 93.3 Customizing the Signal Flow: The Pin Connection window

The signal flow through the mixer can be customized at every processor node via Pin Configuration in the context menu of every processor. User customization overrides all automatic (Flexible and Strict I/O mode) inferred output port settings.

The Pin Connection window is made of three vertical sections:

- an I/O config column
- an interactive diagram
- a sidechain column

By default, the I/O config is set to *Automatic*, i.e. the Manual Config LED light is turned off. In this mode, the diagram will display the standard input/outputs for this plugin, i.e. the number of ports (inputs & outputs) is equal to the number of pins on the plugin, and a one-to-one connection is automatically created.

Adding new instances of the plugin allows to apply this plugin to more inputs or outputs. E.g., a mono effect can be applied to each channel of a  $n$ -channels track by adding as many instances of the plugins as there are input channels (i.e. ports). This happens automatically when adding, e.g., a mono effect to a stereo track:

- Ardour creates two instances of the plugin
- the plugin gets a (2x1) label in the processor box
- its two input ports are each connected to one pin of an instance
- each mono output pin of the plugin is connected to one output port

Output channels can also, in Manual Config mode, be added or removed, whether they are audio or MIDI.

Using the Pin Connection overrides the I/O config setting (Flexible vs. Strict). A processor *can*, even in Strict I/O mode, have a different number of outputs than inputs. Non-customized plugins downstream will follow suit depending on the selected route mode. e.g. adding an additional output to a plugin on a track set to Strict I/O will trickle down the process chain to the output and result in the addition of an output port. This is useful for example in case of a mono to stereo reverb.

The window allows connection of the I/O ports to the plugin pins and other I/O ports, provided they are compatible (MIDI vs. audio), just by dragging and dropping the end connectors on top of one another. A dotted connector's line is a “*thru*” line that directly connects an input to an output without connecting to a pin on the plugin—hence without any audio modification. These “*thru*” connections are latency compensated, with respect to those being affected by the plugin, in order to avoid phasing issues.

An example of using “*thru*” connections, shown below, is separate left/right channel equalization using two mono plugins on a stereo track:

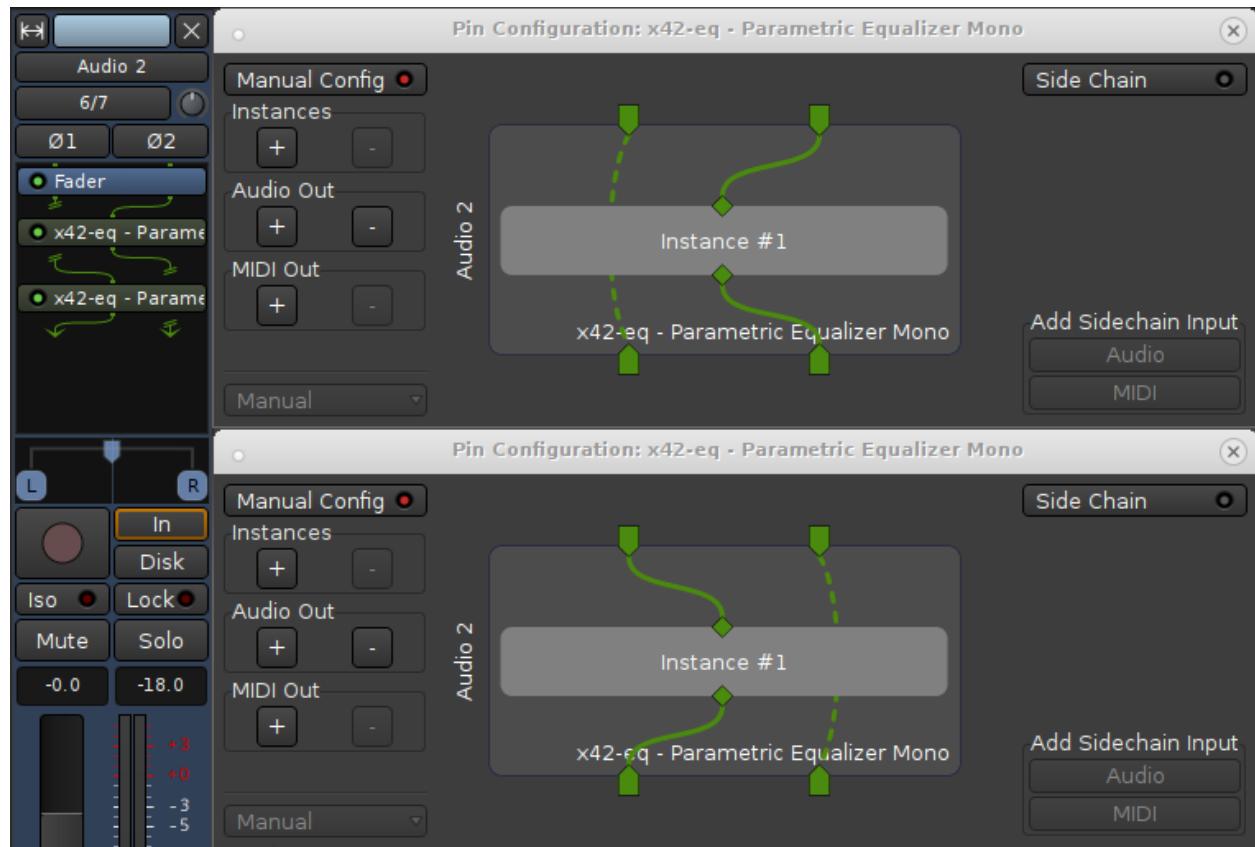


Fig. 3: An example of using two mono plugins on a stereo track.

The only way to add inputs to a processor is via *Sidechaining* from another signal. This is done by “tapping” the signal from another track or bus at any point.

Adding a sidechain signal in Ardour is as simple as enabling the Side Chain button in the Pin Configuration window, and choosing an Audio or MIDI sidechain in the Add Sidechain Input lower right hand section. A new drop-down menu appears, which displays a list of the tracks/busses available to be sidechained, or, for a more complex setup (e.g. sidechaining from hardware directly), the *Routing Grid* (also accessible with a Right-click on the drop-down menu).

The sidechain ports can then be connected, as other inputs, to a pin of the plugin, or an output port as a “*thru*”.

## 68.10 - SIDECHAINING

Dynamic Processors—such as compressors—in general use the the original input signal for analysis and operate on the same signal. Side-chaining uses the signal level of *another input* to control the compression level of the original signal.

Effect Processors which have a side-chain input (sometimes also called *key input*) have an additional input pin to receive a signal from an external input. In Ardour that extra input can be connected in the plugin's Pin Configuration dialog: the signal from one track can be tapped off and used as an input to a plugin on a different track. This dialog is accessed via the plugin's context-menu > Pin Connections....

In case a plugin has a dedicated sidechain input, Ardour automatically creates a port for the input. This is a normal I/O port which can be fed by any external signal. The Pin Configuration dialog is not limited to processors with a dedicated sidechain input, it also allows to manually create (or remove) a sidechain input port and provides for flexible connection of the signal to plugin pins.

The operational flow in the Ardour GUI starts at the processor which is to receive the signal: a sidechain source is selected, and Ardour creates a dedicated send-processor in the source processor box, the level of which can be adjusted either in the Pin Configuration window or directly on the source's send.

### 94.1 A simple example: Sidechain compression

One example is the use of a bass drum track to trigger the compression on a bass track. The sidechain compressor (*a-Compressor*) will be placed on the bass track, and will need to receive the signal from the bass drum track as a way to trigger the compression.

Here, on the bass track, an *a-Compressor* has been added, and the Drum track has been set as the sidechain source. The mixer reflects this by showing an *SC-send* processor in the drum track, very similar to a *send*. The bass track also shows an arrow as one of the *a-compressor* input.

As a result, in the editor, each peak in the kick drum track triggers the compression on the bass track and the resulting track shows the compression kicking in on each kick drum peak, hence reducing the gain. The compression is applied to the bass, but only based on the level of the drum track.

This is commonly used for *ducking* effect, when e.g. a radio speaker's voice triggers the compression on the audio playing.

### 94.2 MIDI Sidechaining

Ardour allows the sidechain sources to be either audio or MIDI tracks/busses. This is particularly useful when a MIDI signal is used to control an audio effect, like a vocoder or an auto-tuner, like *fat1*, the LV2 port of Fons Adriaensen's *Zita AT1* by Robin Gareus:

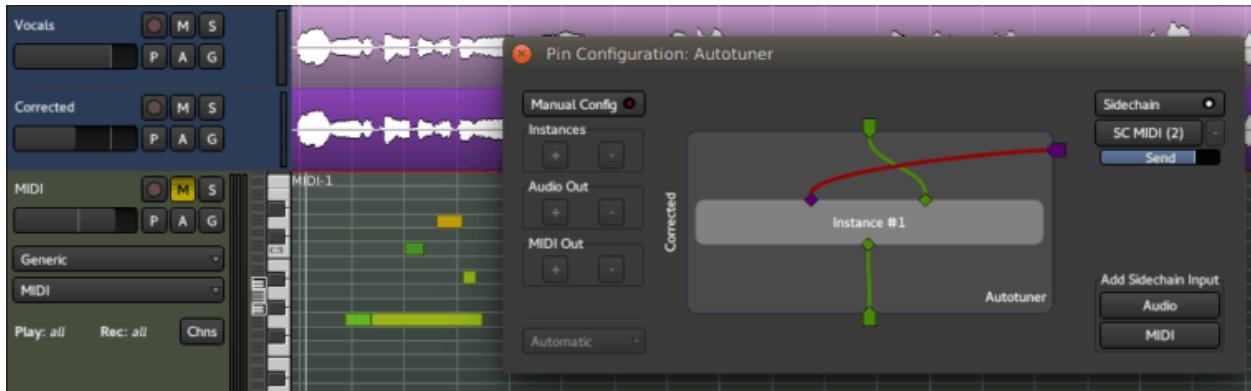


Fig. 1: MIDI sidechaining example: fat1.lv2.

Here, the MIDI track is inputted to the plugin's MIDI IN pin through a sidechain, indicating to the plugin what note the source audio should be corrected to.

Notice that in the example above, the output of the "Vocals" track is connected to the input of the "Corrected" track. We could have chosen to insert the "Vocals" track content as an audio sidechain too, totally disconnecting the input from the plugin, and connecting the plugin's input pin to the audio sidechain port.

### 94.3 Pre-processing the sidechained signal

Sometimes, the effects of a sidechain signal on a plugin can be enhanced by pre-processing the signal.

In the first example above, if the entire drum part is on one track, then compressing with this signal as a sidechain will result in every peak triggering the compression, be they bass drum kicks or snare, cymbals, etc.

In this case, adding an EQ to the drum track with a low pass filter would filter out the peaks created by the high pitched instruments of the drum kit, and allow for a better triggering, though to avoid damaging the original drum track, a send to an intermediary track would be better suited to place the EQ on. This track won't be connected to the Master, as its content is of no musical interest except for its use as a trigger, allowing for some extreme EQ.

## **68.11 - MUTING AND SOLOING**

Each track and bus has two buttons which have important implications for signal flow: mute and solo. The behaviour of these buttons is configurable in Ardour, to suit different studio set-ups.

### **95.1 Without a monitor bus**

When using Ardour without a monitor bus, there is only one way in which mute and solo will work:

- Mute on a track or bus will mute that track on the master bus, so that it will not be heard.
- Solo on a track or bus will solo that track or bus and mute all others. Soloing a bus will also solo any tracks or busses which feed that bus.

### **95.2 With a monitor bus**

For setups with a monitor bus, more options are available, mostly governed by the setting of the Solo controls are Listen controls option in Edit > Preferences > Mixer.

With Solo controls are Listen controls unticked, behaviour is almost exactly the same as the situation without a monitor bus. Mute and solo behave the same, and the monitor bus is fed from the master bus, so it sees the same thing.

With Solo controls are Listen controls ticked, the master and monitor busses behave differently. In this mode, solo controls are more properly called listen controls, and Ardour's solo buttons will change their legend from S to either A or P to reflect this.

Now, without any mute or listen, the monitor bus remains fed by the master bus. Also:

- Mute will mute the track or bus, so that it will not be heard anywhere (neither on the master nor monitor busses), much as before.
- Listen will disconnect the monitor bus from the master bus, so that the monitor bus now only receives things that are “listened to”. Listen will not perform any muting, and hence the master bus will not be affected by a listened track or bus.

When solo controls are listen controls, the listening point can be set to either After-Fade Listen (AFL) or Pre-Fade Listen (PFL). The precise point to get the signal from can further be configured using the PFL signals come from and AFL signals come from options.

The solo-mute arrangement with a monitor bus is shown below:

Here we have a number of tracks or busses (in orange). Each one has an output which feeds the master bus. In addition, each has PFL and AFL outputs; we have a choice of which to use. PFL/AFL from each

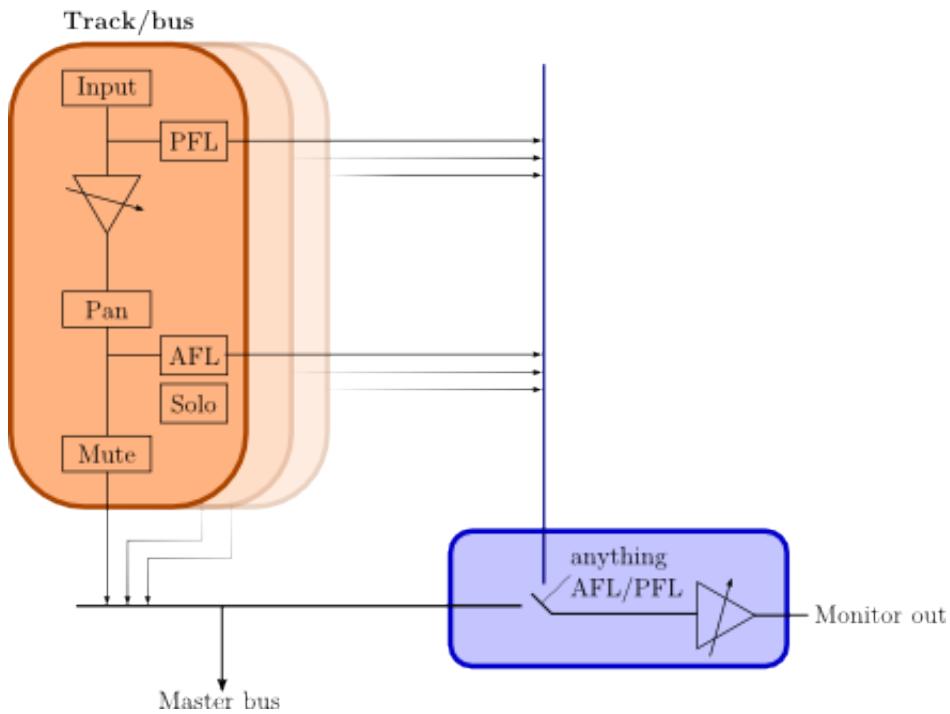


Fig. 1: Mute/solo signal flow

track or bus are mixed. Then, whenever anything is set to AFL/PFL, the monitor out becomes just those AFL/PFL feeds; the rest of the time, the monitor out is fed from the master bus.

In this scheme Solo has no effect other than to mute other non-soloed tracks; with solo (rather than listen), the monitor out is fed from the master bus.

### 95.3 Other solo options

Edit > Preferences > Mixer has some more solo options:

#### 95.3.1 Solo-in-place mute cut

When using solo-in-place (SiP), in other words when soloed tracks are being listened to on the master bus, this fader specifies the gain that will be applied to other tracks in order to mute them. Setting this level to  $-\infty$  dB will mean that other tracks will not be heard at all; setting to some higher value less than 0dB means that other non-soloed tracks will be heard, just reduced in volume compared to the soloed tracks. Using a value larger than  $-\infty$  dB is sometimes called “Solo-In-Front” by other DAWs, because the listener has the sense that soloed material is “in front” of other material. In Ardour, this is not a distinct mode, but instead the mute cut control offers any level of “in-front-ness” that is desired.

#### 95.3.2 Exclusive solo

If this is enabled, only one track or bus will ever be soloed at once; soloing track B while track A is currently soloed will un-solo track A before soloing track B.

### 95.3.3 Show solo muting

If this is enabled, the mute button of tracks and busses will be drawn outlined to indicate that the track or bus is muted because something else is soloed. This is enabled by default, and it is recommended to leave it that way unless extremely comfortable with Ardour's mute/solo behaviour.

### 95.3.4 Soloing overrides muting

If this is enabled, a track or bus that is both soloed and muted will behave as if it is soloed.

### 95.3.5 Mute affects...

These options dictate whether muting the track will affect various routes out of the track; through the sends, through the control outputs (to the monitor bus) and to the main outputs.



## 68.12 - PANNING

Panning is the process of distributing one or more signals across a series of outputs so that the listener will have the experience of them coming from a particular point or area of the overall listening field.

It is used to create a sense of space and/or a sense of motion in an audio mix. Different signals can be spread out across the space, and moved over time.

### 96.1 Types of Panners

The way a panner works depends a great deal on how many signals it is going to process and how many outputs it will send them to. The simplest case is distributing a single signal to 2 outputs, which is the common case when using a “mono” track and a stereo speaker setup.

But panning in Ardour could theoretically involve distributing any number of signals to any number of outputs. In reality, Ardour does not have specific panners for each different situation. Currently, it has dedicated panners for the following situations:

- 1 signal distributed to 2 outputs (the *mono panner*)
- 2 signals distributed to 2 outputs (the *stereo panner*)
- N signals distributed to M outputs (the *VBAP panner*)

Even for each of these cases, there are many different ways to implement panning. Ardour currently offers just one solution to each of these situations, but in the future will offer more.

In addition to the panners, Ardour has a balance control for subtle corrections to existing stereo images.



## 68.13 - MONO PANNER

The default mono panner distributes 1 input to 2 outputs. Its behaviour is controlled by a single parameter, the position. By default, the panner is centered.

### 97.1 Mono Panner User Interface



Fig. 1: The mono panner

The mono panner looks quite similar to the [stereo panner](#) interface. The difference is that the L/R labels in the lower half of the mono panner do not move because there is no “width” to control.

On the adjacent picture, the panner is centered, as shown by the central position of the slider, called position indicator.

### 97.2 Using the mouse

To change the position smoothly, press the right button and drag anywhere within the panner. *Note: grabbing the position indicator is not needed in order to drag.*

Reset to defaults	Click right
Change to a “hard left”	Double click right in the left side of the panner
Change to a “hard right”	Double click right in the right side of the panner
Set the position to center	Double Click right in the middle of the panner

### 97.3 Keyboard bindings

When the pointer is within a mono panner user interface, the following keybindings are available to operate on that panner:

$\leftarrow / \leftarrow$	move position $1^\circ / 5^\circ$ to the left
$\rightarrow / \rightarrow$	move position $1^\circ / 5^\circ$ to the right
0	reset position to center

## 97.4 Using the scroll wheel/touch scroll

When the pointer is within a mono panner user interface, the scroll wheel may be used as follows:

or	move position to the left by 1°
or	move position to the left by 5°
or	move position to the right by 1°
or	move position to the right by 5°

## 68.14 - BALANCE CONTROL

For stereo tracks, it is possible to switch between the default stereo panner and a traditional balance control by right-clicking on the panner widget.



Fig. 1: Stereo Balance control

When the balance is centered, the incoming signals will be unaffected. Moving it to one side will linearly attenuate the signal of the opposite side.

While the balance control is considerably less flexible than the stereo panner, it works with arbitrary content without danger of introducing comb filter artefacts.



## 68.15 - STEREO PANNER

The default stereo panner distributes two inputs to two outputs. Its behaviour is controlled by two parameters, width and position. By default, the panner is centered at full width.

The stereo panner assumes that the signals to distribute are either uncorrelated (i.e. totally independent), or that they contain a stereo image which is mono-compatible, such as a co-incident microphone recording, or a sound stage that has been created with pan pots.\*

With the default values it is not possible to alter the position, since the width is already spread entirely across both outputs. To alter the position, the width must first be reduced.

### 99.1 Stereo Panner User Interface



Fig. 1: The Stereo Panner

The panner user interface consists of three elements, divided between the top and bottom half. Clicking and/or dragging in the top half controls position; clicking and/or dragging in the bottom half controls width (see below for details).

In the top half is the position indicator, which shows where the center of the stereo image is relative to the left and right edges. When this is the middle of the panner, the stereo image is centered between the left and right outputs. When it all the way to the left, the stereo image collapses to just the left speaker.

In the bottom half are two signal indicators, one marked L and the other R. The distance between these two shows the width of the stereo image. If the width is reduced to zero, there will only be a single signal indicator marked M (for mono), whose color will change to indicate this special state.

It is possible to invert the outputs (see below) so that whatever would have gone to the right channel goes to the left and vice versa. When this happens, the entire movable part of the panner changes color to indicate clearly that this is the case.

#### 99.1.1 Position vs. L/R

Although the implementation of the panner uses the “position” parameter, when the user interface displays it numerically, it shows a pair of numbers that will be familiar to most audio engineers.

Position	L/R	English
0	L=50% R=50%	signal image is midway between left and right speakers
-1	L=100% R=0%	signal image is entirely at the left speaker
1	L=0% R=100%	signal image is entirely at the right speaker

One way to remember this sort of convention is that the middle of the USA is not Kansas, but “Los Angeles: 50% New York: 50%”.

### 99.1.2 Examples In Use

Appearance	Settings
	Width=100%, L=50 R=50
	Width=0%, L=50 R=50
	Width=-100%, Position = 0 (center)
	Width=36%, L=44 R=56
	Width=0%, L=0 R=100

#### Using the mouse

Mouse operations in the upper half of the panner adjust the position parameter, constrained by the current width setting.

Mouse operations in the lower half of the panner adjust the width parameter, constrained by the current position setting.

The position can be changed smoothly, by pressing the right button and dragging within the top half of the panner, then releasing. The position will be limited by the current width setting. *Note: it is not necessary to grab the position indicator in order to drag.*

The width can also be changed smoothly, by pressing the right button and dragging within the lower half of the panner, then releasing. The width will be limited by the current position setting. *Note: it is not necessary to grab the L/R indicators in order to drag.*

Reset to defaults	Click right
Change to hard left	Double click right in the upper left half of the panner
Change to a hard right	Double click right in the upper right half of the panner
Move position as far left as possible, given width	Double click right in the upper left half of the panner
Move position as far right as possible, given width	Double click right in the upper right half of the panner
Set the position to center	Click right in the upper middle of the panner
Reset to maximum possible width	Double click right on the lower left side
Invert (flip channel assignments)	Double click right on the lower right side
Set width to 0°	Double click right in the lower middle

### Keyboard bindings

When the pointer is within a stereo panner user interface, the following keybindings are available to operate on that panner:

↑ / ↑	increase width by 1° / 5°
↓ / ↓	decrease width by 1° / 5°
← / ←	move position 1° / 5° to the left
→ / →	move position 1° / 5° to the right
0	reset position to center
↑	reset width to full (100%)

### Using the scroll wheel/touch scroll

When the pointer is within a stereo panner user interface, the scroll wheel may be used as follows:

/	increase width by 1° / 5°
/	decrease width by 1° / 5°
/	move position 1° / 5° to the left
/	move position 1° / 5° to the right

## 99.2 Stereo panning caveats

The stereo panner will introduce unwanted side effects on material that includes a time difference between the channels, such as A/B, ORTF or NOS microphone recordings, or delay-panned mixes.

When the width is reduced, two highly correlated signals with a delay are effectively summed, which will cause comb filtering.

Let's take a closer look at what happens when a source is recorded at 45° to the right side with an ORTF stereo microphone array and then the width manipulated.

For testing, we apply a pink noise signal to both inputs of an Ardour stereo bus with the stereo panner, and feed the bus output to a two-channel analyser. Since pink noise contains equal energy per octave, the expected readout is a straight line, which would indicate that our signal chain does not color the sound:

An ORTF is simulated using Robin Gareus' stereo balance control LV2 to set the level difference and time delay. The Trim/Gain can be ignored—its purpose is just to align the test signal with the 0dB line of the analyser.

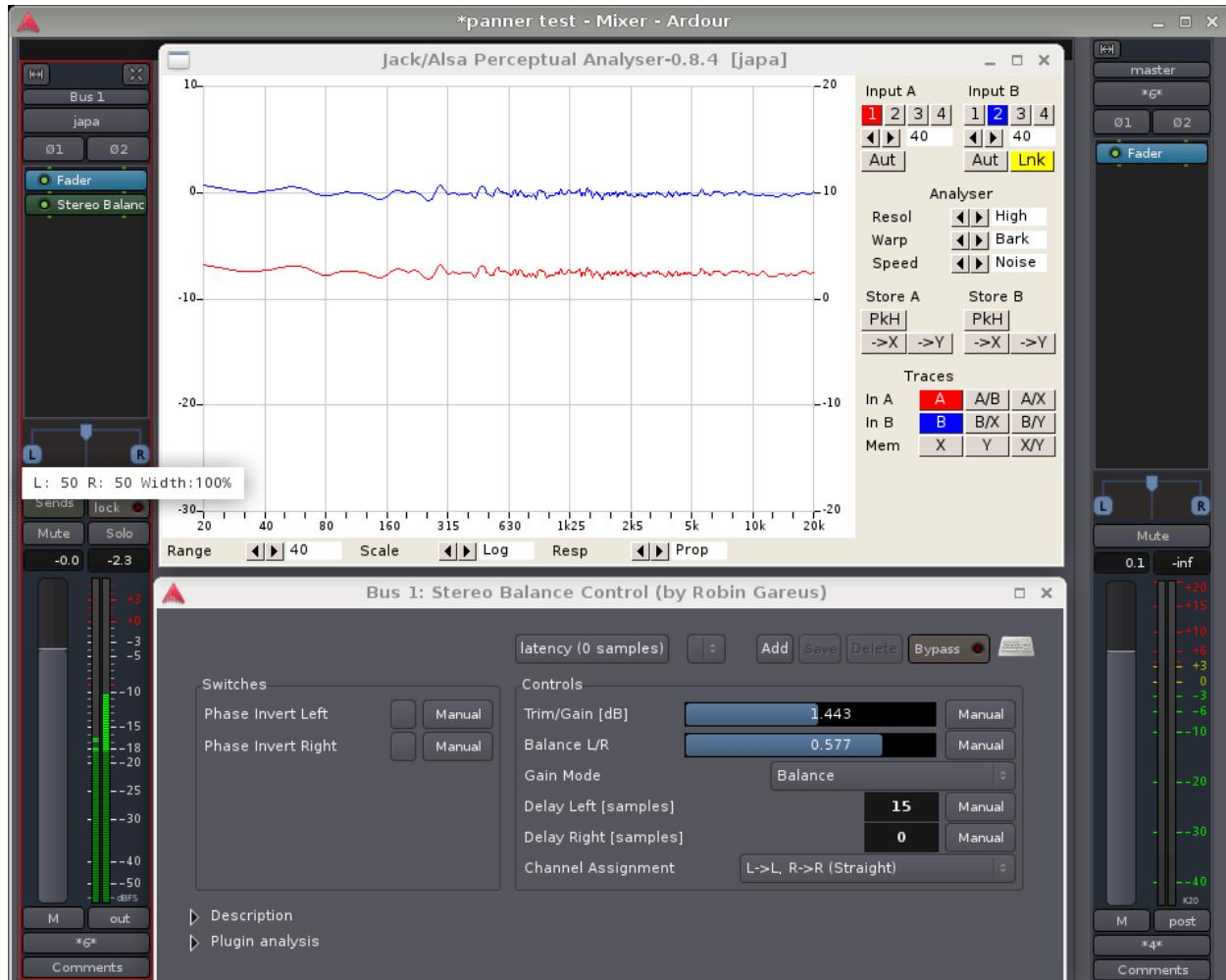


Fig. 2: Stereo panner with ORTF full width

An ORTF microphone pair consists of two cardioids spaced 17 cm apart, with an opening angle of  $110^\circ$ . For a far source at  $45^\circ$  to the right, the time difference between the capsules is  $350 \mu\text{s}$  or approximately 15 samples at 44.1 kHz. The level difference due to the directivity of the microphones is about 7.5 dB (indicated by the distance between the blue and red lines in the analyser).

Now for the interesting part: if the width of the signal is reduced to 50%, the time-delayed signals will be combined in the panner. What happens to the frequency response of the left and right outputs is shown in the following picture:

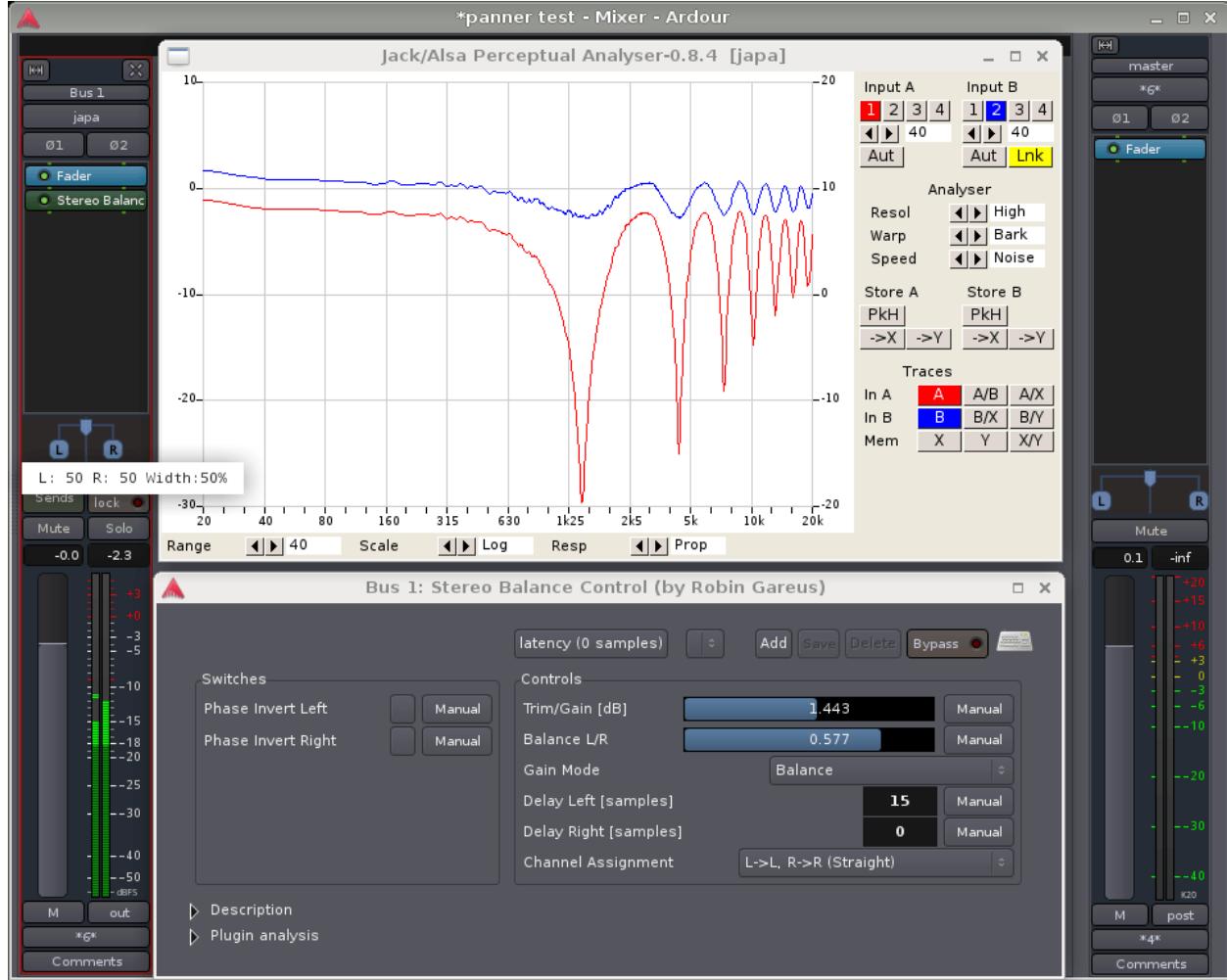


Fig. 3: Stereo panner with ORTF half width

It can be argued that all spaced microphone recordings will undergo comb filtering later, when the two channels recombine in the air between the speakers. Perceptually however, there is a huge difference: our hearing system is very good at eliminating comb filters in the real world, where their component signals are spatially separated. But once they are combined inside a signal chain, this spatial separation is lost and the brain will no longer be able to sort out the timbral mess.

Depending on the material and on how much the width needs to be manipulated, some degree of comb filtering may be acceptable. Then again, it may not. It is advised to listen carefully for artefacts when manipulating unknown stereo signals—many orchestra sample libraries for example do contain time-delay components.



## **68.16 - VBAP PANNER**

Ardour's VBAP panner is currently in development, and its semantics may change in the near future, possibly affecting mixes using it. It is advised not to rely on it for important production work while the dust settles.

The Panner only works in fixed static mode, it does not support automation playback.

VBAP is a versatile and straightforward method to pan a source around over an arbitrary number of speakers on a horizontal polygon or a 3D surface, even if the speaker layout is highly irregular.

### **100.1 Basic concepts**

VBAP was developed by Ville Pulkki at Aalto University, Helsinki, in 1997. It works by distributing the signal to the speakers nearest to the desired direction with appropriate weightings, aiming to create a maximally sharp phantom source by using as few speakers as possible:

- one speaker, if the desired direction coincides with a speaker location,
- two speakers, if the desired direction is on the line between two speakers,
- and three speakers in the general 3D case.

Thus, if the panner is moved onto a speaker, only this speaker will get any signal. This is handy when precise 1:1 routing is needed.

The drawback of VBAP is that a moving source will constantly change its apparent sharpness, as it transitions between the three states mentioned above.

An horizontal VBAP panner has one parameter, the azimuth angle. A full-sphere panner offers an additional elevation angle control.

More elaborate implementations of VBAP also include a spread parameter, which will distribute the signal over a greater number of speakers in order to maintain constant (but no longer maximal) sharpness, regardless of position. Ardour's VBAP panner does not currently include this feature.

### **100.2 Speaker layout**

Each VBAP panner is specific to its speaker layout—the panner has to “know” about the precise location of all the speakers. A complete VBAP implementation must therefore include the possibility to define this layout.

Ardour currently uses a simplified approach: if a track or bus has more than two output channels (which implies stereo), it assumes that there are N speakers distributed in a regular N-gon. That means that for irregular layouts such as 5.1 or 7.1, the direction dialed in will differ a bit from the actual auditory result, but any desired spatialisation can still be achieved.

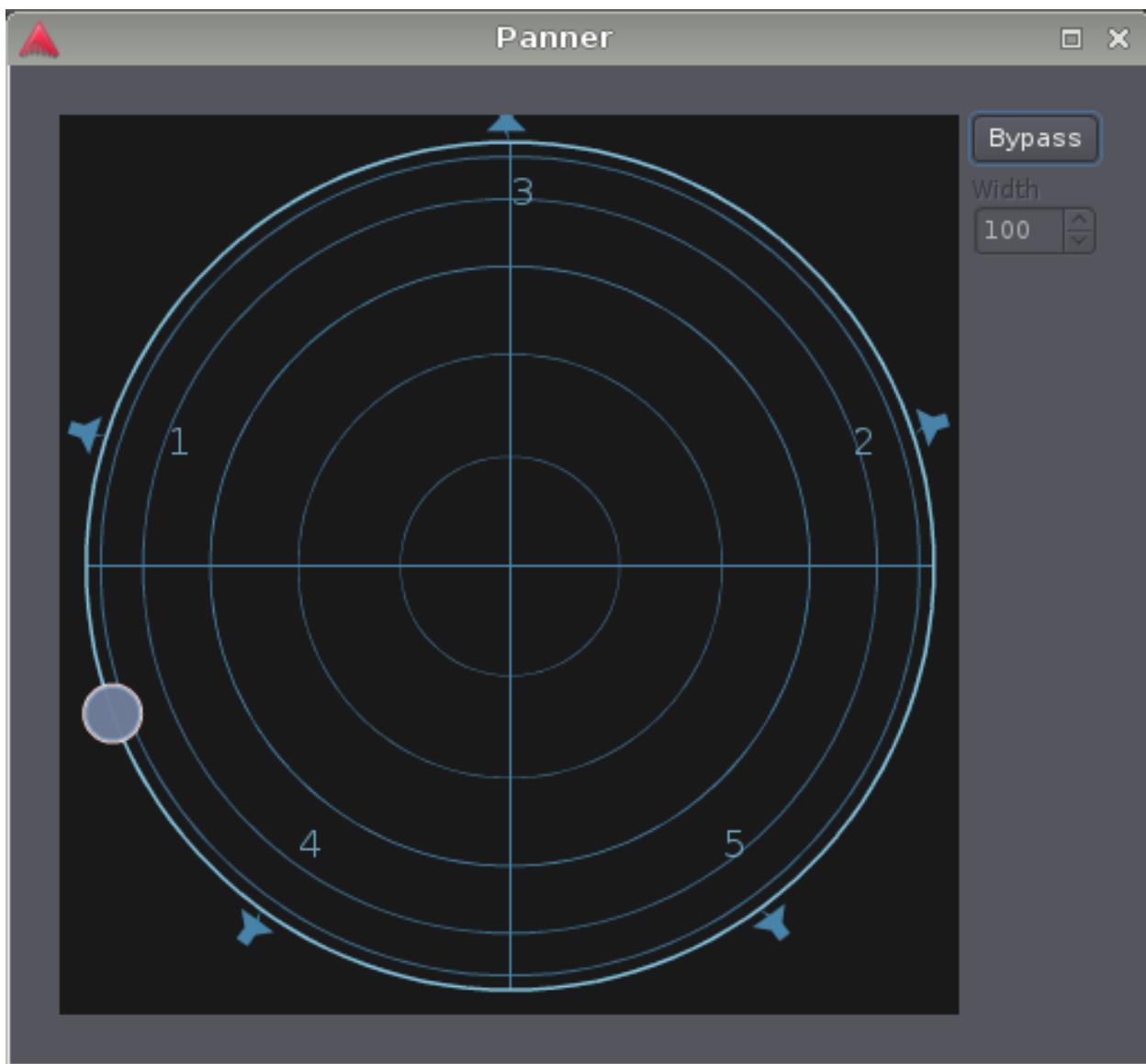


Fig. 1: The VBAP panner with 5 outputs

### 100.2.1 Experimental 3D VBAP

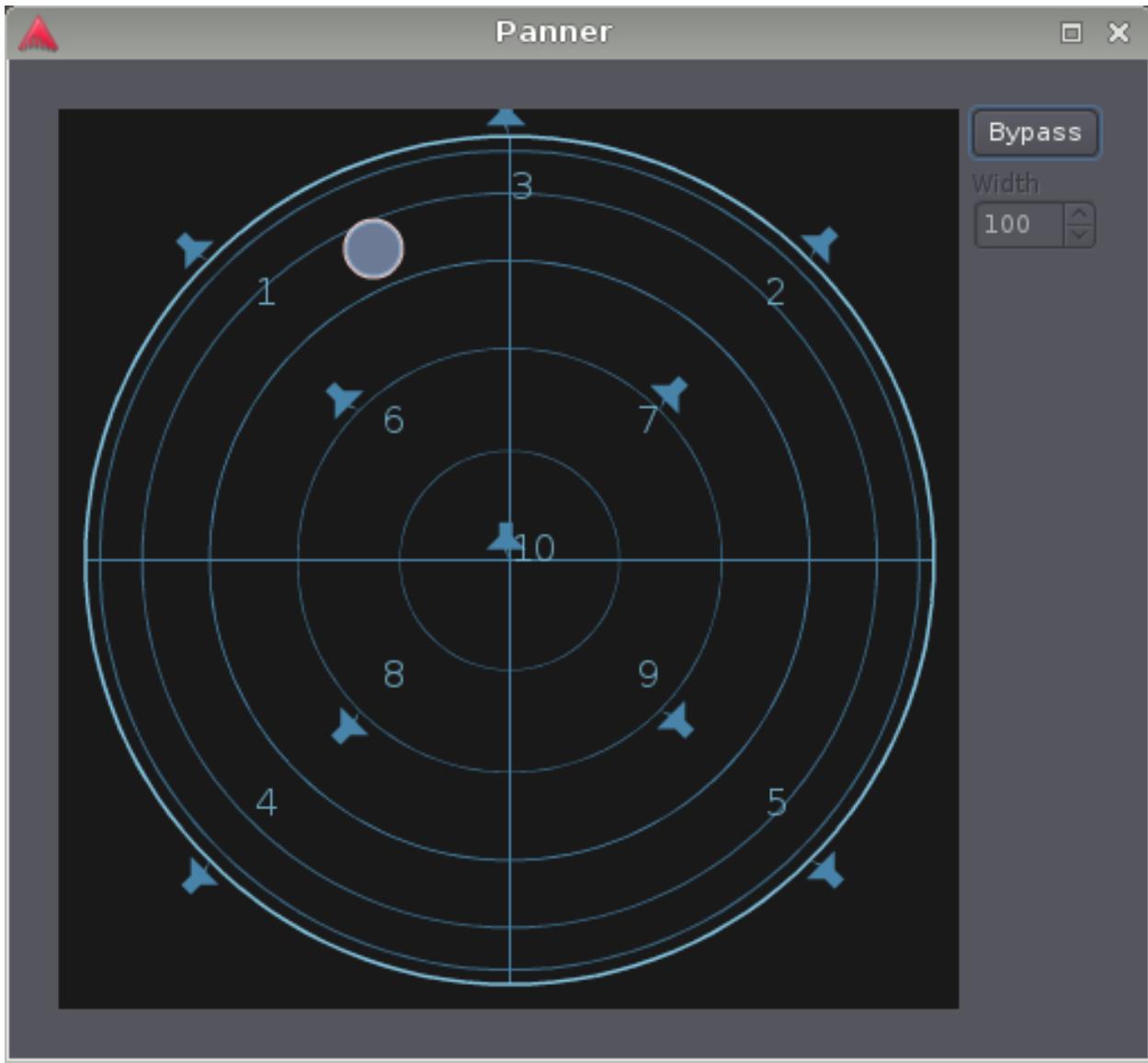


Fig. 2: The VBAP panner with 10 outputs, in experimental 3D mode

For tracks with 10 outputs, Ardour will currently assume a 3-dimensional speaker layout corresponding to Auro-3D 10.1, which is a horizontal 5.1 system, four elevated speakers above L, R, Ls, and Rs, and an additional “voice-of-god” speaker at the zenith.

## 100.3 N:M panning

For tracks and busses with more than one input, Ardour will (for now) assume that the inputs are distributed symmetrically along the latitude around the panner direction. The width parameter controls the opening angle of the distribution sector.

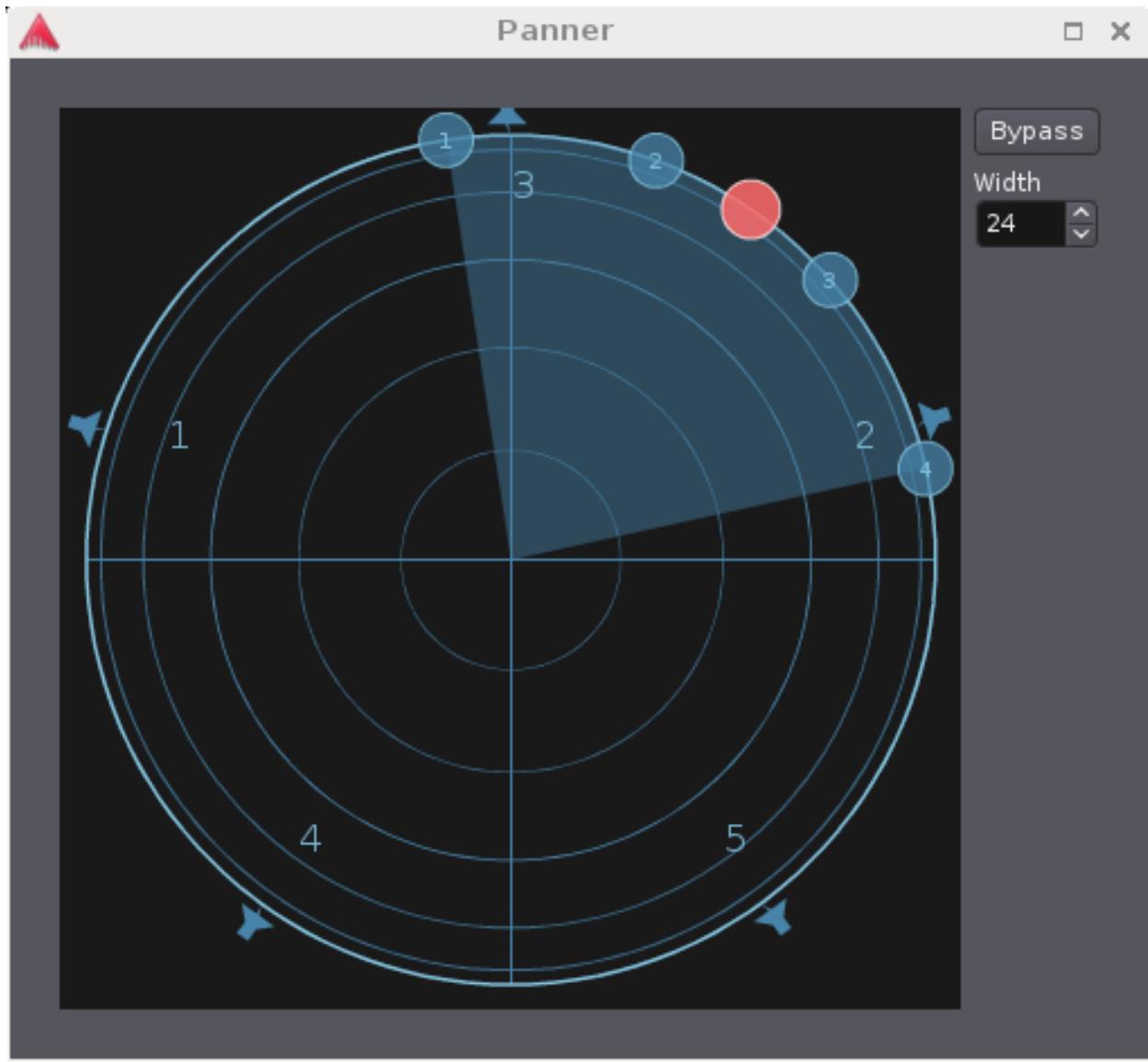


Fig. 3: The VBAP panner in 4 in, 5 out mode

---

CHAPTER

ONE

---

## 69 - PLUGIN AND HARDWARE INSERTS



---

## 69.1 - WORKING WITH PLUGINS

---

Plugins are bits of software that get loaded by Ardour in order to:

- Create various audio or MIDI effects
- Generate audio by functioning as “software instruments”

They are usually written by 3rd parties, though *a few come as part of a standard Ardour install*. The sources for plugins are many and varied; see [here](#) for some information on how to get them.

Ardour supports a variety of different plugin standards:

LADSPA	An early, simple, lightweight plugin API, audio effects only, plugins have no editors/GUI of their own (Ardour provides one, however).
LV2	An extensible, full-featured plugin API, audio and MIDI, plugins can provide their own GUIs but may use the one Ardour provides instead.
AU	OS X only, full featured, audio and MIDI, plugins can provide their own GUI
VST	Plugins using Steinberg’s VST plugin standard. Varies by platform: + + + +   on Linux   (native)       Linux VST       plugins       fully       supported       (VST2.4)   + + + + + + +   on Windows   (native)       Windows VST       plugins       fully       supported       (VST2.4)   + + + + + + +   on OS X   (native)       macOS VST       plugins       fully       supported       (VST2.4)       since Ardour       5.5   + + + +
Windows VST Plugins on Linux	VST plugins for Windows, but being used on Linux. <i>Not supported by normal builds of Ardour.</i> <a href="#">Read more...</a>

### 102.1 Adding/Removing/Copying Plugins

Within Ardour, plugins are just another type of Processor and so the techniques for adding/removing/copying/moving processors apply to plugins as well. These techniques are covered on the *Processor Box* page.



## 69.2 - PROCESSOR BOX



Fig. 1: Processor Box.

In Ardour terminology, a processor is anything which treats the signal in some way and gets plugged into a mixer strip. Ardour provides several builtin processors such as the fader or panners. Processors can also be plugins used for effects or as instruments, as well as sends or inserts which affect *signal routing*.

The arrangement of processors is arbitrary, and there is no limit to how many there can be. The Processor Box will automagically add a scrollbar to itself if there are more processors in it than can be shown in the given space.

The main box in the top half of a mixer strip shows the processor box. Processors are shown as colored rectangles, with a small LED beside them that lights up when the processor is enabled. The color of the processor depends on its location in the sequence; processors that are pre-fader are colored in red, and post-fader processors are colored green (in the default theme).

The processor box will always contain a blue Fader processor. This indicates where in the processor chain the main channel fader is located; this is the fader shown in the lower half of the strip. It can be enabled and disabled like any other processor.

## 103.1 Adding Processors

Processors can be added to the chain by Right-clicking in the processor list. This does three things:

- A gap is opened up to indicate the location of the click. The gap shows where any new processors will be inserted.
- The processor under the click is selected.
- An options menu is presented.

From the menu, new processors can be inserted.

Processors can also be dragged and dropped from the *Favorite Plugins window* to an appropriate spot in the Processor Box.

The Favorite Plugins window can be populated via the *Plugin Manager*, or by dragging and dropping an existing processor from the processor box to the Favorite Plugins window.

## 103.2 To Reorder (Move) Processors

Processors can be re-ordered using drag and drop. Dragging a processor allows it to be moved around within the chain, or copied to another processor list on another track or bus.

## 103.3 To Enable/Disable a Processor

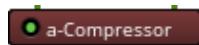


Fig. 2: A typical processor.

To the left of the name of each processor is a small LED symbol; if this is lit-up, the processor is active. Clicking on it will deactivate the processor and effectively bypass it.

Some processors have their own bypass controls that are independent of the one that Ardour provides; this can make it appear that the plugin is non-responsive when its independent bypass control is active.

## 103.4 Selecting Processors

A processor in the processor box can be selected with a Left-click on it; it will be highlighted in red. Other processors can be selected at the same time by Left-clicking on them while holding down the key, and ranges can be selected by Left-clicking on them while holding down the key.

## 103.5 Removing Processors

Context-click on the processor to be removed, and select Delete; or Right-click on it; or Left-click on it and press the Delete key. If multiple processors are selected, they will all be deleted at the same time.



## 69.3 - PLUGIN MANAGER

The Plugin Manager serves two purposes. Primarily it is used to control the display status of plugins. It can also be used to find and insert plugins into the *Processor Box*. It is displayed either by a double-click in the Processor Box or by choosing New Plugin > Plugin Manager... from the Processor Box context menu.

Displayed for each plugin is the status (normal, favorite, hidden), name, type, category, creator (author), and the number of audio and MIDI connections. The plugins can be sorted by clicking on a column header.

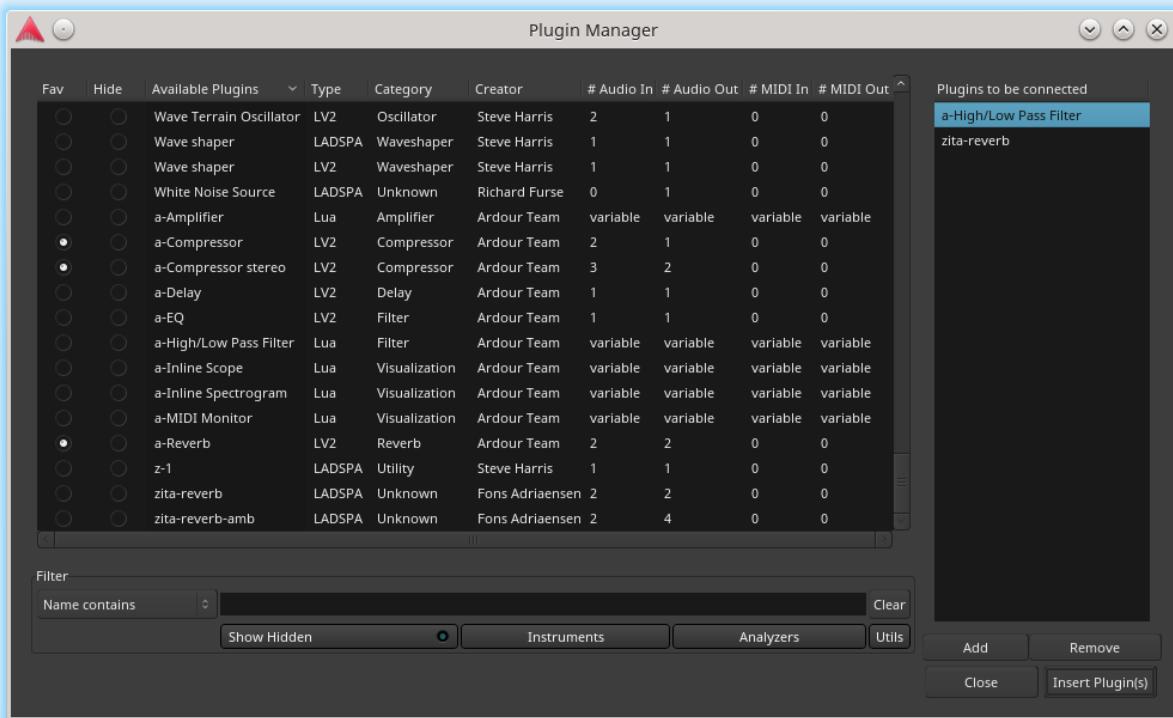


Fig. 1: The Plugin Manager window.

### 104.1 Plugin Display Status

Click on a Fav(rite) or Hide radio button to change a plugin's display status. Clicking on an already selected radio button will cancel it, returning the plugin to the normal display status. Plugins marked as a favorite show up in the Processor Box context menu under New Plugin > Favorites. Setting the hide radio button

on a plugin will keep the plugin from showing in the Processor Box context menus New Plugin > By Creator or New Plugin > By Category.

## 104.2 Filtering Listed Plugins

The bottom left part of the Plugin Manager is used to filter the listed plugins. Typing into the text-box will filter the plugins based on the filter mode selected by drop-down box. Clicking Clear empties the text-box.

## 104.3 Inserting Plugins in the Processor Box

The right part of the plugin manager shows plugins that have been selected for insertion into the Processor Box. A plugin can be added by either double clicking the plugin entry in the top left part, or, if already selected in top left part, by clicking Add.

Plugins can be removed from the right part with a double click, or, if already selected, by clicking Remove.

## 69.4 - MANAGING PLUGIN PRESETS

All plugin control widgets, whether they are created by Ardour or by the plugin, have a common set of controls at the top of the window. These include 4 controls for managing plugin presets.

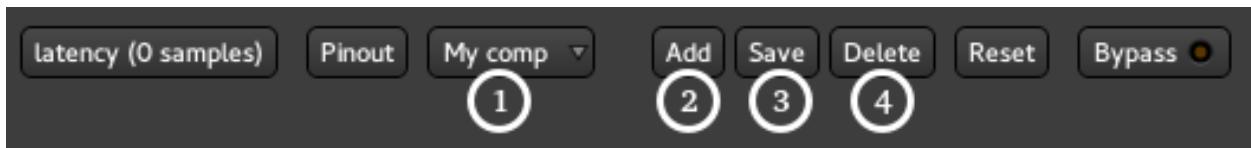


Fig. 1: The plugin presets toolbar.

### 105.1 What Is a Plugin Preset?

A preset for a plugin is simply a saved set of values for all of a plugin's parameters. If you load a preset, you are restoring all the parameters of that plugin to the values stored in the preset. This is an easy, fast way to manage your preferred settings for particular plugins.

### 105.2 The Preset Selector

The preset selector (1) is a regular selector that can be clicked to display a list of all known presets for this plugin. This will include presets that have been created by the user, and for some plugin formats, presets that come with the plugin itself.

### 105.3 Loading a New Preset

Clicking on the preset selector pops up a menu showing the names of all available presets. Clicking on the name of a preset loads it, and various controls in the plugin editor change to reflect the new value of some or all parameters.

### 105.4 Creating a Preset

Saving the current plugin settings as a new preset is done by clicking on the Add button (2) at the top of the window. A dialog will appear asking for a name for the preset.

## 105.5 Saving a Preset

To modify the settings in an existing preset, the preset selector must be used to load the preset, then, when the settings have been adjusted, the Save button (3) clicked. The new values will be stored, overwriting the previous version of this preset.

## 105.6 Deleting a preset

Deleting an existing preset is achieved by loading the preset first, then clicking the Delete button (4). The preset will be removed, and the preset selector turns blank, showing that no preset is currently loaded (although the settings will stay as they were).

## 69.5 - WORKING WITH ARDOUR-BUILT PLUGIN EDITORS

The plugin editor can be shown by double-clicking on the plugin within the *processor box*. A new window will appear showing the editor/GUI for the plugin.

### 106.1 Generic Plugin Editor

If a plugin does not have its own GUI, Ardour will construct a generic plugin editor from a small set of common control elements. Ardour will do this even for plugins that have their own, if Edit > Preferences > GUI > Use Plugins' own interface instead of Ardour's is disabled.

The generic UI can be temporarily switched to by right clicking on a processor and selecting Edit with generic controls. This is necessary in order to access the *plugin automation controls*.

In the generic UI, any controller can be reset to its default state by Left-clicking on it.

### 106.2 Analysis Graph

At the bottom of the generic plugin editor, clicking the arrow displays the Analysis Graph.

This graph displays:

- the transfer function in white,
- the phase response in red (optional),
- the post effect spectrum in green.

The transfer function plots the output amplitude of the plugin (considered as a “black box”) against its input amplitude, along the audio spectrum.

The phase response, that can be switched on or off using the Show phase checkbox, plots the phase of the plugin’s output against its input phase, along the audio spectrum. The scale is shown in yellow on the right.

The green spectrum plots the output signal spectrum, after the plugin (for tracks that have a signal on).

The dB scale selector in the bottom left allows to change the vertical scale of the graphs.

### 106.3 MIDI instruments specificities

The generic UI provides, for all MIDI instruments plugins, a keyboard, that can be used either with the mouse, or by using a QWERTY keyboard as a piano. Both the channel and the velocity can be set above the keyboard.

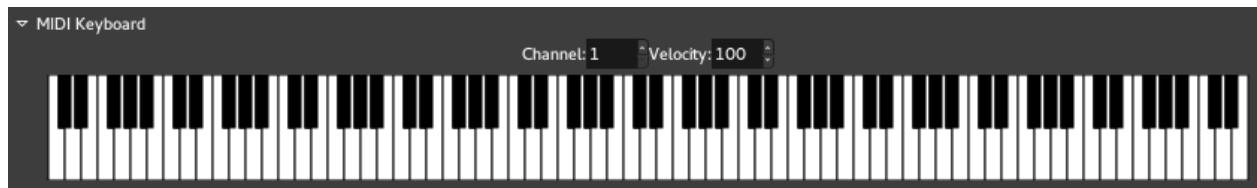


Fig. 1: The MIDI keyboard in instruments plugins

---

CHAPTER  
SEVEN

---

## 69.6 - PLUGINS BUNDLED WITH ARDOUR

Ardour does not come with any built-in signal processors of its own (other than volume faders) but does ship, since v5.0, with the small group of plugins listed below. These plugins are listed as authored by “Ardour Team”, which are LV2 plugins, and are named with “a-” as the start of the name (like a-EQ) or are listed as authored by “Ardour LUA Task Force” in which case they are example (but still useful) LUA scripts. These plugins use Ardour’s generic GUI, and they work on all supported platforms so that projects started on one platform will sound the same on another platform—if they use just these plugins.

a-Amplifier	A versatile $\pm 20\text{dB}$ multi-channel amplifier
a-Compressor	A side-chain enabled compressor with the usual controls. Comes in stereo and mono versions
a-Delay	A basic single-tap delay line, with tempo sync
a-EQ	A nice sounding 4-band parametric EQ with shelves
a-Fluid Synth	Wraps the Fluidsynth SoundFont2 synthesis engine as a new sample player
a-High/Low Pass Filter	Independent high and low pass filters with steepness up to 48dB per octave
a-Inline Scope	A mixer strip inline waveform display
a-Inline Spectrogram	A mixer strip inline spectrum display
a-MIDI Monitor	A mixer strip inline display to show recent MIDI events
a-Reverb	A reverb that finds a balance between sounding good, using a lot of CPU and having too many controls



## 69.7 - GETTING MORE PLUGINS

The following list shows a few plugin packages. In some cases, a package contains just one or two plugins; in other cases, dozens.

This list does not aim at being exhaustive.

### 108.1 Plugins by Standard:

#### 108.1.1 LV2

- SWH <http://plugin.org.uk/lv2/> [GNU GPLv3]
- ll-plugins <http://ll-plugins.nongnu.org/> [GNU GPLv3]
- ZynAddSubFX <http://zynaddsubfx.sourceforge.net/> [GNU GPLv2+]
- OvertoneDSP <https://www.overtonedsp.co.uk/> [Proprietary]
- Invada Studio <https://launchpad.net/invada-studio/> [GNU GPLv2]
- Pianoteq <https://www.pianoteq.com/> [Proprietary]

#### 108.1.2 LADSPA

- Kokkini Zita <http://kokkinizita.linuxaudio.org/linuxaudio/> [GNU GPL/GNU GPLv3]
- Blepvco <http://www.smbolton.com/linux.html> [GNU GPLv2+]
- Blop <http://blop.sourceforge.net/> [GNU GPLv2]
- CAPS <http://quitte.de/dsp/caps.html> [GNU GPLv3]
- CMT <http://www.ladspa.org/cmt/overview.html> [GNU GPLv2]
- FOO <https://github.com/sampov2/foo-plugins> [GNU GPLv2]
- NJL <https://github.com/tialaramex/njl-plugins> [GNU GPLv2]
- Omnis <http://www.nongnu.org/om-synth/omnis.html> [GNU GPLv3]
- SWH <http://plugin.org.uk/> [GNU GPLv3]
- TAP <http://tap-plugins.sourceforge.net/> [GNU GPLv2]
- VCF <http://www.suse.de/~mana/ladspa.html> [GNU LGPL]
- VLevel <http://vlevel.sourceforge.net/> [GNU GPLv2]

- Vocoder [http://www.sirlab.de/linux/download\\_vocoder.html](http://www.sirlab.de/linux/download_vocoder.html) [GNU GPLv2+]
- WASP <http://linux01.gwdg.de/~nlissne/wasp/index.html> [GNU GPLv3]
- Nova <http://chuck.stanford.edu/planetccrma/mirror/fedora/linux/planetccrma/10/i386/repoview/ladspa-nova-plugins.html> [GNU GPLv2+]
- Socal's LEET Plugins <http://code.google.com/p/leetplugins/> [GNU GPLv2]

### 108.1.3 Linux VST (LXVST)

- Loomer <http://www.loomer.co.uk/> [Proprietary]
- Distrho <http://distrho.sourceforge.net/ports.php> [GNU GPLv3]
- Argotlunar <http://argotlunar.info/> [GNU GPLv2]
- U-he <https://u-he.com/> [Proprietary]

## 108.2 How to install plugins?

### 108.2.1 Linux

Installation will vary a little depending on how the plugins have been obtained. If a particular plugin package appears in the local repository, installing it using is done by using the normal software package management tool for the system. Most Linux distributions that are good for audio work will have most of the LADSPA and LV2 plugins mentioned above available in ready-to-use form.

Finding them will typically require *searching* the distribution's repository to find the name of the package. The tools for doing this vary from distribution to distribution. A good place to start searching is with the name of the package (e.g. "caps" or "cmt"). There are no fixed rules about what different Linux distributions call their packages for a given set of plugins.

If the package isn't available, then the plugins can be built from source (plugins are generally fairly easy to compile and well-documented).

LADSPA plugins are shared library files. They need to be installed in either /usr/lib/ladspa, /usr/local/lib/ladspa or in a directory mentioned in the local LADSPA\_PATH environment variable.

LV2 plugins are folders/directories. They need to be installed in either /usr/lib/lv2, /usr/local/lib/lv2 or a directory mentioned in the local LV2\_PATH environment variable.

Linux VST (LXVST) plugins are distributed as shared library files. They are typically installed in /usr/lib/lxvst, /usr/local/lib/lxvst or a directory mentioned in the local LXVST\_PATH environment variable.

### 108.2.2 OS X

Except for the particularly technical computer user, building and installing plugins in the LV2 (or LADSPA) format is probably not something worth planning on.

Most of the plugins likely to be used on OS X will be in Apple's AudioUnit format. These have their own installation process that tends to just work.

## **69.8 - USING WINDOWS VST PLUGINS ON LINUX**

Thanks to the combined work of Torben Hohn, Kjetil Mattheusen, Paul Davis and a few other developers, it is possible to use Windows VST plugins (that is, plugins in VST format built and distributed for the Windows platforms) on Ardour running on Linux.

However, doing so has three *substantial* downsides:

- It requires a special build of Ardour that is fundamentally very different from normal builds
- Support depends on [Wine](#), a Windows “emulator”
- As usual with plugins, a crashing plugin will take Ardour down with it—and crashes in Windows VST plugins are more likely when used in this way

The dependence on Wine makes it almost impossible for the Ardour project to support this feature. Wine’s functionality generally improves over time, but any given release of Wine may behave worse with some or all Windows VST plugins. It may even just crash Ardour completely.

Step back and think about what “using Windows VSTs” really means: taking bits of software written with only one idea in mind—running on the Windows platform—and then trying to use them on an entirely different platform. It is a bit of a miracle (thanks largely to the incredible work done by the Wine project) that it works at all. But is this the basis of a stable, reliable DAW for a non-Windows platform? Getting Ardour on Linux to pretend that its really a Windows application running on Windows?

It is understandable that there are many outstanding plugins available as Windows VSTs and, that in many cases, no equivalent is available for Linux. If a workflow is so dependent on those plugins, Ardour should be used on Windows (or potentially used with an actual Windows VST host running inside of Wine). If the effort can be made, a better environment can be obtained by using a normal build of Ardour and exploring the world of plugins built to run on Linux natively. This covers LADSPA, LV2 and Linux VST formats, and even some outstanding proprietary plugins such as those from [Loomer](#).

### **109.1 A Plea To Plugin Manufacturers**

Please consider porting your plugins so that users can enjoy them on Linux too. Several other commercial plugin developers have already done this. You can choose between using “Linux VST” (which is what Loomer and others have done)—you will find toolkits like JUCE that help to make this fairly easy—or using LV2 format which is ultimately more flexible but probably requires more work. We have users—thousands of users—on Linux who would like to use your plugins.



## **70 - AUTOMATION**

Automation is the ability to dynamically control various aspects of a track's innate attributes and the attributes of any processors attached to it. In Ardour, automation can be used to make dynamic changes to a track's:

- Volume
- Panning
- Trim
- Muting
- Any attached processor's parameters

Any combination of these can be enabled on a single track; as such, it offers a lot of power and flexibility over how a track will ultimately sound when played back.



## **70.1 - AUTOMATION NOMENCLATURE**

Track automation occurs in one or more lanes. Each lane has a control that allows setting the amount or position of a certain parameter associated with the lane. Parameters are things that can be controlled on a track's automation lane, such as volume, panning, muting, trim, etc. Automation curves consist of lines connected by control points, that live within the confines of a lane; these tell Ardour how to change a given parameter over time. Automation modes govern how a given automation lane will behave during playback.



## 70.2 - AUTOMATION MODES

In order to understand how automation in Ardour works, it is necessary to understand the four modes of automation. They are: Manual, Play, Write, and Touch.



Fig. 1: The automation mode menu.

Manual mode is basically analogous to a processor's bypass switch. Whenever an automation lane is in this mode, it is inactive and any level that is manually set for controlling the lane's parameter will persist during playback like normal.

In Ardour, every track and processor parameter is initially set to Manual mode.

Play mode tells Ardour to use the automation curve in the automation lane to control the level of the parameter controlled by the lane *during playback*. The control that normally sets the parameter will be *unresponsive to manual input* and will move automatically in accord with the lane's automation curve during playback.

Write mode allows continuous, dynamic setting of a control during playback; all such settings are written to the lane the control is in. This defines the lane's automation curve in the interval being played, and overwrites any existing automation curve in the lane being manipulated.

Touch mode is similar to Write mode, except it only overwrites sections of a lane's automation curve when the control is changed in some way. This allows for changing only the parts of an automation curve that are desired to be changed, while leaving the rest unchanged.



## 70.3 - AUTOMATION LANES



Fig. 1: A typical automation lane.

An automation lane is similar to a track in that it holds data that can be played back; however, unlike a track, it is not an independent entity—it is always attached to the track that it controls. Automation lanes also contain zero to one automation curves. Each lane controls one and only one parameter of the track it is attached to.

Every track will have at least five automation lanes associated with it: trim, fader, mute, and pan (which consists of two lanes: L/R and Width); it can possibly have many more if there are any processors associated with it. All these lanes are automatically attached to the track but hidden, and initially they are all empty (have no automation curves in them).

Automation lanes typically have the following controls:

- A hide button (square button with an “X” inside)
- A horizontal fader
- An automation mode selector

The hide button, as the name implies, hides the automation lane. The horizontal fader controls the level of the parameter that the lane controls; manipulating this while in Write or Touch mode during playback will make changes to the lane’s automation curve. The automation mode selector selects which mode the lane is in (Manual, Play, Write, or Touch).

The hide button will only hide the lane; it does not remove it from the track. The automation lane never really goes away—the closest one can get to that is to clear the automation curve and hide the lane.



## 70.4 - AUTOMATION CURVES

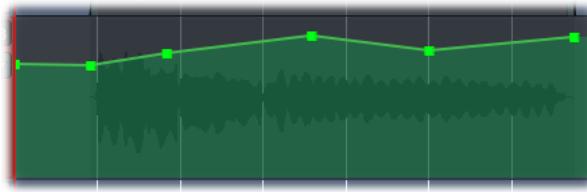


Fig. 1: A typical automation curve.

An automation curve is a series of lines connected by control points that defines a continuous line. As the curve is traversed from left to right, the line defines the level of the parameter controlled by the automation lane.

The curve by itself does nothing; it will *only* control playback if the lane it is in is in Play mode.



## 70.5 - CONTROLLING A TRACK WITH AUTOMATION



Fig. 1: The automation menu.

To automate a parameter on a given track, click on the track's A button and select a parameter to control from the menu that appears. Once a parameter has been selected, an automation lane for that parameter will appear beneath the track. The lane thus shown will be empty; from here an automation curve must be defined.

If the height of the automation lane is too small to see all of its controls, the height can be increased by Left clicking on the bottom border of the lane and dragging it.

There are three ways to define an automation curve:

- Record it using Write mode
- Record it using Touch mode
- Draw it using the mouse

### 115.1 Recording an Automation Curve Using Write Mode

To create an automation curve using Write mode, first set the lane's mode selector to Write, then set the playhead to the position where the automation curve should start, then set the transport to play. While the playhead is moving, Ardour will continuously record any changes made with the lane's fader. Even if no changes are made to the fader, they will overwrite anything that existed in the lane where the playhead is moving. When the desired automation curve has been recorded, stop the transport.

After the transport is stopped, the lane's mode selector will automatically switch to Touch mode—it is generally a bad idea to leave an automation lane in Write mode, as it is a destructive operation that makes it easy to inadvertently overwrite existing automation curves.

## 115.2 Recording an Automation Curve Using Touch Mode

Creating an automation curve using Touch mode is similar to the method employed in creating one using Write mode; the only difference is that changes are written to the automation curve *only* when the lane's fader is moved—at all other times, whatever was in the automation curve will remain as it was.

Touch mode is useful when only small parts of the automation curve need touching up versus Write mode, which is usually used to create the automation curve in the first place.

## 115.3 Drawing an Automation Curve Using the Mouse

In Draw mode, control points can be entered in the automation lane by Left-clicking in the lane at a point where there is no existing control point.

Once added, a control point can be Left-clicked and dragged to a desired location. Hovering over a control point will show its current level in dB. To remove a control point, Left-click it and press Delete, or Right-click on it.

## 115.4 Controlling the Track

Once an automation curve has been defined through any of the methods outlined above, the track won't do anything with it until the lane that the curve was defined in is set to Play mode. Then, during playback, as the playhead moves through the automation curve, the lane's control will move in accord with the curve.

The lane's fader will *not* be responsive to manual input while it is in Play mode.

---

**CHAPTER  
SIX**

---

**71 - MIXDOWN**



## 71.1 - EXPORT DIALOG

When the work in Ardour is finished, one or multiple sound file(s) need to be created, be it to be printed to a medium such as a CD or DVD, uploaded to a streaming site or sent to another person or software for further work. This can be done either using Session > Export > Export to Audio file(s)..., or Session > Export > Stem Export....

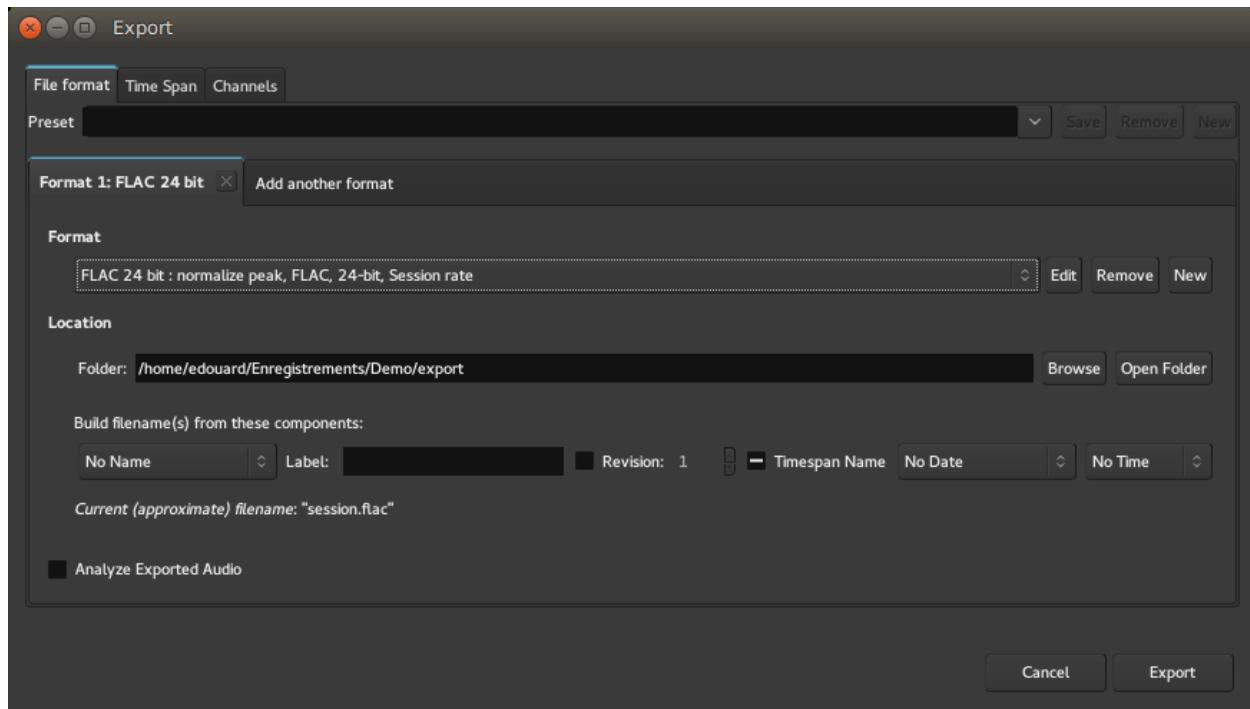


Fig. 1: The Export window

### 117.1 File Format

This tab contains controls for the format of the exported audio file(s). More than one format can be enabled here, in which case each will be exported in turn. Ardour is supplied with a list of export formats, including:

- BWAU 32float
- CD (Red Book)
- DVD-A

- FLAC 24 bit
- FLAC 24 bit (tagged)
- Ogg/Vorbis
- Ogg/Vorbis (tagged)
- Ring Tone

These formats can be edited, or new ones created, with the “*Edit Export Format Profile*” dialog, which appears when clicking the Edit or New buttons to the right of the drop-down list of formats.

Presets can also be created, consisting of one or more formats. Ardour provides some ready-made presets, too:

- CD + DVD-A
- CD + FLAC
- CD + FLAC (tagged)
- CD + Ogg/Vorbis + FLAC (tagged)
- CD + Ogg/Vorbis
- CD + Ogg/Vorbis (tagged)
- CD only
- DVD-A only
- FLAC
- FLAC (tagged)
- Ogg/Vorbis + FLAC
- Ogg/Vorbis + FLAC (tagged)
- Ogg/Vorbis
- Ogg/Vorbis (tagged)

## 117.2 The location

Aside from providing a way to tell Ardour *where* to put the created file(s), the location part of the window allows to name the exported files with a lot of choice regarding the naming convention, hence blending into the user’s workflow, and providing a clean way to keep the export folders from being cluttered with poorly named files.

The name of the file(s) can optionally be made of:

- The session or snapshot’s name
- A custom label (i.e., any text)
- A revision number
- The name of the timespan (see below)
- A date (in multiple formats)
- A time (also in multiple format).

As in the screenshot above, when writing a file could erase a present file with the same name, Ardour shows a yellow warning line in the bottom of the window, and a button to list all the files that would be erased and replaced.

### 117.3 Analyze exported audio

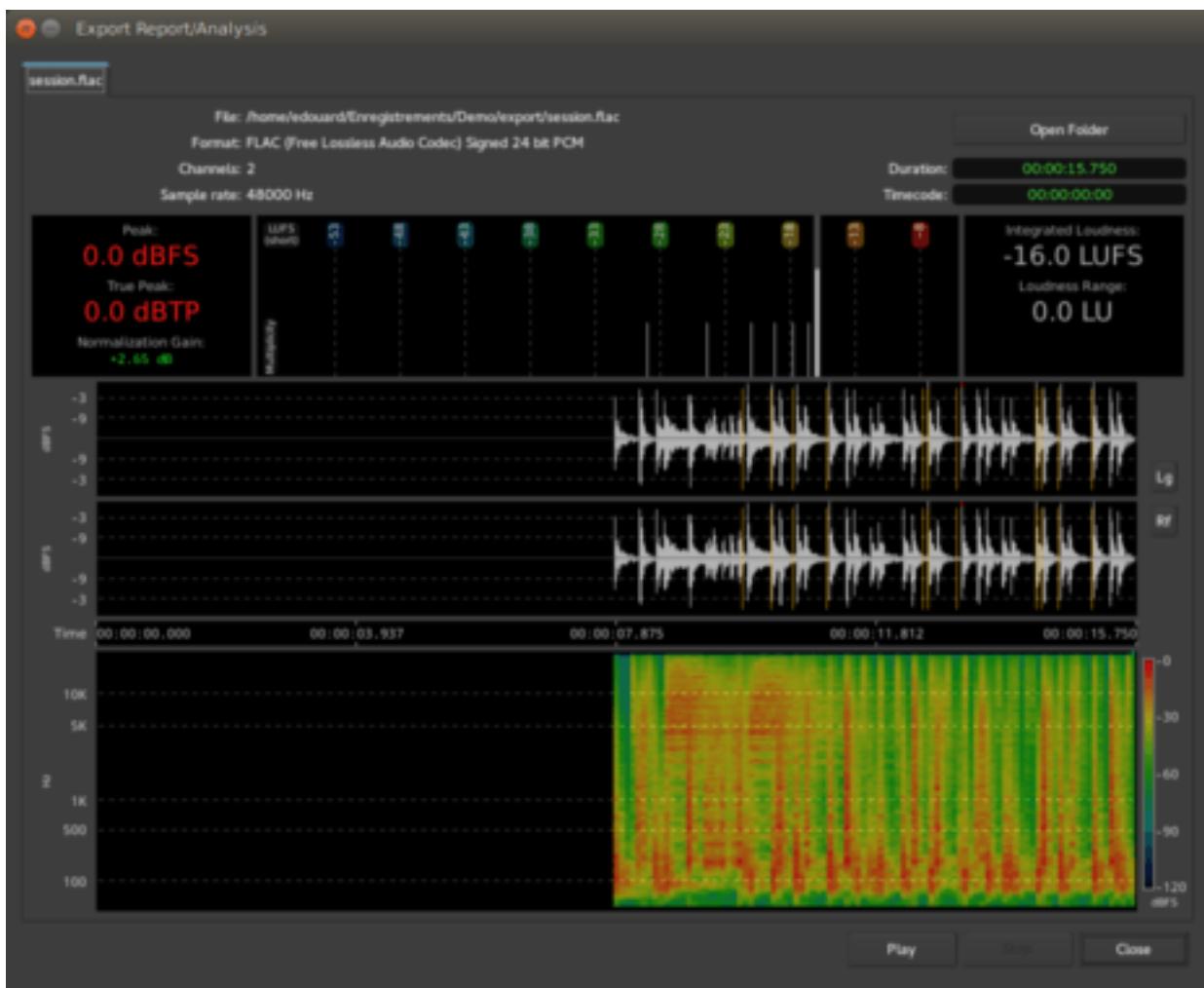


Fig. 2: The Export Report/Analysis window

Checking Analyze Exported Audio shows the Export Report/Analysis window. This provides a lot of useful information about the exported file:

- the file name and location
- its format
- its channel count
- its sample rate
- its duration and timecode.

It also allows to Play the file, and the Open Folder button gives a quick access to the place where it has been created.

The most prominent feature though, are the two generated views of the audio file in time (waveform) and frequency (sonograph) domain, and the loudness analysis, giving:

- the Peak value
- the True Peak value (to take inter sample peaks into account)
- the Normalization Gain (if it has been applied)
- the Integrated Loudness
- the loudness range
- a graph of the multiplicity of the peaks at the different loudness levels.

## 117.4 Time Span

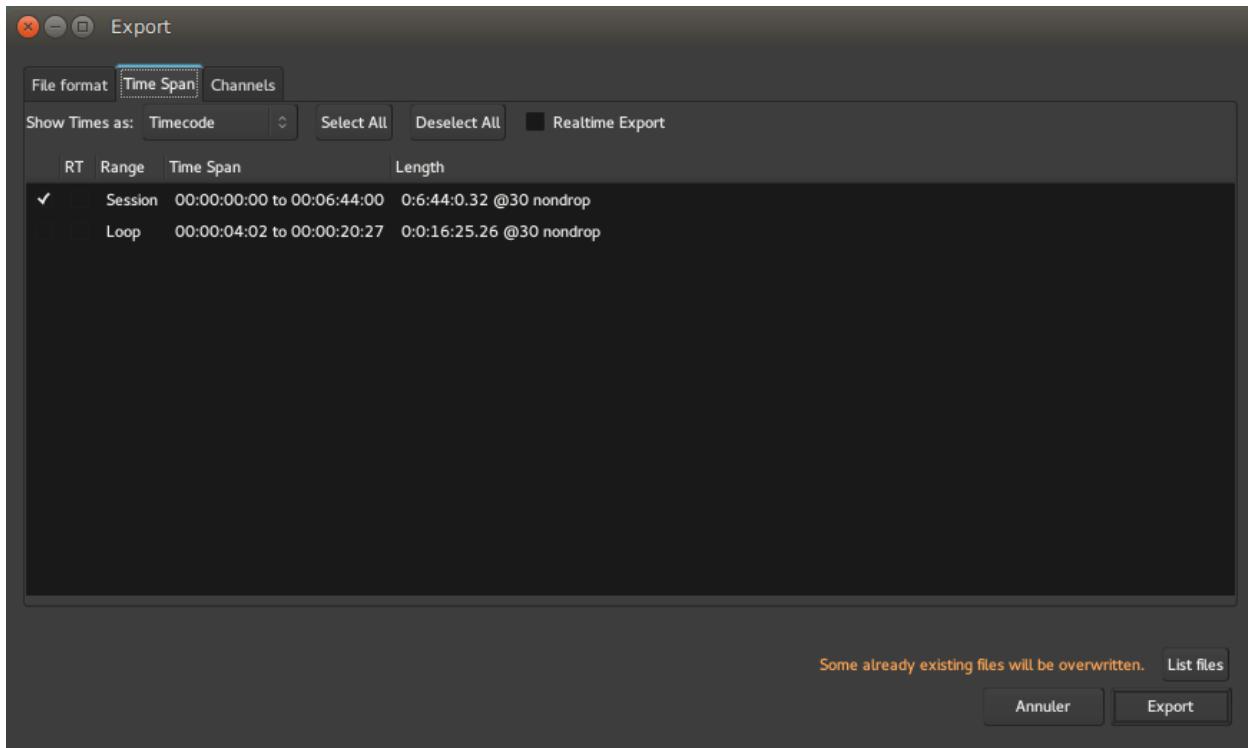


Fig. 3: The Time Span tab

This tab allows to select the range (or ranges) of the timeline to export. By default, “session” is enabled—this will export the whole session from the start marker to the end marker. Any loop or range present in the session can be chosen, or a combination thereof.

The realtime checkboxes allow to export audio as it is played, and not freewheeling to render the file as fast as Ardour can. This can prevent odd behaviours from some plugins (reverbs, etc...). This can be chosen globally (with the Realtime Export checkbox at the top) or individually on a per time span basis, with the RT checkbox next to each time span.

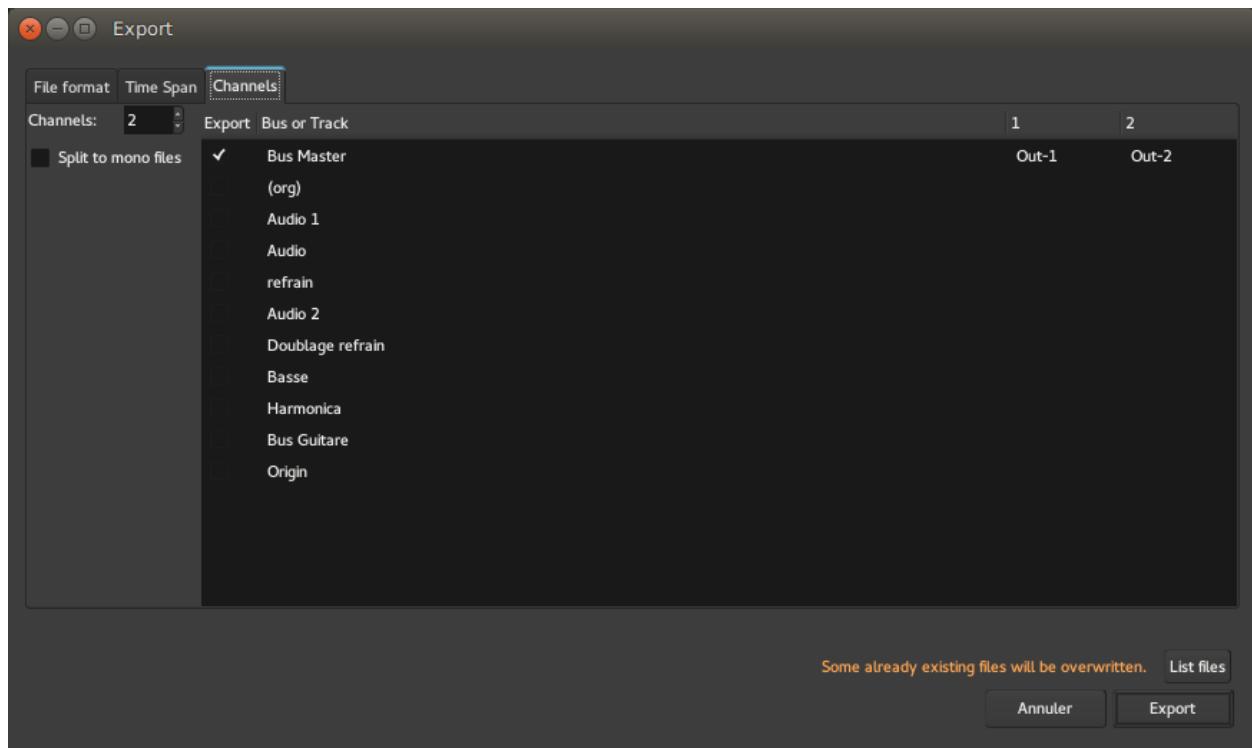


Fig. 4: The Channels tab

## 117.5 Channels

This tab decides which outputs (tracks or busses) should be sent to the exported file. By default, only the Master Bus is sent.

## 117.6 Stem Export

Stem exporting allows to transfer files between different systems and softwares by exporting each track individually, including silence, to keep them in sync.

If ‘Stem Export’ is chosen, the ‘Channels’ tab appears slightly differently: in this case each chosen channel (track or bus) is exported to its own file, instead of all channels being mixed together into a single file.

The exported tracks or busses can, by checking Apply track/bus processing, be exported with the effects/processors applied, so that the destination system does not need those effects plugins.

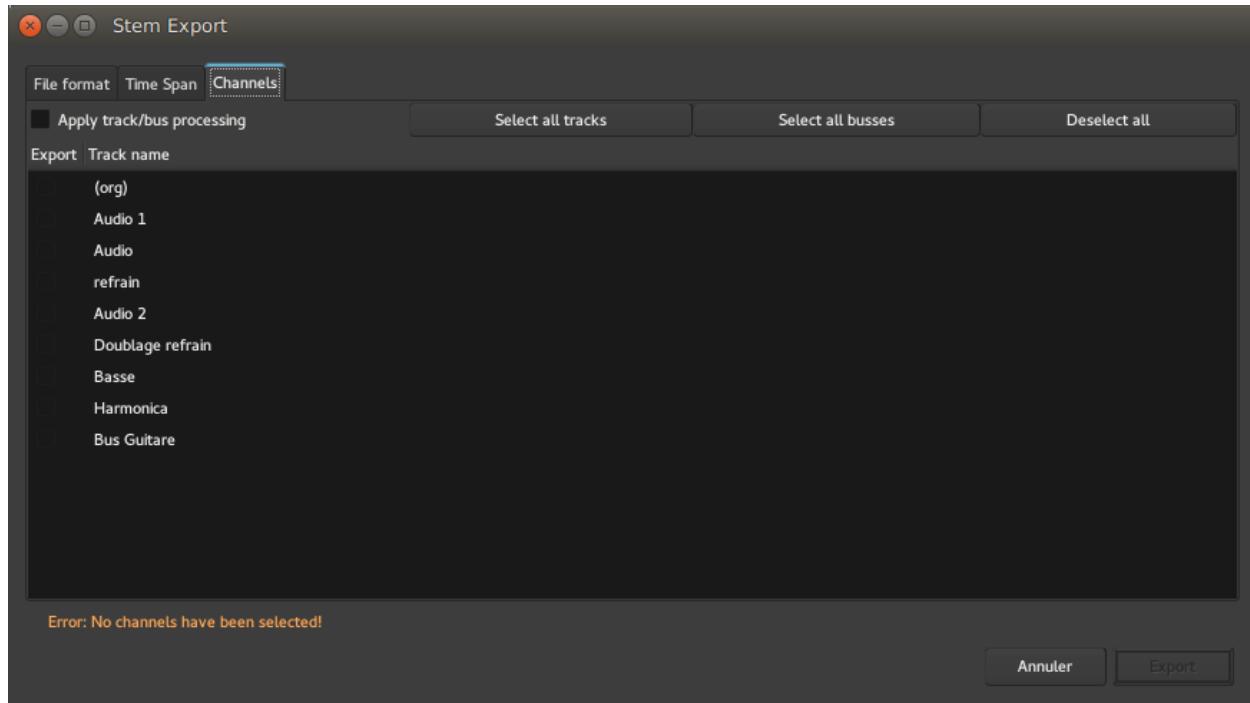


Fig. 5: Stem export

## 71.2 - EXPORT FORMAT PROFILES

### 118.1 Export Format Profiles

An Export Format Profile specifies the file format in which Ardour will export audio files, and also other audio file export options.

Export Format Profiles are edited via the Edit Export Format Profile dialog.

#### 118.1.1 Normalize

If enabled, levels of exported files will be normalized to the level chosen here. The normalization can be either:

- Peak, which adjusts the gain to bring the highest signal peak to the chosen level (in dBFS),
- Loudness, which adjusts the gain to bring the average amplitude to the chosen level (in LUFS), without exceeding the chosen true-peak value (in dBTP). EBU R128 is only available for mono or stereo sounds while true-peak works for any channel layout.

#### 118.1.2 Trim silence at start/end

These checkboxes allow to remove any part Ardour considers silent (0dB), at the beginning or/and end of each exported track.

#### 118.1.3 Add silence at start/end

These checkboxes allow to add silence at the beginning or/and end of each exported track. The duration of the added silence can be manually fixed in the adjacent ‘timer’ input fields.

#### 118.1.4 Compatibility/Quality/File format/Sample rate

##### Compatibility

Selecting an item in the ‘Compatibility’ emphasizes the settings in the other columns that are compatible with the selected standard, by turning incompatible options red. When an incompatible quality/format/sample rate is selected, the compatibility column checkbox disappears.

**New Export Format Profile**

Normalize:  Peak 0,00  dBFS  Loudness -23,00  LUFS ▲ -1,00  dBTP  
 Trim silence at start  Add silence at start: 00:00:00:00  
 Trim silence at end  Add silence at end: 00:00:00:00

Compatibility	Quality	File format	Sample rate
CD	Any	AIFF	Session rate
DVD-A	Lossless (linear PCM)	AU	8 kHz
iPod	Lossy compression	BWF	22.05 kHz
Something else	Lossless compression	IRCAM	44.1 kHz
		WAV	48 kHz
		W64	88.2 kHz
		CAF	96 kHz
		RAW	176 kHz
		Ogg Vorbis	192 kHz
		FLAC	

Sample rate conversion quality: Best (sinc)

**Linear encoding options**

Sample Format	Dithering
8-bit unsigned	Shaped Noise
16-bit	Triangular
24-bit	Rectangular
32-bit	None
float	
double	

Create CUE file for disk-at-once CD/DVD creation  
 Create TOC file for disk-at-once CD/DVD creation  
 Create chapter mark file for MP4 chapter marks

Label: CD (Red Book) (copy) : normalize peak, WAV, 16-bit, 44,1 kHz

Command to run post-export  
(%f=file path, %d=directory, %b=basename, see tooltip for more):

## Quality

The appropriate item in the ‘Quality’ column will be highlighted when a file format is chosen. At the moment, selecting a Quality setting does not show the compatible File formats.

## File format

This column contains a list of Ardour’s supported export file types. Selecting one updates the options underneath it.

## Sample rate

A specific sample rate can be chosen for the exported files, or the current session’s sample rate (by choosing ‘Session rate’), without sample rate conversion.

## Sample rate conversion quality

In case the chosen sample rate does not match the current session’s sample rate, the sample rate conversion quality can be chosen here. Better quality options are slower.

## 118.1.5 Format Options

Options relevant to the chosen file format will appear just under the Compatibility/Quality/File format/Sample rate table.

## Tag with session’s metadata

If the exported file format supports metadata (e.g. FLAC, Ogg Vorbis), use data entered in the *Session Metadata* window to tag the exported files.

## Sample Format and Dithering

The Sample Format is the bit depth of exported files, i.e. the numbers of values a sample can have. Increasing the sample format results in a better defined audio file at the cost of increasing the file size.

If the exported files bit depth is less than Ardour’s native bit depth (32 bits floating point by default), the dithering algorithm, that chooses how to compute the conversion can be chosen in the Dithering column.

## 118.1.6 Options

These options are presented whatever the chosen format is:

## Create CUE/TOC/chapter mark file

As well as exporting an audio file, Ardour can create a file (in CUE, TOC or MP4ch format respectively) containing CD track information, as defined in the *Ranges & Marks List*. Those files can then be used to either burn a CD or DVD, or to create “chapters” inside a compatible mp4 video container.

### 118.1.7 Label

The Label field allows to choose the name which will be shown for this format in the drop-down list of export formats in the ‘File Formats’ tab of the *Export dialog*.

### 118.1.8 Command to run post-export

If this is not blank, it is considered as a command to be run after the export of each file. Either the command must exist in \$PATH, or an absolute path to an executable file can be specified here.

Certain sequences are allowed here to stand for the exported file name and various parameters. Currently these are:

%f	Full path and filename of the exported audio file
%d	Directory containing the exported audio file (including trailing directory separator)
%b	Basename of the exported audio file (without extension)
%s	Path to the current session file
%n	Name of the current session file
%%	A literal percent sign

Any part of the command-line enclosed in double-quotes (“) will be used as-is.

For example, exporting an mp3 file can be done by inserting `lame -b320 %f` which will convert the exported audio file ('%f') to a 320 kbs mp3 using the lame encoder (provided lame is installed first on the system).

---

**CHAPTER  
NINE**

---

**PART X - VIDEO**



## **72 - VIDEO TIMELINE AND MONITORING**

Ardour offers a video timeline and video monitoring for convenient audio mixing and editing to video, in order to produce film soundtracks and music videos, or perform TV post-production tasks.

The video capabilities are:

- Import a single video and optionally extract the soundtrack from it.
- Provide a video monitor window, or full-screen display, of the imported video in sync with any of the available Ardour timecode sources.
- Display a frame-by-frame (thumbnail) timeline of the video.
- Allow for a configurable timecode offset.
- *Lock* audio regions to the video.
- Move audio regions with the video at video-frame granularity.
- Export the video, trim start and end, add blank frames and/or multiplex it with the soundtrack of the current session.

The setup of the video subsystem is modular and can be configured in different ways, including:

- One machine for all video decoding, video monitoring and audio editing tasks
- Two machines, one for video monitoring, one for Ardour
- Three machines, separate video server (for timeline decoding and file archive), dedicated video monitor, and Ardour

Ardour does *not*:

- allow for more than one video to be loaded at a time.
- provide video editing capabilities



## 73 - VIDEO TIMELINE SETUP

No configuration is required if everything is to meant be run on a single machine, and the version of Ardour comes from <http://www.ardour.org>. Everything is pre-configured and included with the download/install.

### 121.1 Single Machine

If Ardour is compiled from source, or installed from a 3rd party repository, three additional tools will need to be installed manually, which are used by Ardour to provide video features:

- xjadeo (the video monitor application): <http://xjadeo.sf.net>
- harvid (a video decoder used for the thumbnail timeline): <http://x42.github.com/harvid/>
- ffmpeg, ffprobe (used to import/export video, extract soundtracks and query video information): <http://ffmpeg.org>

Ardour requires xjadeo version 0.6.4, harvid version 0.7.0 and ffmpeg (known to work versions: 1.2, 2.8.2)

The Ardour development team is in control of the first two applications. ffmpeg however can be a bit of a problem. To avoid conflicts with distribution packages, Ardour looks for `ffmpeg_harvid` and `ffprobe_harvid`.

All four applications need to be found in `$PATH` (e.g. `$HOME/bin` or `/usr/local/bin`). For convenience the binary releases of harvid include `ffmpeg_harvid` and `ffprobe_harvid`, but if the distribution provides suitable ffmpeg commands, symbolic links can be created to the distribution-provided binaries:

```
sudo ln -s /usr/bin/ffmpeg /usr/bin/ffmpeg_harvid sudo ln -s /usr/bin/ffprobe /usr/bin/ffprobe_harvid
```

Binary releases are available from [ardour.org](http://ardour.org) as well as an installer script: [install\\_video\\_tools.sh](#).

The easiest way to install the video-utilities is by running the following line in a terminal:

```
sh -c "$(curl -s -L http://git.io/tVUCkw)"
```

### 121.2 Studio Setup

As Setting up a proper A/V post-production studio can be a complicated task, it is advised to read the info in the previous section to get familiar with the tools involved first. As much as the Ardour team streamlines and simplifies the *single machine* setup, the studio setup is focused on modularity.

- TODO:
- Synchronization ardour → video-display-box should be accomplished by external means. Jack-transport(netjack), MTC, LTC (OSC and/or ssh-pipe work but introduce additional latency + jitter)

- Ardour launches **XJREMOTE** (environment variable, default ‘xjremote’ which comes with xjadeo).
- Either use a custom shell script that ssh’es into the remote box and launches/controls xjadeo there, selects the sync-source and passes though communication between ardour xjadeo via ssh (xjadeo is launched stopped with the session).
- ...or override xjremote’s behavior—instead of IPC with a local running xjadeo-process, using OSC for example. Xjadeo would run permanently and Ardour will just tell it to load files and set offsets via OSC. See [xjremote-osc](#) example script.
- If the video server runs remotely, Ardour needs to be configured in Ardour > Preference > Video (hostname of the video-server).
- Ideally the machines have a common shared folder (NFS or similar). Ardour’s import (audio-extract) and export (mux) functionality depends on having access to the video file. Also Ardour’s video-import transcodes the file into a suitable proxy-format that allows reliable seeking to any frame...

## **74 - TRANSCODING, FORMATS & CODECS**

This chapter provides a short primer on video files, formats and codecs – because it is often cause for confusion:

A video file is a container. It usually contains one video track, one or more audio tracks, and possibly subtitle tracks, chapters... The way these tracks are stored in the file is defined by the file format. Common formats are avi, mov, ogg, mkv, mpeg, mpeg-ts, mp4, flv, or vob.

Each of the tracks by itself is encoded using a Codec. Common video codecs are h264, mpeg2, mpeg4, theora, mjpeg, wmv3. Common audio codecs are mp2, mp3, dts, aac, wav/pcm.

Not all codecs can be packed into a given format. For example the mpeg format is limited to mpeg2, mpeg4 and mp3 codecs (not entirely true). DVDs do have stringent limitations as well. The opposite would be .avi: pretty much every audio/video codec combination can be contained in an avi file-format.

To make things worse, naming conventions for video codecs and formats are often identical (especially MPEG ones) which leads to confusion. All in all it is a very wide and deep field. Suffice there are different uses for different codecs and formats.

### **122.1 Ardour specific issues**

Ardour supports a wide variety of video file formats codecs. More specifically, Ardour itself actually does not support any video at all but delegates handling of video files to [ffmpeg](#), which supports over 350 different video codecs and more than 250 file formats.

When importing a video into Ardour, it will be transcoded (changed from one format and codec to another) to avi/mjpeg for internal use (this allows reliable seeking to frames at low CPU cost—the file size will increase, but hard disks are large and fast).

The export dialog includes presets for common format and codec combinations (such as DVD, web-video,...). If in doubt, one of the presets should be used.

As a last note: every time a video is transcoded, the quality can only get worse. Hence for the final mastering/muxing process, one should always go back and use the original source of the video.



## 75 - WORKFLOW & OPERATIONS

### 123.1 Overview of Operations

Session > Open Video...	Add/replace a video to/on the timeline
Window > Video Monitor	Open/close external video monitor window
View > Video Monitor > ...	Various settings of the video monitor
Session > Export > Export to Video File...	Export session and multiplex with video-file
Left-drag the video in the timeline	Re-align video and move 'locked' audio-regions along
Context-menu on the video-timeline: Lock	Prevent accidental drags
Audio region context menu: <i>Name_Of_The_Region</i> > Position > Lock to video	Mark audio region(s) to be moved along with the video.

### 123.2 Adding a video

Adding a video is a two-step process: selecting a video file, and choosing import mode and optionally selecting an audio track to extract. Only one video can be present in the session, so opening a video when one is already opened results in replacing the video.

#### 123.2.1 Launching the video server (optionnal)

Importing a video makes Ardour start the video server automatically. If the Show video Server Startup Dialog option in the Video section of the *preferences* is checked, the Launch Video Server window is shown, allowing more complex operations, e.g. connecting to a remote video server instead of a local one.

#### 123.2.2 Selecting a file

This step is rather straight-forward. The panel on the right side allows to seek through the video and displays basic file information. It is also useful to check if the video format/codec is supported.

The lower part of the window shows some options:

- Open Video Monitor Window to automatically show the video monitor (Harvid). This can also be done later by using the Window > Video Monitor menu which is binded to V by default.

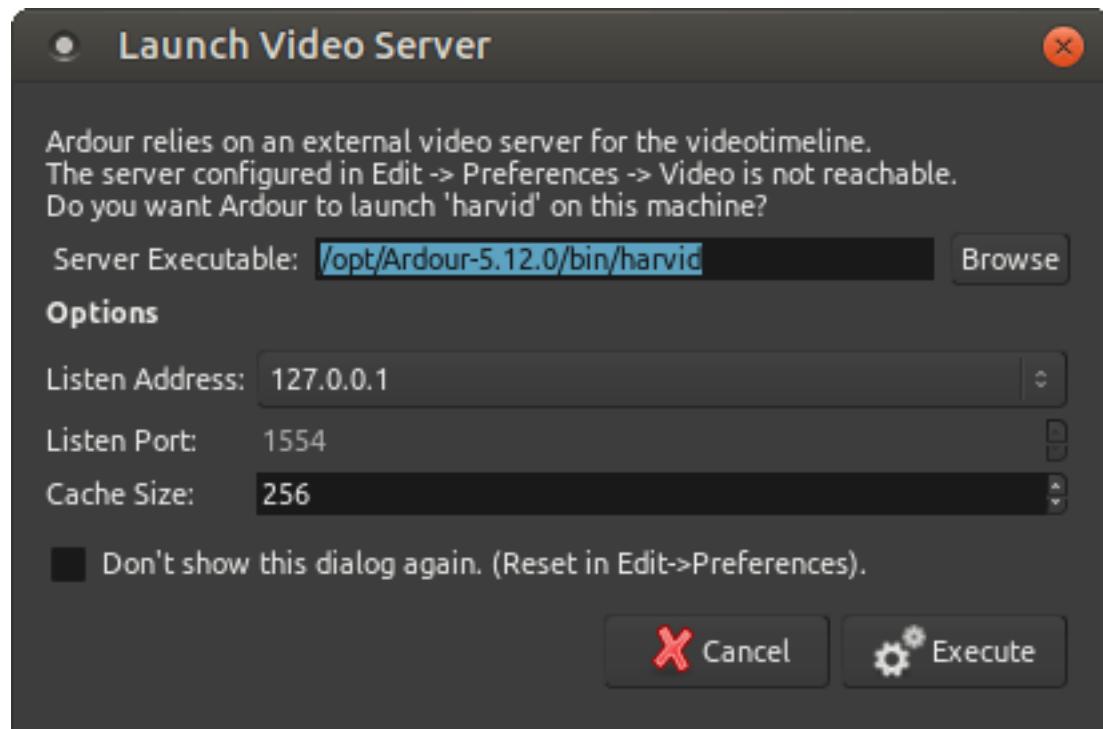


Fig. 1: The Launch Video Server dialog

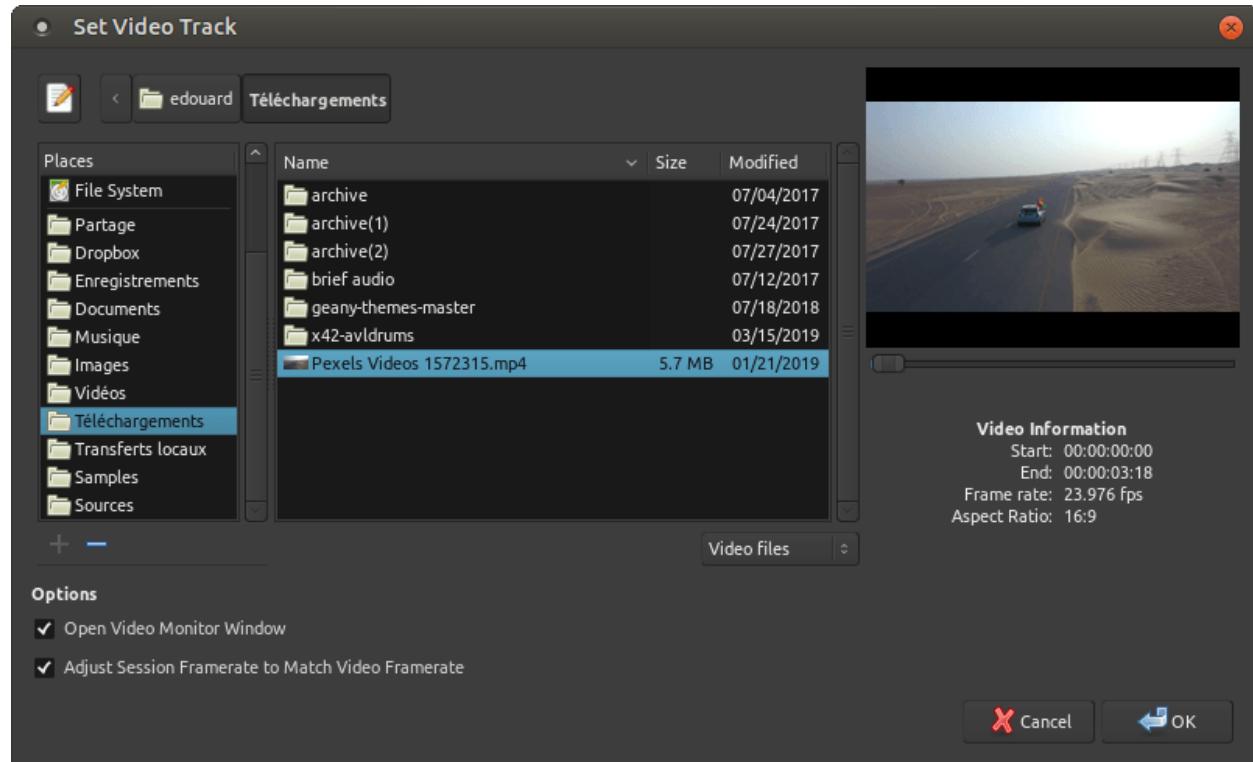


Fig. 2: The video open dialog

- Adjust Session Framerate to Match Video Framerate which can also be set later with the *Session Properties*. Having the session and video framerate at the same value allows their sync not to drift off.

### 123.2.3 Import options

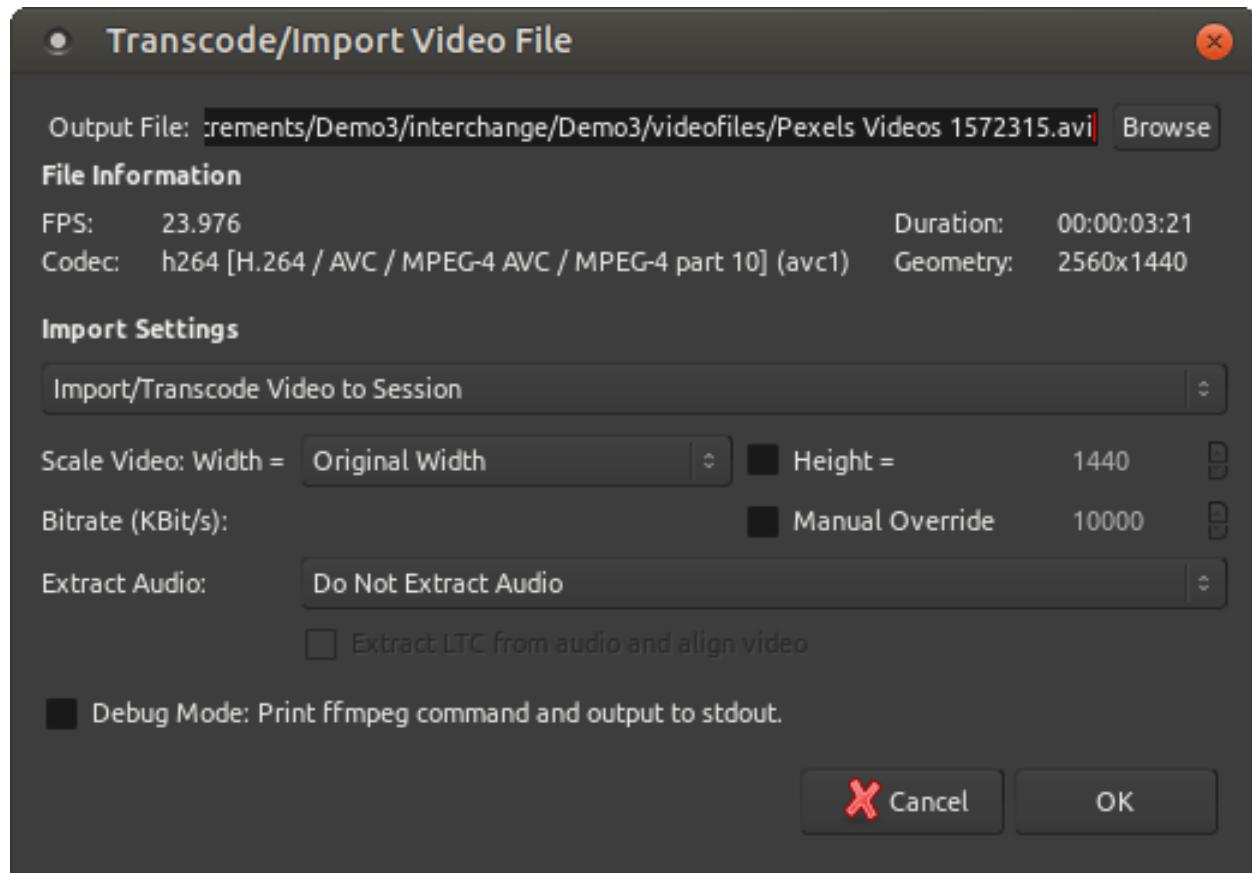


Fig. 3: The Transcode/Import Video dialog

This step analyzes the video file in more detail and offers import options:

Import/Transcode Video to Session	This is the default. The video will be imported in a suitable video format/codec for the timeline and video monitor and saved inside the session folder. A location other than the session folder can also be chosen (external disk, or network storage of the video server on a different machine) by using the Output File: field.
Reference from Current Location (Previously Transcoded Files Only)	Only useful for opening files that were previously encoded (are already in a good format/codec for Ardour). Should be used with care.
Do not Import Video (Audio Import Only)	Useful for extracting audio only.

By default the video is imported using the original width/height. If it is a large video (e.g. full-HD), it makes sense to scale it down to decrease the CPU load and disk I/O required to decode and play the file.

A small, low-quality representation of the image is usually sufficient for editing soundtracks. The default bitrate in kbit/sec is set to use 0.7 bits per pixel (in comparison, the average DVD medium uses 5000 kbit/s; at PAL resolution this is about 0.5 bits per pixel, but the DVD is using the mpeg2—a denser compression algorithm than the mjpeg codec used by Ardour).

The Extract Audio: offers options regarding the Audio part of the stream, allowing to either not extract audio, or to choose which audio stream to add to the session.

When extracting any audio, if it includes *LTC timecodes*, those can be extracted and used to sync the video by checking the option below.

## 123.3 Working with A/V

Working with A/V in Ardour is similar to working in a pure audio setup, except for the presence of a video timeline in the *ruler* zone, and a Xjadeo video window, showing a preview of the result.

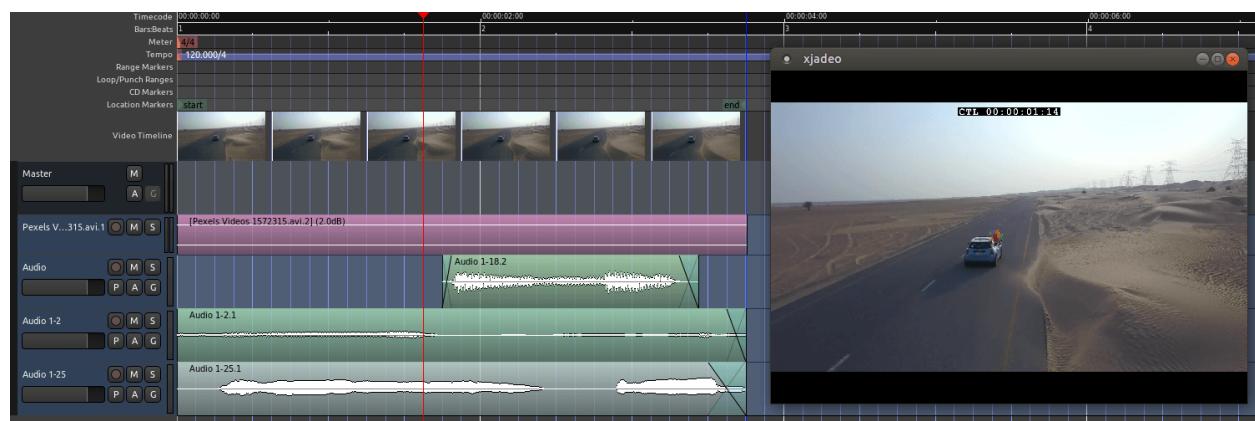


Fig. 4: The Video Timeline

The Xjadeo window supports some user interactions, such as showing some OSD information or changing the zoom level. Xjadeo's documentation is available on its [website](#).

## 123.4 Exporting Video

The video export will take audio from the current Ardour session and multiplex it with a video file. The soundtrack of the video is taken from an audio export of Ardour's master bus.

An arbitrary video file can be chosen. For high quality exports, the original file (before it was imported into the timeline) should be used. This is the default behaviour if that file can be found. If not, Ardour will fall back to the imported proxy-video which is currently in use on the timeline. Any existing audio tracks on this video file are stripped.

The range selection allows to cut or extend the video. If the session is longer than the video duration, black frames are prefixed or appended to the video. (Note: this process may fail with non-standard pixel aspect ratios). If Ardour's session range is shorter, the video will be cut accordingly.

Audio sample-rate and normalization are options for Ardour's audio exporter. The remaining settings are options that are directly passed on to ffmpeg.

The file format is determined by the extension chosen for it (.avi, .mov, .flv, .ogv, .webm,...). Note: not all combinations of format, codec, and settings produce files which are according to specifications. For example,

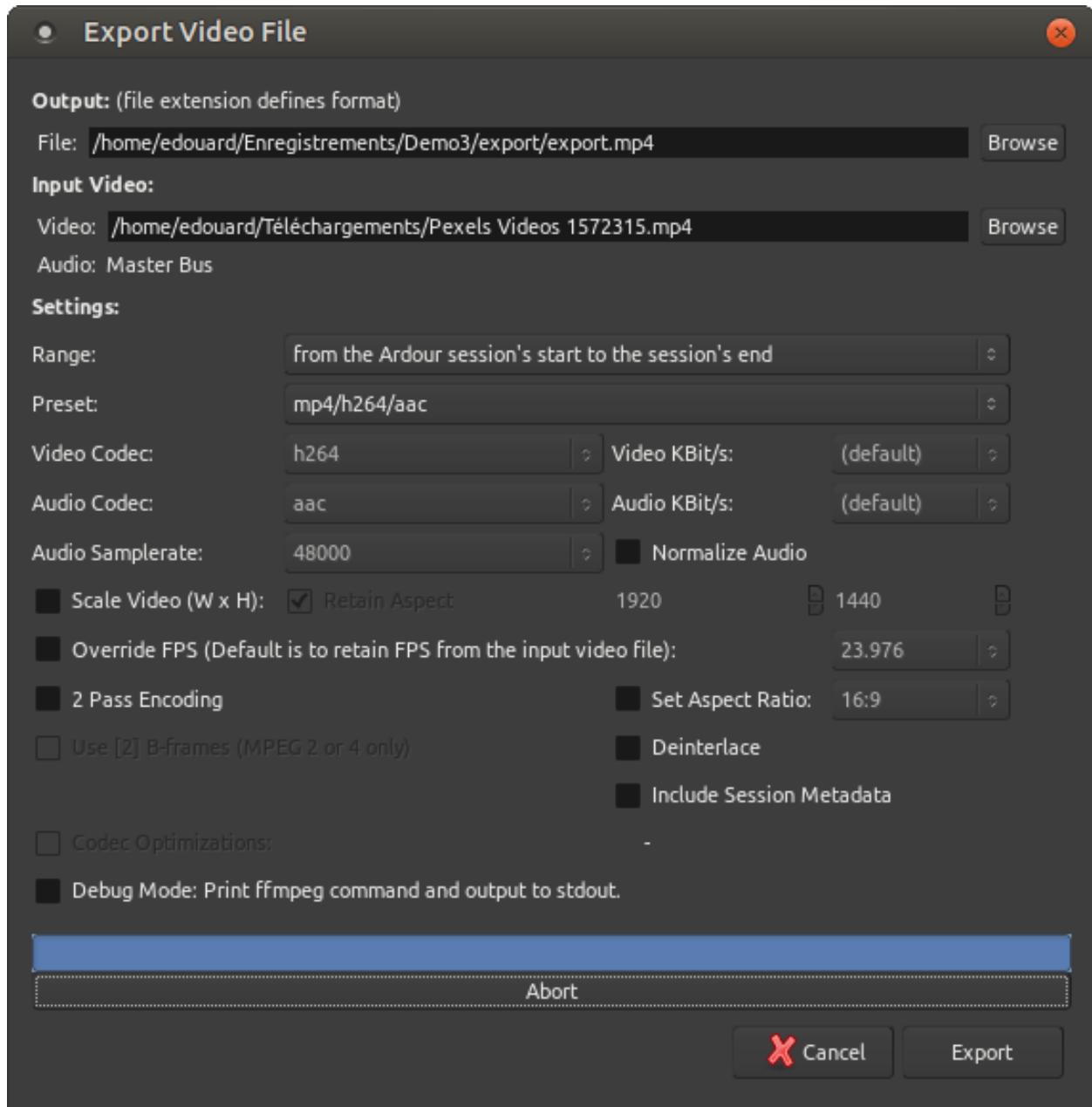


Fig. 5: The Video Export Dialog

flv files require sample rates of 22.1 kHz or 44.1 kHz, mpeg containers can not be used with ac3 audio-codec, etc. If in doubt, one of the built-in presets should be used.

Ardour video export is not recommended for mastering! While ffmpeg (which is used by Ardour) can produce high-quality files, this export lacks the possibility to tweak many settings. We recommend using [winff](#), [devede](#) or [dvdauthor](#) to mux and master. Nevertheless this video-export comes in handy to do quick snapshots, intermediates, dailies or online videos.

---

**CHAPTER  
FOUR**

---

**PART XI - CONTROL SURFACES**



## **76 - CONTROLLING ARDOUR WITH OSC**

OSC lets synthesizers and other devices communicate with Ardour. OSC devices can send commands relating to playback (such as play or stop), performance (such as volume, play, stop, and almost any other function (such as Edit, or Undo).



## 76.1 - OSC: CONTROLLING ARDOUR WITH OSC

OSC lets synthesizers and other devices communicate with Ardour. OSC devices can send commands relating to playback (such as play or stop), performance (such as volume, play, stop, and almost any other function (such as Edit, or Undo).

*Note:* OSC control has changed dramatically since Ardour 4.7. The Path structure has been completely redone, Banking has been introduced, The controller is now able to tell Ardour what kind of feedback it can work with (including bank size) and two new math styles have been added to gain controls. If you are using an Ardour version of 4.7 or less, please read *Osc control in Ardour 4.7 and prior*.

Ardour is probably one of the most OSC-controllable audio applications around, but as with all OSC-controllable apps, you can't do much without knowing what messages can be sent. This document describes the various categories of messages that Ardour understands. It is subject to change, particularly the "Actions" part below, since this relates to the GTK GUI for Ardour rather than the backend.

### 126.1 Connecting to Ardour via OSC

OSC support is not enabled by default, but can be turned on via Edit > Preferences > Control Surfaces. Once enabled, Ardour will listen on port 3819 by default. This port number can be changed by editing \$ARDOUR\_CONFIG and adding this line within the <Config> section:

```
<Option name="osc-port" value="Your choice here"/>
```

Ardour sends any feedback to the port and address that sent any feedback request. The port does not have to match Ardour's port. In fact it is better not to. This means that Ardour can deal with more than one controller at a time. The two controllers can bank independently and even use different math for faders. This could be used to allow talent to adjust their own monitor mix using a tablet or phone that can run an OSC controller. For a full explanation of how Ardour's feedback works please read *OSC feedback In Ardour*.

### 126.2 Control Surface Set Up

Control surface set up allows the controller to tell Ardour about its capabilities. The surface can tell Ardour how many control strips it has for banking, if it is capable of setting its faders or buttons to values set by Ardour's GUI or automation, What kind of math the faders use and more.

Any time the /set\_surface command is sent, the current bank is recalculated and if feedback is turned on, the values of each strip's controls are sent (or refreshed) as well. This will also refresh the Master feedback setup.

Surface Port Setting is available in the OSC GUI.

As of Ardour 5.1, There is now a GUI setup in response to those using tablets with applications such as touchOSC or AndrOSC who need to be able to set a port for Ardour to send to. It can also change the default setting for set\_surface. For more information about Ardour's OSC configuration GUI please read [Ardour's Setup Dialog](#).

If /set\_surface is not sent, the default values are used:

- *Bank Size*: 0— No banking (or infinite bank size).
- *Strip Types*: 0— All strip types except hidden and special.
- *Feedback*: 0— All off.
- *Fader Mode*: 0— gain in dB (not relevant with feedback off)
- *Send Page Size*: 0— No Send Paging.
- *Plugin Page Size*: 0— No Plugin Paging.

These values give the same behaviour as prior versions of Ardour. (or the closest possible)

/set_surface    bank_size    strip_types    feedback    fadermode send_page_size plugin_page_size	See below for an explanation of each parameter.
--	---

The /set\_surface message may have all values except the last in-line. For example: `/set_surface/8/31/8403/0/8 i 16` would be valid. Do be careful of switches which send a 0 on release, it may be necessary to set the value as the release value rather than the press value.

The /set\_surface message may have less than the full set of parameters. those left out will remain as they were before the /set\_surface message was sent. All parameters included must be valid. For example, setting send page size would require also setting bank\_size, strip\_types, feedback and gain mode. using only two parameters will set bank\_size and strip\_types. Sending /set\_surface with no parameters will result in Ardour returning a /set\_surface message with the current settings. Surfaces using /set\_surface iii b st fb gm as was the case in versions of Ardour older than 5.10 will continue to work.

### **126.2.1 bank\_size**

Bank Size is the number of channel strips the controller supports without banking. Setting this to 0 turns banking off by setting the bank size to infinite.

Bank size can also be set with `/set_surface/bank_size size`.

### **126.2.2 strip\_types**

strip\_types is an integer made up of bits. The easy way to deal with this is to think of strip\_types items being worth a number and then adding all those numbers together for a value to send. Strip Types will determine what kind of strips will be included in bank. This would include: Audio, MIDI, busses, VCAs, Master, Monitor and hidden or selected strips.

Aside from setting the track types for the main mix assignments, using /set\_surface/strip\_types with more than one surface button will allow switching between modes for example: inputs only, busses only, selected only, hidden only, by having the buttons send values of: 3, 12, 256, 512. A full mix button might have a value 31.

While Master and Monitor are listed as possibilities, most surfaces will not use them. Using /master and /monitor makes more sense. However, in the case where there are no master or monitor fader strips on the surface, it may be necessary to include them in the banked strips.

Please see: [Calculating Feedback and Strip-types Values](#).

Strip types can also be set with `/set_surface/strip_types types`.

### 126.2.3 feedback

Feedback is an integer made up of bits. The easy way to deal with this is to think of feedback items being worth a number and then adding all those numbers together for a value to send.

Please see: [Calculating Feedback and Strip-types Values](#).

Feedback can also be set with `/set_surface/feedback feedback`.

### 126.2.4 gainmode

Gainmode is a an int that acts as a bool:

- 0 (*or false*) dB value as a float from -193 to +6. Sent as `/strip/gain` SSID value. (-193 or below are the same as -inf)
- 1 (*or true*) A positional fader based on the same math as Ardour's GUI. An Float from 0 to 1. Sent as `/strip/fader` SSID value

Gainmode applies only to feedback values. The controller can choose which gain math to use by choosing to use the `//gain` or `//fader` path to send to Ardour. This makes sure a controller that doesn't set up Ardour's OSC can still use either math. The gainmode for feedback also determines the path Ardour uses for feedback so that the feedback messages match the control messages.

Gain mode can also be set with `/set_surface/gainmode gainmode`.

### 126.2.5 send\_page\_size

Send\_page\_size is an int for the number of send channels that can be controlled at one time. Each channel has a name, level and enable control. (added in Ardour 5.10)

Send page size can also be set with `/set_surface/send_page_size send_page_size`.

### 126.2.6 plugin\_page\_size

plugin\_page\_size is an int for the number of plugin controls that can be controlled at one time. Each control has a name and level. As each plugin is different (as is each parameter), the surface should expect to control the plugin parameters with a variable control (pot or slider) with a float value from 0 to 1 (even on/off switches). (added in Ardour 5.10)

Plugin page size can also be set with `/set_surface/plugin_page_size plugin_page_size`.

## 126.3 Querying Ardour for information

The control Surface may wish to control the type a frequency of updates it receives. It can do this with querying commands. See: [Querying Ardour with OSC](#).

## 126.4 List of OSC messages

Parameter types show how the value will be used. However, they may be sent as a different type if needed, see: [Parameter Types in OSC](#).

### 126.4.1 Master or Global messages

#### Transport Control

/transport_stop	Stops a rolling transport
/transport_play	Puts transport in play mode
/toggle_roll	Toggles between play and stop
/stop_forget	Stop transport and delete/forget last take
/set_transport_speed	Where <i>speed</i> is a float ranging from -8.0f to 8.0f
/ffwd	Adds 1.5 times to transport speed to maximum +8 times normal speed
/rewind	Adds -1.5 times to transport speed to maximum -8 times normal speed
/goto_start	Move playhead to start of session
/goto_end	Move playhead to end of session
/jump_bars bars	Where <i>bars</i> is a float (+/-) of the number of bars to jump
/jump_seconds seconds	Where <i>seconds</i> is a float (+/-) of the number of seconds to jump
/toggle_click	Toggle metronome click on and off
/add_marker	(adds marker to the current transport position)
/remove_marker	Removes marker at the current transport position (if there is one)
/next_marker	Move playhead to next marker
/prev_marker	Move playhead to previous marker
/locate spos roll	where <i>spos</i> is the target position in samples and <i>roll</i> is a bool/integer defining whether you want transport to be kept rolling or not
/loop_toggle	Toggle loop mode on and off
/loop_location start end	<i>start</i> is the beginning of a loop and <i>end</i> is the end of a loop both are integer frame positions.
/midi_panic	Ardour will send an all notes off to all midi tracks
/cancel_all_solos	Cancel All Solos/PFLs/AFLs

New for Ardour 5.9.

`/scrub delta`

Where *delta* is a float indicating forward or reverse movement. See [OSC Scrub Modes](#)

`/jog delta`

Where *delta* is a float indicating forward or reverse movement

`/jog/mode mode`

Where *mode* is an int from 0 to 7 indicating what the `/jog` command controls. See [OSC Jog Modes](#)

## Recording control

/toggle_punch_in	
/toggle_punch_out	
/rec_enable_toggle	Toggles master record enable

## Transport Information

/transport_frame	Ardour sends /transport_frame <i>current_frame</i>
/transport_speed	Ardour sends /transport_speed <i>speed</i>
/record_enabled	Ardour sends /record_enabled <i>recordenable_status</i>

## Editing-related

/undo	
/redo	
/save_state	(this is the regular Session > Save operation)

## Master and Monitor strip control

/master/gain dB	dB is a float indicating the desired gain in dB
/master/fader position	position is a float between 0 and 1 setting the desired position of the fader
/master/db_delta delta	where <i>delta</i> is a float that will increase or decrease the gain of master by the amount of the delta. (Ardour 5.11+)
/master/trimdB dB	dB is a float from -20 to +20 representing the desired trim gain in dB
/master/pan_stereo_position position	position is a float from 0 to 1 representing the desired pan position
/master/mute key	key is an optional float 1 representing a master bus select
/master/select state	state is an int of 0 or 1 representing the desired mute state
/monitor/gain dB	dB is a float indicating the desired gain in dB
/monitor/fader position	position is a float between 0 and 1 setting the desired position of the fader
/monitor/db_delta delta	where <i>delta</i> is a float that will increase or decrease the gain of monitor by the amount of the delta. (Ardour 5.11+)
/monitor/mute state	state is an int of 0 or 1 where 1 is muted
/monitor/dim state	state is an int of 0 or 1 where 1 is dimmed
/monitor/mono state	state is an int of 0 or 1 where 1 is mono mode

### 126.4.2 Track specific operations

For each of the following, *ssid* is the Surface Strip ID for the track

SSID has a different meaning than RID in Ardour version 4.7 and before. Effectively, banking is always being used and the SSID is generated on the fly. The SSID is the position of the strip within bank as an int 1 to bank size. There are no gaps as there have been in the past. Depending on the value of *strip\_types* sent to Ardour, Master and Monitor, may be included in the list of SSIDs or not as set in */set\_surface*.

Some Surfaces (many Android applets) are not able to deal with more than one parameter in a command. However, the two parameter commands below can also be sent as /strip/command/ssid param. In this case the param should be a float even if an int is required below.

/bank_up	Change bank to the next higher bank.
/bank_up <i>delta</i>	Where <i>delta</i> is a float of 1 to bank up and -1 is bank down for use with a surface that only supports one parameter.
/bank_down	Change bank to the next lower bank.
/use_group <i>state</i>	Where <i>state</i> is a float of 1 to use group or 0 to not use group. <a href="#">more info</a>
/strip/mute ssid <i>mute_st</i>	where <i>mute_st</i> is a bool/int representing the desired mute state of the track.
/strip/solo ssid <i>solo_st</i>	where <i>solo_st</i> is a bool/int representing the desired solo state of the track.
/strip/solo_iso ssid <i>state</i>	where <i>state</i> is a bool/int representing the desired solo isolate state of the track.
/strip/solo_safe ssid <i>state</i>	where <i>state</i> is a bool/int representing the desired solo safe/lock state of the track.
/strip/monitor_input ssid <i>monitor_st</i>	where <i>monitor_st</i> is a bool/int where 1 is forced input monitoring.
/strip/monitor_disk ssid <i>monitor_st</i>	where <i>monitor_st</i> is a bool/int where 1 is forced disk monitoring. When 0, Ardour will ignore the disk.
/strip/recenable ssid <i>rec_st</i>	where <i>rec_st</i> is a bool/int representing the desired rec state of the track.
/strip/record_safe ssid <i>rec_st</i>	where <i>rec_st</i> is a bool/int representing the desired record safe state of the track.
/strip/polarity ssid <i>invert</i>	where <i>invert</i> is a bool/int representing the desired polarity of the track.
/strip/gain ssid <i>gain</i>	where <i>gain</i> is a float ranging from -193 to 6 representing the desired gain.
/strip/fader ssid <i>position</i>	where <i>position</i> is a float ranging from 0 to 1 representing the fader control.
/strip/db_delta ssid <i>delta</i>	where <i>delta</i> is a float that will increase or decrease the gain of a track by a fixed amount.
/strip//automation *ssid <i>mode</i>	where <i>mode</i> is an int ranging from 0 to 3 representing the desired automation mode.
/strip//touch *ssid <i>state</i>	where <i>state</i> is an int of 1 for touched and 0 for released. See <a href="#">OSC Automation</a> .
/strip/trimdB ssid <i>trim_db</i>	where <i>trim_db</i> is a float ranging from -20 to 20 representing the desired trim.
/strip/pan_stereo_position ssid <i>position</i>	where <i>position</i> is a float ranging from 0 to 1 representing the desired pan position.
/strip/pan_stereo_width ssid <i>width</i>	where <i>width</i> is a float ranging from 0 to 1 representing the desired pan width.
/strip/send/gain ssid <i>sendid send_gain</i>	where <i>sendid</i> = nth_send, <i>send_gain</i> is a float ranging from -193 to +6.
/strip/send/fader ssid <i>sendid send_gain</i>	where <i>sendid</i> = nth_send, <i>send_gain</i> is a float ranging from 0 to 1 representing the desired fader position.
/strip/send/enable ssid <i>sendid state</i>	where <i>sendid</i> = nth_send, <i>state</i> is 1 for enabled and 0 for disabled.
/strip/list	see: <a href="#">Querying Ardour with OSC</a> .
/strip/sends ssid	see: <a href="#">Querying Ardour with OSC</a> .
/strip/receives ssid	see: <a href="#">Querying Ardour with OSC</a> .
/strip/plugin/list ssid	see: <a href="#">Querying Ardour with OSC</a> .
/strip/plugin/descriptor ssid	see: <a href="#">Querying Ardour with OSC</a> .
/strip/plugin/reset ssid <i>piid</i>	where <i>piid</i> = nth Plugin, will reset all values to the plugin's original values.
/strip/plugin/activate ssid <i>piid</i>	where <i>piid</i> = nth Plugin, will set the plugin's state to active.
/strip/plugin/deactivate ssid <i>piid</i>	where <i>piid</i> = nth Plugin, will set the plugin's state to inactive.
/strip/plugin/parameter ssid <i>piid param value</i>	where <i>piid</i> = nth Plugin, <i>param</i> = nth param, <i>value</i> is a float ranging from -193 to 6.
/strip/name ssid <i>name</i>	where <i>name</i> is a string for the desired name of the track.

### 126.4.3 Selected Strip Operations

New for Ardour 5, A whole set of operations that work on the selected or expanded strip.

Selected strip operations are complex enough for their own page. Please read: [Selection Considerations in OSC](#). This is most important if more than one OSC surface is being used with Ardour.

There are two kinds of selection in OSC. GUI selection and local expansion. By default expansion follows selection.

- GUI selection: Use */strip/select* to set. Selecting a strip in the GUI will set OSC surface select and the surface will set GUI selection as well.
- Local expansion: Use */strip/expand* to expand a strip without changing overall selection. When

/strip/expand is set to 0 or false, the select channel will go back to using the strip selected by the GUI. While expand is turned on, selecting a strip on the GUI does not select the OSC strip. Sending a /strip/select message will override the expand as if it had been set to false. Good for more than one OSC controller at a time.

/strip/select ssid <i>y/n</i>	Where <i>y/n</i> = 1 for select. Sets both GUI select and strip to expanded mode. (0 is ignored)
/strip/expand ssid <i>y/n</i>	Where <i>y/n</i> = 1 for expanded mode. Sets only local strip to Expanded. Setting to 0 resets the expansion to follow selection.
/select/expand <i>y/n</i>	Where <i>y/n</i> = 1 for expanded mode, 0 for Select mode.
/select/recenable <i>y/n</i>	Where <i>y/n</i> is 1 for enabled and 0 for disabled
/select/record_safe <i>y/n</i>	Where <i>y/n</i> is 1 for safe and 0 for unlocked
/select/mute <i>y/n</i>	Where <i>y/n</i> is 1 for enabled and 0 for disabled
/select/solo <i>y/n</i>	Where <i>y/n</i> is 1 for enabled and 0 for disabled
/select/solo_iso <i>state</i>	where <i>state</i> is a bool/int representing the desired solo isolate state of the track
/select/solo_safe <i>state</i>	where <i>state</i> is a bool/int representing the desired solo safe/lock state of the track
/select/monitor_input <i>y/n</i>	Where <i>y/n</i> is 1 for monitor from input and 0 for auto
/select/monitor_disk <i>y/n</i>	Where <i>y/n</i> is 1 for monitor from disk and 0 for auto
/select/polarity <i>invert</i>	where <i>invert</i> is a bool/int representing the desired polarity of the track
/select/gain <i>gain</i>	Where <i>gain</i> is a float ranging from -193 to 6 representing the desired gain of the track in dB.
/select/fader <i>position</i>	Where <i>position</i> is an float ranging from 0 to 1 representing the fader control position.
/select/db_delta <i>delta</i>	where <i>delta</i> is a float that will increase or decrease the gain of the selected track by the amount of the delta. (Ardour 5.11+)
/select//automation * <i>mode</i>	where <i>mode</i> is an int ranging from 0 to 3 representing the desired automation mode for the control. <a href="#">See OSC Automation.</a>
/select//touch * <i>state</i>	where <i>state</i> is an int of 1 for touched and 0 for released. <a href="#">See OSC Automation.</a>
/select/trimdB <i>trim_db</i>	where <i>trim_db</i> is a float ranging from -20 to 20 representing the desired trim of the track in dB.
/select/pan_stereo_position <i>position</i>	where <i>position</i> is a float ranging from 0 to 1 representing the desired pan position of the track
/select/pan_stereo_width <i>width</i>	where <i>width</i> is a float ranging from 0 to 1 representing the desired pan width of the track
/select/pan_elevation_position <i>position</i>	where <i>position</i> is a float ranging from 0 to 1 representing the desired pan elevation of the track
/select/pan_frontback_position <i>position</i>	where <i>position</i> is a float ranging from 0 to 1 representing the desired front to back position of the track
/select/pan_lfe_control <i>value</i>	where <i>value</i> is a float ranging from 0 to 1 representing the desired LFE control value for the track
/select/send_gain”, <i>sendid</i> <i>send_gain</i>	where <i>sendid</i> = nth_send, <i>send_gain</i> is a float ranging from -193 to +6 representing the desired gain in dB for the send
/select/send_fader”, <i>sendid</i> <i>send_gain</i>	where <i>sendid</i> = nth_send, <i>send_gain</i> is a float ranging from 0 to 1 representing the desired position for the send as a fader
/select/send_enable”, <i>sendid</i> <i>state</i>	where <i>sendid</i> = nth_send, <i>state</i> is 1 for enabled and 0 for disabled
/select/send_page”, <i>delta</i>	where <i>delta</i> is an int or float selecting another send as a delta from the current send.
/select/plugin_page”, <i>delta</i>	where <i>delta</i> is an int or float selecting another plugin parameter as a delta from the current parameter.
/select/plugin/parameter”, <i>plugin parameter value</i>	where <i>plugin</i> = nth plugin, <i>parameter</i> = nth parameter and <i>value</i> is a float from 0 to 1

`/select/send_page` and `/select/plugin_page` may be used with a page up and page down switch by using a switch with a value of 1 for page up and a switch with a value of -1 for page down. An encoder can be used as well. (these commands were added in Ardour version 5.10)

#### 126.4.4 Menu actions

Every single menu item in Ardour's GUI is accessible via OSC. There is a single common syntax to trigger the action as if it was selected with the mouse (or keyboard):

`/access_action action_name`

As of Ardour 5.9, `access_action` can be inlined for control surfaces that are unable to send string parameters. The `action_name` is composed of a group and an action in the form of *Group/action* which fits very well as an OSC path extension:

`/access_action/Group/action key_pressed`

The `key_pressed` is optional, but if present is a float 1 or 0 where the command is ignored if `key_pressed` is 0.

Some of the Menu Actions duplicate other OSC commands. In all cases it is better to use the OSC commands rather than the Menu Actions if possible as the OSC commands are more direct.

The *list of actions* shows all available values of `action-name` for Ardour.



## 76.2 - OSC: USING THE SETUP DIALOG

Starting with Ardour 5.1 OSC has a graphic setup dialog. This dialog can be accessed from Preferences->Control Surfaces. Select OSC and click on the Show Protocol Settings.

The Ardour OSC dialog has three tabs. The main tab, the Strip Types tab and the Feedback tab.

Many OSC devices get their IP from a DHCP making it difficult to set an IP in Ardour's OSC settings. Therefore, most of the settings are *default* settings. Values are set and the next OSC surface to send an OSC message to Ardour will use those settings. An OSC surface that has previously sent a message to Ardour will retain the settings it already had. Any change to a setting will reset all device settings. A */refresh* message will set that device to any new settings. The use of */set\_surface* will override all settings except *Port Mode*. *Port Mode* affects all connected surfaces and so all surfaces must use either the set manual port or send OSC messages from the same port they expect to receive feedback on.

### 127.1 Dialog settings

#### 127.1.1 OSC setup tab

##### Connection:

This field is informational only. It shows where Ardour will receive OSC messages. The system Name and the Port are the most important parts. Normally, Ardour will use 3819 as its server port. However, if some other server is already using this port, Ardour will try to use the next port up and will keep trying up to 10 ports up.

##### Port Mode:

This drop down allows the choice of Auto or Manual outbound port setting. The Auto port mode, will send OSC messages back to the port messages from that surface are received from. This setting allows two surfaces on the same IP to operate independently. However, there are a number of OSC control surfaces that do not monitor the same port they send from and in fact may change ports they send from as well. Manual allows the outgoing port, the port the surface will receive on, to be manually set. In Manual port mode only one control surface per IP can work. Most phone or tablet OSC controllers like touchOSC or Control need Manual port mode. More than one controller can be used so long as each has its own IP.

##### Manual Port:

This is an Entry box for setting the outgoing port when in Manual port mode.

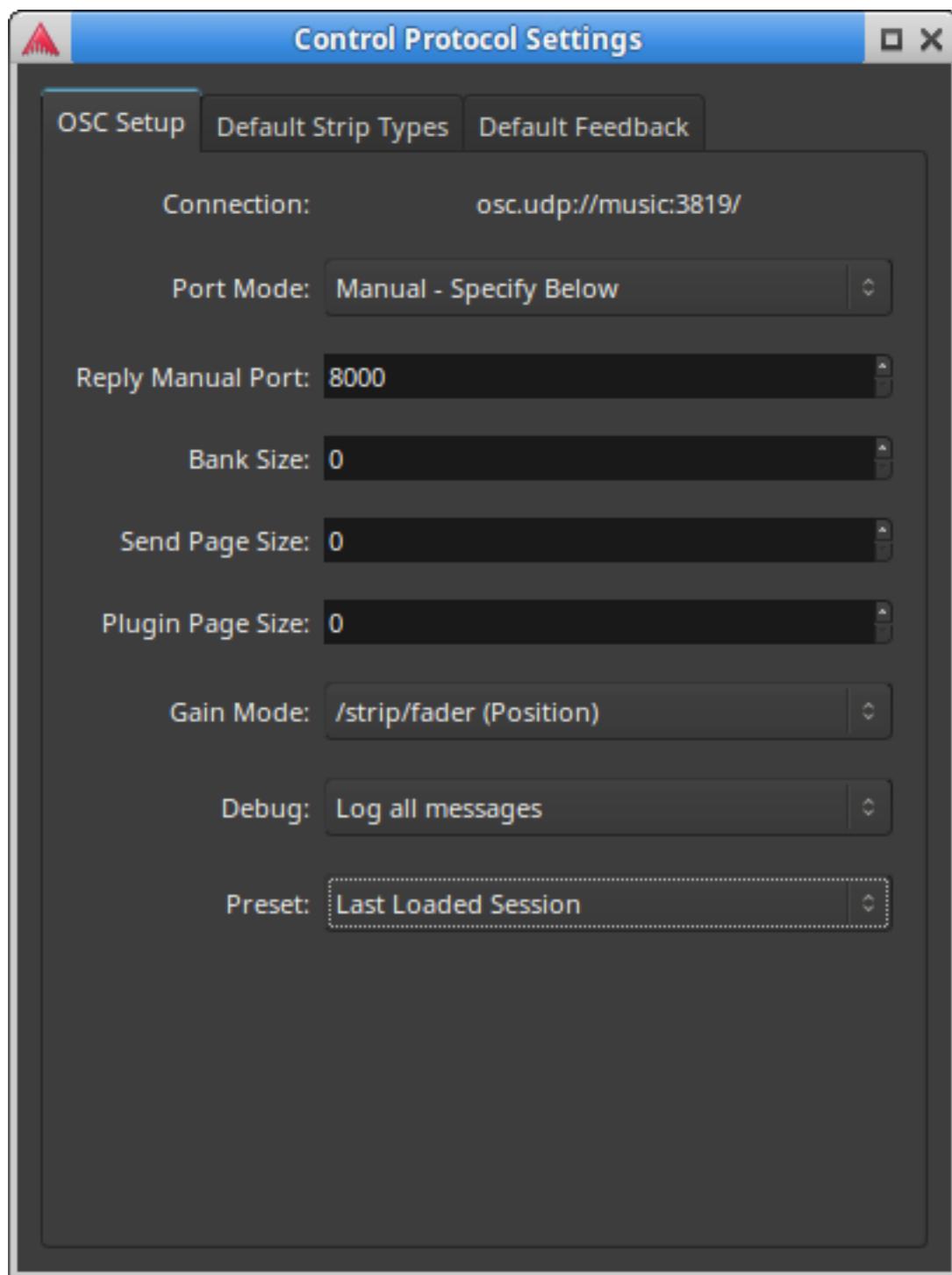


Fig. 1: The OSC configuration dialog

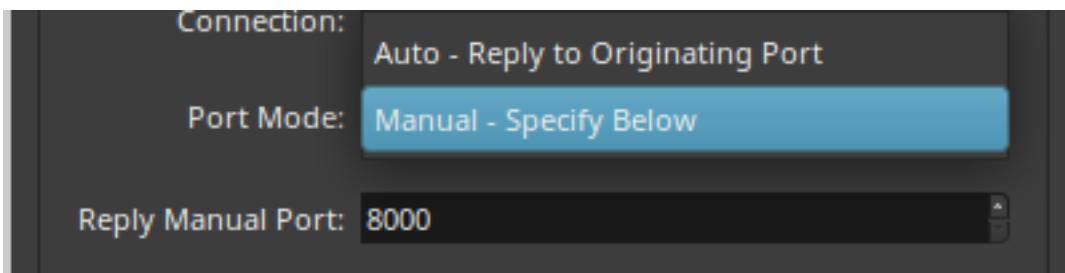


Fig. 2: Port Mode Dropdown

**Bank Size:**

This sets the default bank size for the next surface to send a `/set_surface/*` OSC message. Bank size 0 (the default) sets no banking and allows controlling all strips included in strip\_types at once. The entry area will be bright blue for a port that is not valid (ports below 1024 or 3819).

**Send Page Size:**

This allows setting the size of pages for sends. In the case there are more sends than controls. A size of 0 is the same as no paging and all sends are directly controllable.

**Plugin Page Size:**

This allows setting the size of pages for plugin controls. Some plugins have hundreds of controls and so it may be necessary to page the plugin controls to a limited number of physical controls. A size of 0 is the same as no paging and all plugin controls are directly controllable.

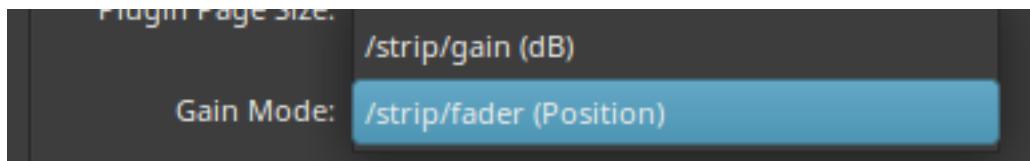
**Gain Mode:**

Fig. 3: Gain Mode Dropdown

Sets the faders (and sends faders) feedback math to position where a value between 0 and 1 represents the fader position of the same fader in the mixer GUI or dB where the feedback from fader movement will be returned as a dB value. When the Gain Mode is set to position, the `/*name` feedback for the channel will show dB values in text while the fader is being adjusted and then return the name text.

**Debug:**

For debugging purposes this allows logging either all OSC messages Ardour receives or invalid messages received or none.

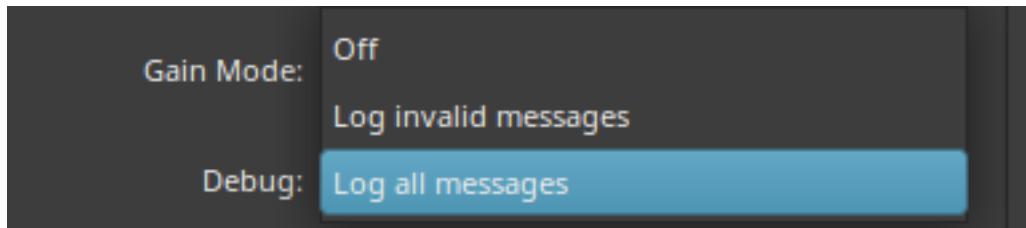


Fig. 4: Debug Mode Dropdown

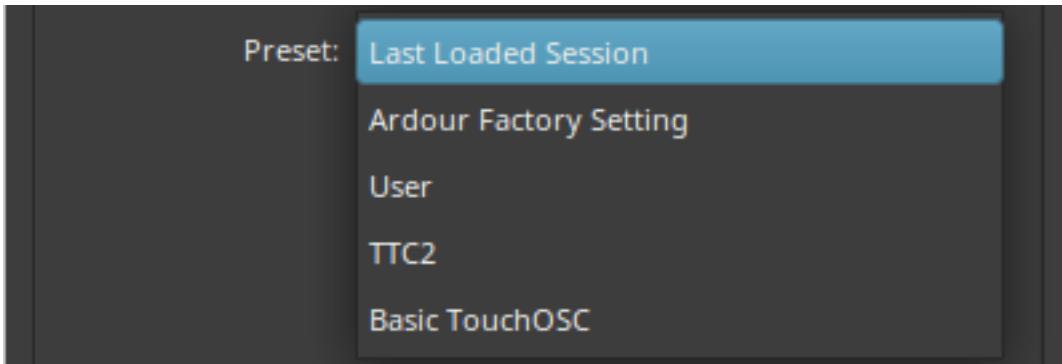


Fig. 5: Preset Loader

**Preset:**

Ardour now allows the use of preset settings. The default settings used are the settings from the last session or the factory defaults the first time OSC is enabled. As soon as any of these settings are changed, the Preset will change to “User” and the new settings will be save to the osc directory in the *Ardour configuration directory* as *user.preset*. This preset file can be renamed for future use. It is suggested to also change the name value inside to avoid confusion in the preset listing. Ardour will ship with some of it’s own presets that go with some popular OSC control and map combinations.

**Clear OSC Devices**

This button has been removed after Ardour 5.10. Instead this action is triggered by any change in the settings.

This button clears operating device profiles so that Ardour will reset all devices settings to use the new defaults from changed settings. a device may still override these new settings with the /set\_surface set of commands. The reason for setting defaults settings is that some OSC controllers are not able to send more than one parameter at a time and so having correct defaults allows one “Connect” button rather than 4.

### 127.1.2 Default Strip Types tab

This allows selecting which of Ardour’s mixer strips will be available for control. The Factory default is all strips except master, monitor and hidden strips. If it is desired to only see input tracks the others can be deselected. It is also possible to change these settings from the control surface. A set of buttons could select showing only inputs or only busses. If a group is selected in the GUI then showing only selected strips will show only that group. Showing hidden tracks is handy for cases where a groups of tracks that grouped to a bus or controlled by a VCA are hidden, but one of those tracks needs a tweak.

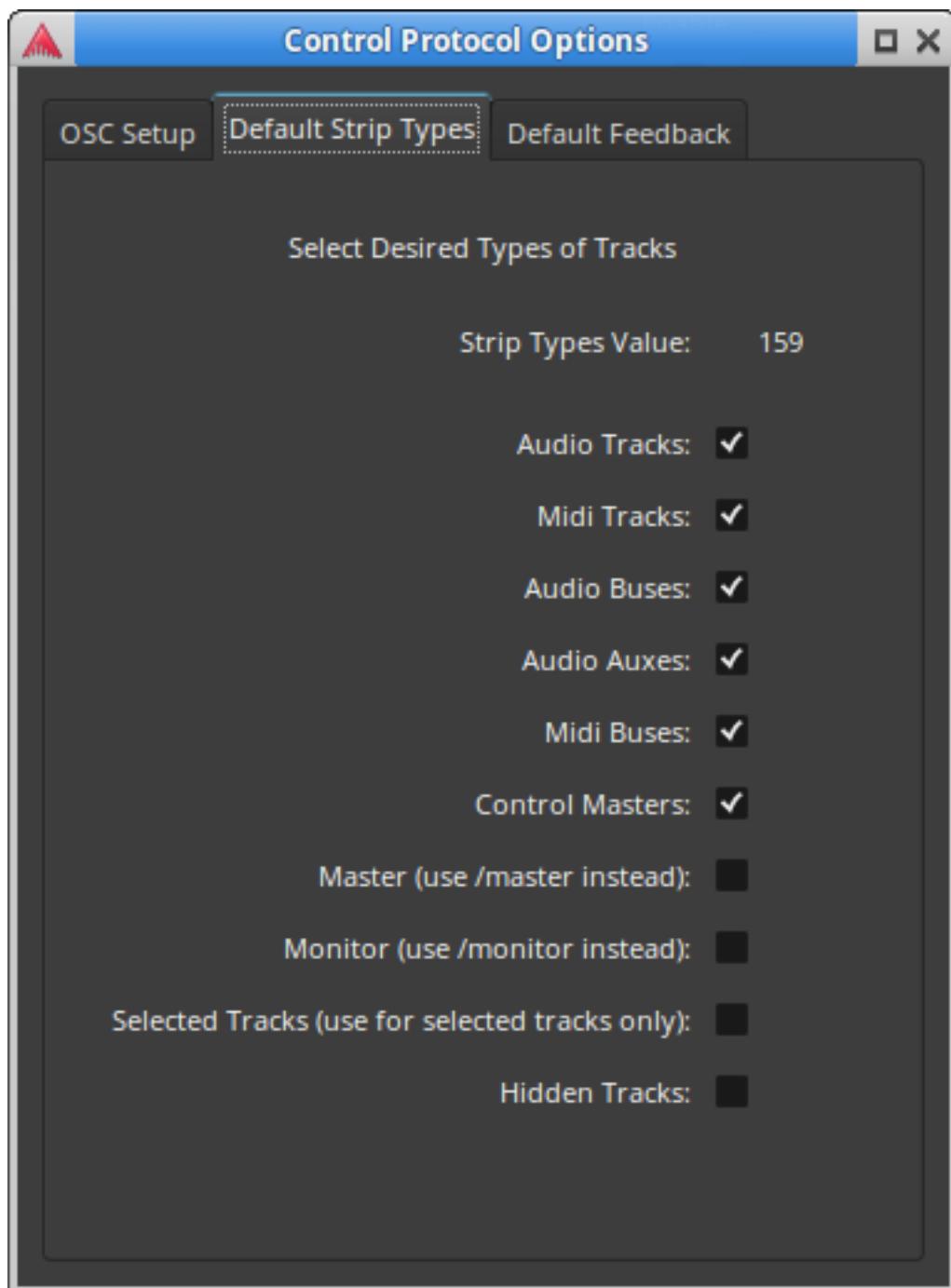


Fig. 6: The Default Strip Types tab

### 127.1.3 Default Feedback tab

This allows setting up which controls provide feedback. The Factory default is none. If the controller is unable to receive feedback, this should be left blank. In the case of metering, Metering as a LED strip only works if Metering as a Float is disabled.

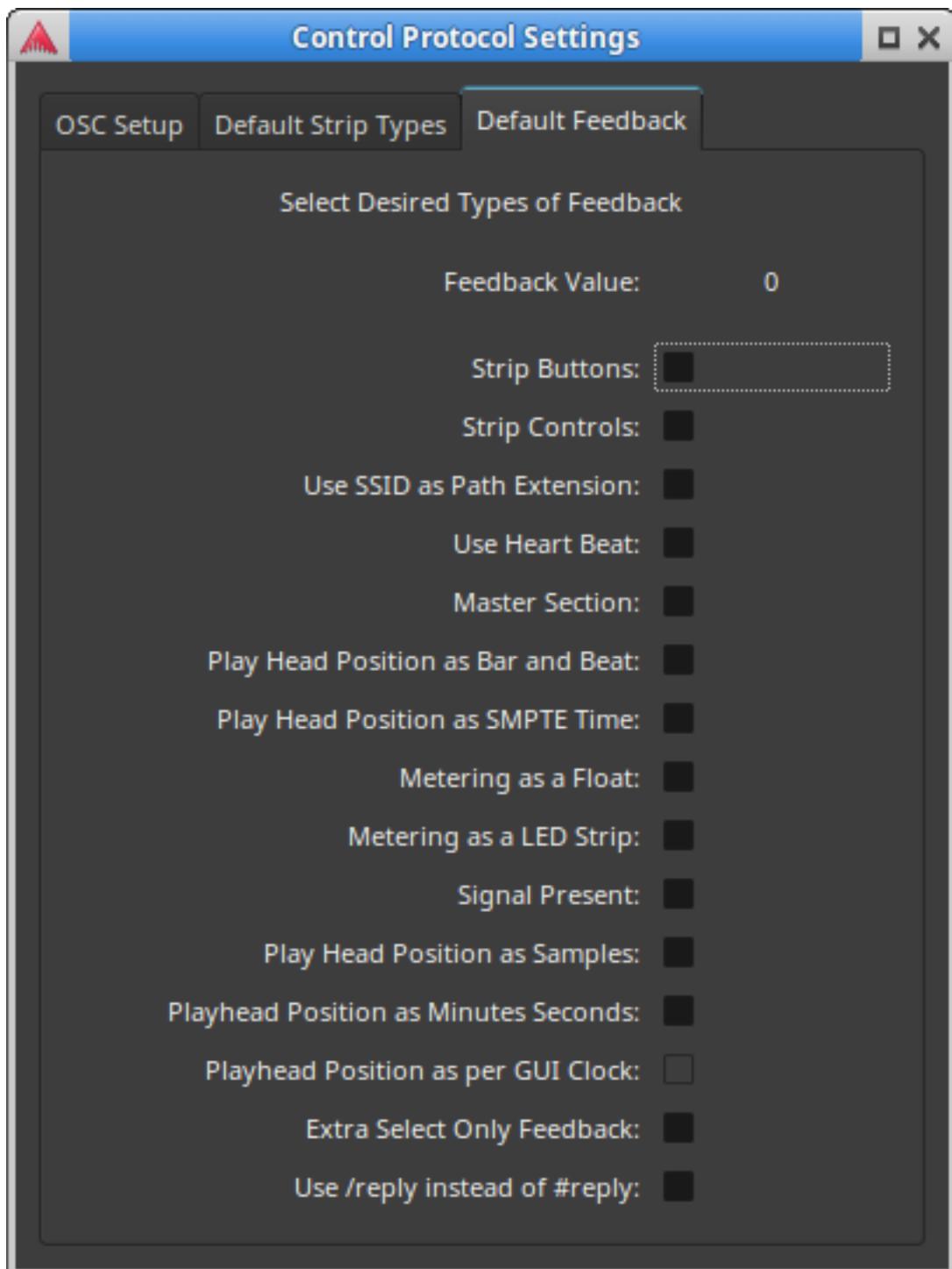


Fig. 7: The Default Feedback tab



## 76.3 - OSC: QUERYING ARDOUR

In order to make a custom controller that knows what strips Ardour has, the controller needs to be able to query Ardour for that information. These set of commands are for smarter control surfaces That have the logic to figure out what to do with the information. These are not of value for mapped controllers like touchOSC and friends. The controller will need to send these queries to Ardour as often as it needs this information. It may well make sense to use regular feedback for things that need to be updated often such as position or metering. Here are the commands used to query Ardour: (added in Ardour 5.5)

/strip/list	Ask for a list of strips
/strip/sends <i>ssid</i>	Asks for a list of sends on the strip <i>ssid</i>
/strip/receives <i>ssid</i>	Asks for a list of tracks that have sends to the strip <i>ssid</i> points to
/strip/plugin/list <i>ssid</i>	Asks for a list of plug-ins for strip <i>ssid</i> .
/strip/plugindescriptor <i>ssid piid</i>	Asks for a list of descriptors for plug-in <i>piid</i> on strip <i>ssid</i>
/set_surface	Ask for the current surface setting. Reply is in the same form as setting the surface would be.

### 128.1 A list of strips

`/strip/list` asks Ardour for a list of strips that the current session has. Ardour replies with a message for each strip with the following information:

Strip type - One of:

- AT - Audio Track
- MT - MIDI Track
- B - Audio Bus
- MB - MIDI bus
- AX - Aux bus
- V - VCA

Strip name

Number of inputs

Number of outputs

Muted

Soloed

Ssid (strip number)

Record enabled

After all the strip messages have been sent, one final message is sent with:

- The text `end_route_list`
- The session frame rate
- The last frame number of the session
- Monitor section present

The `/set_surface` should be set before this is called. That way The right set of strips will be sent in return (though the default is good for most uses) and feedback will start correctly.

If the surface is using `/strip/list`, the surface needs to know if the strips have changed. This would be true if a strip gets moved, created or deleted. When this happens Ardour sends `/strip/list` to the surfaces that have previously requested a `/strip/list`. This lets the surface know that its list of strips is no longer valid.

A bus will not have a record enable and so a bus message will have one less parameter than a track. It is the controllers responsibility to deal with this.

## 128.2 A list of sends

`/strip/sends ssid` asks Ardour for a list of sends for strip number ssid. The reply is sent back to the controller as one message with the following information:

Ssid that information is for

Each send's information:

- The send's target bus ssid
- The send's target bus name
- The send id for this strip
- The send gain as a fader position
- The Send's enable state

The controller can tell how many sends there are from the number of parameters as each send has 5 parameters and there is one extra for ssid.

## 128.3 A list of tracks that send audio to a bus

`/strip/receives ssid` will return a list of tracks that have sends to the bus at the ssid. The reply will contain the following information for each track connected to this bus:

- The ssid of the track sending
- The name of the sending track
- The id of the send at that track
- Its gain in fader position
- The send's enable state

## 128.4 A list of plug-ins for strip

`/strip/plugin/list ssid` will return a list of plug-ins that strip ssid has. The reply will contain the following information:

Ssid that information is for

Each plugin's information:

- The plug-in's id
- The plug-in's name

## 128.5 A list of a plug-in's parameters

`/strip/plugin/descriptor ssid pid` will return the plug-in parameters for pid plug-in on the ssid strip. The reply will be sent as a number of messages, one for each parameter. Each message will contain the following information:

Ssid of the strip the plug-in is in

The plug-in id for the plug-in

The plug-in parameter id for the plug-in

The plug-in parameter's name

Information about that parameter:

- A bitset of flags (see below)
- Data type
- Minimum value
- Maximum value
- The number of scale points
- zero or more scale points of one value and one string each
- The current parameter value

After all the parameters have been sent this way, one final message " `/strip/plugin/descriptor_end` is sent with these parameters:

- Ssid of the strip the plugin is in
- The plug-in id for the plug-in

The flag bitset above has been defined as (from lsb):

- 0—enumeration
- 1—integer step
- 2—logarithmic
- 5—sample rate dependent
- 6—toggled
- 7—controllable
- 8—hidden

Bits 3 and 4 are not used, they were max unbound and min unbound in previous versions and always zero. While this seems complex, it is really not that bad. Minimum, maximum and value will in most cases give you all you need. For simpler access to plug-ins, the /select/plugin/ set of commands will handle most needs.

## 76.4 - OSC: FEEDBACK

Feedback from the Ardour to the control surface is very useful for a number of things. Motor faders need to know where the track they have been attached to is at before they were assigned otherwise the DAW fader will jump to where the controller fader is. Likewise, the buttons on each strip need to know what their value is so they can light their LED correctly. Transport controls should let you know if they are active too. This is what feedback is all about.

Ardour does feedback by sending the same path back that is used to control the same function. As such any controls that have feedback have a parameter that is the value of the control or its state (on or off). In the case of OSC paths listed on the main OSC page as having no parameter, if they have feedback, they will also work with a 1 for button press and 0 for button release. This is because many OSC controllers will only use exactly the same path for feedback as for control. For example:

```
/transport_stop [ ]
```

can be used also in the form:

```
/transport_stop press | where press is an int/bool indicating if the button is pressed or not.
```

The feedback does not have the same meaning as the control message. Where the button release sent to Ardour will be ignored and has no meaning. Both states have meaning in feedback to the controller. The feedback will be:

```
/transport_stop state | where state is an int/bool indicating if the transport is stopped or not.
```

With feedback turned on, OSC control commands that try to change a control that does not exist will get feedback that resets that control to off. For example, sending a /strip/recenable to a buss will not work and Ardour will try to turn the controller LED off in that case. Also note that Pan operation may be limited by pan width in some cases. That is with pan width at 100% (or -100%) there is no pan position movement available.

It may come as a surprise, but feedback often generates more network traffic than control itself does. Some things are more obvious like head position or meters. But even a simple button push like transport start sends not only a signal to turn on the play LED, but also one to turn off the stop LED, the Rewind LED, the Fast Forward LED and the Loop LED. That is still minor, think instead of a surface refresh such as happens when the surface is first connected and then most of that happens every time the fader strips are banked. This is why feedback is enabled in sections so that as little feedback as is actually needed is sent. This is also a consideration if the surface is connected via wifi.

## 129.1 List of OSC feedback messages

### 129.1.1 Feedback only

These messages are feedback only. They are sent as status from Ardour and some of them may be enabled separately from other feedback. See: [Calculating Feedback and Strip-types Values](#).

See strip section below for info about ssid and wrapping it into the path. Also /master and /monitor support what the /strip does.

In the case where Gainmode is set to position, the track name will show the dB value while values are changing.

/strip/name <i>ssid</i> <i>track_name</i>	where <i>track_name</i> is a string representing the name of the track
/strip//automation_name *ssid name	where <i>name</i> is a string representing the current automation mode for the control. <a href="#">See OSC Automation.</a>
/session_name <i>ses-</i> <i>sion_name</i>	where <i>session_name</i> is a string representing the name of the session
/strip/meter <i>ssid meter</i>	where <i>meter</i> is a value representing the current audio level. (the exact math used is determined by the feedback bits set)
/strip/signal <i>ssid signal</i>	where <i>signal</i> is a float indicating the instantaneous audio level is -40dB or higher.
/position/smpte time	where <i>time</i> is a string with the current play head time. Seconds as per smpete.
/position/bbt beat	where <i>beat</i> is a string with the current play head bar/beat.
/position/time time	where <i>time</i> is a string with the current play head time. Seconds are in milliseconds
/position/samples samples	where <i>samples</i> is a string with the current play head position in samples.
/heartbeat LED	where <i>LED</i> is a float that cycles 1/0 at 1 second intervals.
/record_tally state	Some record enable is true or “ready to record”. For a “Recording” sign at studio door.

### 129.1.2 Transport Control

/transport_stop state	<i>state</i> is true when transport is stopped
/transport_play state	<i>state</i> is true when transport speed is 1.0
/ffwd state	<i>state</i> is true when transport is moving forward but not at speed 1.0
/rewind state	<i>state</i> is true when transport speed is less than 0.0
/loop_toggle state	<i>state</i> is true when loop mode is true
/cancel_all_solos state	Where <i>state</i> true indicates there are active solos that can be canceled.
/jog mode/name name	Where <i>name</i> is a string indicating the name of the current jog mode.

### 129.1.3 Recording control

/rec\_enable\_toggle state

Master record enabled.

### 129.1.4 Master and monitor strips

Master and monitor strips are similar to track strips but do not use the SSID. Rather they use their name as part of the path:

/master/gain <i>dB</i>	where <i>dB</i> is a float ranging from -193 to +6 representing the actual gain of master in dB
/master/fader <i>position</i>	where <i>position</i> is an int ranging from 0 to 1023 representing the fader control position
/master/trimdB <i>dB</i>	where <i>dB</i> is a float ranging from -20 to +20 representing the actual trim for master in dB
/master/pan_stereo_position <i>position</i>	where <i>position</i> is a float ranging from 0 to 1 representing the actual pan position for master
/master/mute <i>state</i>	where <i>state</i> is a bool/int representing the actual mute state of the Master strip
/monitor/gain <i>dB</i>	where <i>dB</i> is a float ranging from -193 to 6 representing the actual gain of monitor in dB
/monitor/fader <i>position</i>	where <i>position</i> is an int ranging from 0 to 1023 representing the fader control position
/monitor/mute <i>state</i>	where <i>state</i> is a bool/int representing the actual mute state of the Monitor strip
/monitor/dim <i>state</i>	where <i>state</i> is a bool/int representing the actual dim state of the Monitor strip
/monitor/mono <i>state</i>	where <i>state</i> is a bool/int representing the actual mono state of the Monitor strip

### 129.1.5 Track specific operations

For each of the following, *ssid* is the surface strip ID for the track

Some Surfaces (many Android applets) are not able to deal with more than one parameter in a command. However, the two parameter commands below can also be sent as /strip/command/*ssid* param. Feedback can be set to match this with the /set\_surface/feedback *state* command. See [Calculating Feedback and Strip-types Values](#).

/bank_up LED	where <i>LED</i> is a bool that indicates another bank_up operation is possible.
/bank_down LED	where <i>LED</i> is a bool that indicates another bank_down operation is possible.
/strip/name ssid track_name	where <i>track_name</i> is a string representing the name of the track (note there is no corresponding command to set the track name)
/strip/mute ssid mute_st	where <i>mute_st</i> is a bool/int representing the actual mute state of the track
/strip/solo ssid solo_st	where <i>solo_st</i> is a bool/int representing the actual solo state of the track
/strip/monitor_input ssid monitor_st	where <i>monitor_st</i> is a bool/int. True/1 meaning the track is force to monitor input
/strip/monitor_disk ssid monitor_st	where <i>monitor_st</i> is a bool/int. True/1 meaning the track is force to monitor disk, where both disk and input are false/0, auto monitoring is used.
/strip/recenable ssid rec_st	where <i>rec_st</i> is a bool/int representing the actual rec state of the track
/strip/record_safe ssid rec_st	where <i>rec_st</i> is a bool/int representing the actual record safe state of the track
/strip/gain ssid gain	where <i>gain</i> is a float ranging from -193 to 6 representing the actual gain of the track in dB.
/strip/fader ssid position	where <i>position</i> is an float ranging from 0 to 1 representing the actual fader position of the track.
/strip//automation *ssid mode	where <i>mode</i> is an int ranging from 0 to 3 representing the actual automation mode for the control. <i>See OSC Automation.</i>
/strip/trimdB ssid trim_db	where <i>trim_db</i> is a float ranging from -20 to 20 representing the actual trim of the track in dB.
/strip/pan_stereo_position ssid position	where <i>position</i> is a float ranging from 0 to 1 representing the actual pan position of the track

### 129.1.6 Selected Operations

Selection feedback is the same as for strips, only the path changes from */strip* to */select* and there is no *ssid*. There are some extra feedback and commands that will be listed here.

/select/n_inputs number	where <i>number</i> number of inputs for this strip
/select/n_outputs number	where <i>number</i> number of outputs for this strip
/select/comment text	where <i>text</i> is the strip comment
/select/solo_iso state	where <i>state</i> is a bool/int representing the Actual solo isolate state of the track
/select/solo_safe state	where <i>state</i> is a bool/int representing the actual solo safe/lock state of the track
/select/polarity invert	where <i>invert</i> is a bool/int representing the actual polarity of the track
/select/pan_stereo_width width	where <i>width</i> is a float ranging from 0 to 1 representing the actual pan width of the track
/select/send_gain", sendid send_gain	where <i>sendid</i> = nth_send, <i>send_gain</i> is a float ranging from -193 to +6 representing the actual gain in dB for the send
/select/send_fader", sendid send_gain	where <i>sendid</i> = nth_send, <i>send_gain</i> is a float ranging from 0 to 1 representing the actual position for the send as a fader
/select/send_name sendid send_name	where <i>send_name</i> is a string representing the name of the buss this send goes to.

### 129.1.7 Menu actions

Every single menu item in Ardour's GUI is accessible via OSC. However, there is no provision for returning the state of anything set this way. This is not a bad thing as most menu items either do not have an on/off state or that state is quite visible. Bindings that affect other parameters that OSC does track will show on those OSC controls. Examples of this might be track record enable for tracks 1 to 32, play or stop.



## 76.5 - OSC: FEEDBACK AND STRIP-TYPES VALUES

*/set\_surface* has two values the user needs to calculate before use. In general these will not be calculated at run time, but beforehand. There may be more than one button with different values to turn various kinds of feedback on or off or to determine which kinds of strips are currently viewed/controlled.

Both *feedback* and *strip-types* use bitsets to keep track what they are doing. Any number in a computer is made out of bits that are on or off, but we represent them as normal base 10 numbers. Any one bit turned on will add a unique value to the number as a whole. So for each kind of feedback or strip type to be used, that number should be added to the total.

### 130.1 strip\_types

*strip\_types* is an integer made up of bits. The easy way to deal with this is to think of *strip\_types* items being worth a number and then adding all those numbers together for a value to send. Strip Types will determine What kind of strips will be included in bank. This would include: Audio, MIDI, busses, VCAs, Master, Monitor and hidden or selected strips.

- 1: AudioTracks.
- 2: MidiTracks.
- 4: AudioBusses.
- 8: MidiBusses.
- 16: VCAs.
- 32: Master.
- 64: Monitor.
- 128: Audio Aux.
- 256: Selected.
- 512: Hidden.
- 1024: Use Group.

Selected and Hidden bits are normally not needed as Ardour defaults to showing Selected strips and not showing Hidden strips. The purpose of these two flags is to allow showing only Selected strips or only Hidden strips. Using Hidden with other flags will allow Hidden strips to show inline with other strips.

Use Group on will tell ardour that any control on a strip that is part of a group will affect all strips within that group. Default is off or the control should only affect the strip the control is applied to. The */use\_group f state* command can be used to temporarily change this on the fly.

Some handy numbers to use might be: 15 (all tracks and busses - 1 + 2 + 4 + 8), 31 (add VCAs to that - 15 + 16). Master or Monitor strips are generally not useful on a surface that has dedicated controls for these strips as there are */master\** and */monitor\** commands already. However, on a surface with just a bank

of fader strips, adding master or monitor would allow access to them within the banks. Selected would be useful for working on a group or a set of user selected strips. Hidden shows strips the GUI has hidden.

Audio Aux? say what? I am sure most people will have noticed that they can find no *Aux* strips in the Ardour mixer. There are none. There are busses that can be used a number of ways. From analog days, in OSC, a bus is something that gets used as a sub mix before ending up going to Master. An auxiliary bus is used like a separate mixer and its output goes outside the program or computer to be used as: a monitor mix, a back up recording, or what have you. In OSC where controller strips may be limited, it may be useful not to use up a strip for an aux that is not really a part of the mix. It is also useful to get a list of only aux busses if the control surface is a phone used to provide talent monitor mix control on stage. Each performer would be able to mix their own monitor. The user is free to enable both busses and auxes if they would prefer.

## 130.2 feedback

Feedback is an integer made up of bits. The easy way to deal with this is to think of feedback items being worth a number and then adding all those numbers together for a value to send.

- 1: Button status for strips.
- 2: Variable control values for strips.
- 4: Send SSID as path extension.
- 8: heartbeat to surface.
- 16: Enable master section feedback.
- 32: Send Bar and Beat.
- 64: Send timecode.
- 128: Send meter as dB (-193 to +6) or 0 to 1 depending on gainmode
- 256: Send meter a 16 bit value where each bit is a level and all bits of lower level are on. For use in a LED strip. This will not work if the above option is turned on.
- 512: Send signal present, true if level is higher than -40dB
- 1024: Send position in samples
- 2048: Send position in time, hours, minutes, seconds and milliseconds
- 8192: Turn on select channel feedback
- 16384: Use OSC 1.0 /reply instead of #reply

So using a value of 19 (1 + 2 + 16) would turn on feedback for strip and master controls, but leave meters, timecode and bar/beat feedback off.

## 76.6 - OSC: JOG MODES

The `/jog` command will have a different affect depending on which jog mode is selected. The jog system has two commands and gives feedback of the mode chosen.

<code>/jog delta</code>	Where <i>delta</i> is a float indicating the amount and direction.
<code>/jog/mode mode</code>	Where <i>mode</i> is an int from 0 to 7 indicating the mode

Feedback is as below

<code>/jog/mode/name name</code>	Where <i>name</i> is a string indicating the name of the current jog mode.
<code>/jog/mode mode</code>	Where <i>mode</i> is an int from 0 to 7 indicating the current jog mode.

### 131.1 Jog Modes

- 0 Jog, each tick moves the Playhead forward or backward .2 seconds.
- 1 Nudge, Moves the Playhead forward or backward by the amount of the nudge clock.
- 2 Scrub, see *Scrub mode*.
- 3 Shuttle, each tick raises or lowers the transport speed by 12.5%.
- 4 Marker, Moves the Playhead to the previous or next Marker.
- 5 Scroll, each tick scrolls the edit window by one, forward or back.
- 6 Track, moves the current bank left or right by one strip.
- 7 Bank, Moves the current bank left or right by one bank.

The jog mode may be set using a slider with 0 to 7 limits, a group of switches or radio buttons. What works in any situation will depend on the controller.

### 131.2 Scrub

Scrub deserves special mention. In an ideal world, scrub would be jog with sound. However, Ardour does not have that functionality yet. So scrub starts the transport rolling at either 50% or 100% depending on how fast the jog wheel is turned. The position of the last tick is always saved and if no more ticks are received, the transport is located there when stopped at time out. If the jog wheel gives a value of 0 when released the transport stops at the location the value 0 is sent.



## 76.7 - OSC: AUTOMATION

Ardour has automation modes for many of its controls. As of version 5.9, OSC can control what automation mode a fader uses. (*See Automation.*)

The form of the automation mode command is:

```
/strip/[control]/automation ii ssid mode
```

- /strip may also be /select in which case the only parameter is the mode.
- [control] as of Ardour version 5.9 control can be:
  - gain
  - fader

This list will expand.

- ssid can be a parameter as shown or inline (automation/[ssid]). /select has no ssid.
- mode can be one of:
  - 0 Manual mode
  - 1 Play mode
  - 2 Write mode
  - 3 Touch mode

The mode value may be sent as a float allowing a “pot” or “slider” with a range of 0 to 3 to be used to control mode.

The next version of Ardour will add `/strip/[control]/automation_name is ssid mode_name` as feedback. A surface may choose to use only the first character of the string (M, P, W or T) instead of the whole string (this is in git now).

The touch mode needs more input so there is a Touch command as well (added post 5.9). It is almost identical to the automation command:

```
/strip/[control]/touch ii ssid touch
```

The only difference is the last parameter is 1 for touched and 0 for touch released. All of the rest of the explanation above applies.



## 76.8 - OSC: PERSONAL MONITORING CONTROL

Personal monitoring can allow a performer with a smart phone to set their personal monitor mix for a floor wedge or in-ear monitoring. In Ardour 5.6 OSC commands to allow this were added.

### 133.1 Setup

Some setup needs to be done in the GUI mixer window before this can work.

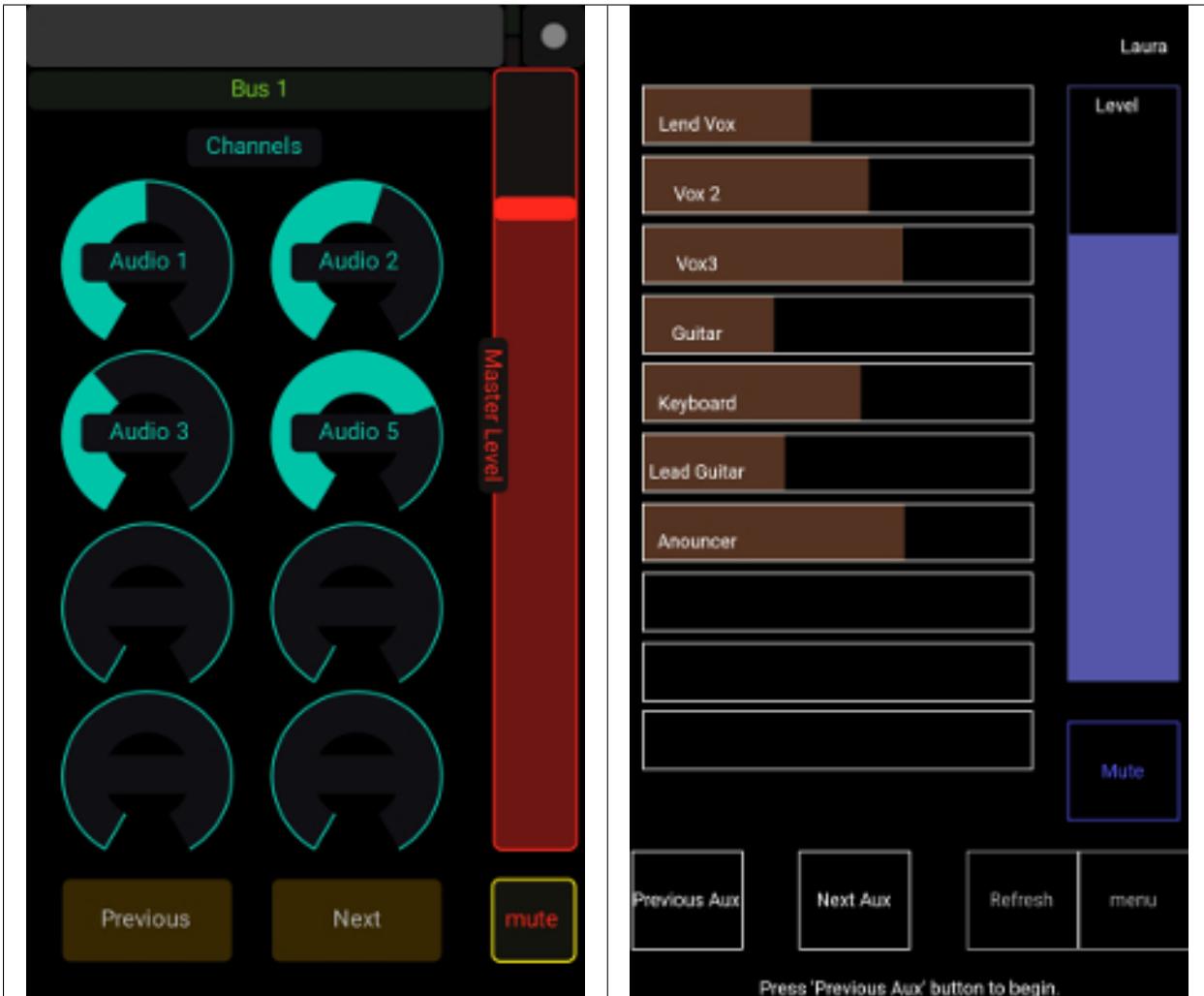
- Create a bus for each performer who will have personal monitoring. A good practice is to name the bus with the performers name.
- Connect the output of that bus to one of the audio interface's playback ports that is not otherwise used. OSC will now include this bus in its list of aux busses as it no longer has its output connected to the Master bus.
- Add an aux send to each channel the performer needs to hear in their personal mix. Many performers only need three or four sources to be mixed. If the performer needs to hear a set of inputs that are combined into a bus, adding the aux send to that bus may make more sense than adding ten drum channels for example.
- If the performer wishes to hear effects in their monitor, an extra send from the send bus, placing the performers aux send after the effect or a plugin can be added in line in the aux bus itself.

This gives stage or studio monitoring for the performer.

### 133.2 The OSC commands and feedback for personal monitoring

All of the personal monitoring commands and feedback start with a */cue*. It is expected that a surface used as a personal monitor control will use only */cue* commands.

Most phone OSC applets (TouchOSC, Control) require manual port to be set. There are certainly more controls than needed. Using send enables for example, may lead to wasted time discovering why a send has no sound. A good easy to use controller that fits on most phones while still being controllable even with big fingers might look like:



Ardour is not limited to talking to one personal monitor controller at a time, but is able to deal with many simultaneously, each controlling its own Aux bus.

The send controls and feedback all have the send id (1 to n) in line as part of the OSC path. So the path for the second send would be `/cue/send/fader/2` to set the level. It is considered that most surfaces used for this will only be able to handle one parameter.

### 133.2.1 Commands

/cue/connect	Returns a list of aux busses and connects to the first.
/cue/aux <i>aux-number</i>	where <i>aux-number</i> is an integer or float which is the aux bus number this surface will use.
/cue/next_aux	Sets the aux bus to one bus higher.
/cue/previous_aux	Sets the aux to one bus lower. This can also be used as a “connect” button to save space in a phone layout.
/cue/fader <i>position</i>	where <i>position</i> is a float for the position of the fader between 0.0 and 1.0.
/cue/mute <i>state</i>	where <i>state</i> is a float of 0.0 for mute off and 1.0 for the Aux bus mute on.
/cue/send/fader/ <i>id</i> <i>position</i>	where <i>position</i> is a float for the position of the fader between 0.0 and 1.0.
/cue/send/enable/ <i>id</i> <i>state</i>	where <i>state</i> is a float of 0.0 for disable and 1.0 for enable.

### 133.2.2 Feedback

/cue/name <i>name</i>	where <i>name</i> is a string that is the name of the currently selected aux bus.
/cue/name/ <i>id</i> <i>name</i>	where <i>name</i> is a string that is the name of the aux bus that <i>id</i> belongs to.
/cue/fader <i>position</i>	where <i>position</i> is a float from 0.0 to 1.0 that shows the fader position for the selected aux bus.
/cue/mute <i>state</i>	where <i>state</i> is a float of 0.0 or 1.0 that shows the state of the mute for the selected aux bus.
/cue/signal <i>activity</i>	where <i>activity</i> is a float of 0.0 or 1.0 that shows audio activity for the selected aux bus.
/cue/send/name/ <i>id</i> <i>name</i>	where <i>name</i> is a string that is the name of the channel that send <i>id</i> belongs to.
/cue/send/fader/ <i>id</i> <i>position</i>	where <i>position</i> is a float from 0.0 to 1.0 that is the position for the fader for the send that <i>id</i> belongs to.
/cue/send/enable/ <i>id</i> <i>state</i>	where <i>state</i> is a float of 0.0 or 1.0 that is the state of the enable for the send that <i>id</i> belongs to.

While a fader is being adjusted, the corresponding //*name*\* text will give the level in db.



## 76.9 - OSC: PARAMETER TYPES

An OSC message is laid out in this form:

/path/of/command type parameter

The type is there to indicate what the parameter is. This gives the idea that parameter types are quite strict and if the command requires an Integer “*i*” then the controller had better send it. However, the checking of the parameter type is left to the receiving software.

What this means in practical terms is that the surface can get away with sending the wrong type of parameter. There are some places where that just doesn’t make sense. For example, a parameter that is specified as a Float with a range of 0 to 1, could be sent as an Integer, but would only have full scale and minimum value with nothing in between. This is not much use for a fader, though OK for a button.

There are a number of OSC controllers based on iOS and Android tablets that only send or receive parameters as floats or text. These controllers should have no problem sending bool or int values as floats. Ardour will interpret the values as required.



## 76.10 - OSC: SELECTION AND EXPANSION CONSIDERATIONS

Ardour does not send every possible feedback value for each channel. It does send expanded information on the selected channel. There are also extra commands for the selected strip. All the feedback and select commands have their own path `/select`. This means that for the selected channel the surface does not have to keep track of the strip ID. The `/select` strip will follow the “current mixer strip” in the GUI editor window.

There are two major uses for this:

1. Single strip control surfaces. Using `/access_action Editor/select-next-route` or `/access_action Editor/select-prev-route` to step through the mixer strips.
2. Using a “Super strip” section of knobs to control parts of the strip that are changed less often such as polarity, sends or plugin parameters.

Selection in Ardour’s OSC implementation are complicated by the possibility of using more than one OSC controller at the same time. User “A” may select strip 4 and use a selected controller to make changes to that strip. User “B” may subsequently select strip 7 to make changes on. This leaves user “A” making changes to strip 7 which they did not choose.

For this reason Ardour offers local expansion aside from the GUI selection. Local expansion only affects the one OSC controller. GUI selection is global and affects all controllers using GUI selection as well as the GUI.

Both select and expansion use the `/select` set of commands.

In general, in a one user situation where that one user may use either the OSC surface or the GUI, using GUI based selection makes the most sense. This is the default because this is the more common use.

When there is more than one operator, then expansion only is the mode of choice. It may make sense for one of the surfaces to use GUI selection where the operator is also using the GUI for some things. However, the set up should be carefully analyzed for the possibility of selection confusions. Expansion should be considered the *safe* option.

It is always OK to use expansion on the surface even in a one user scenario. This allows the user to use GUI and surface selection for different uses.

It is also possible to use both if desired. `/strip/select` will always set the GUI select, but `/strip/expand` will set the select feedback and commands locally without changing the GUI select. Another `/strip/expand` or a `/strip/select` will override that expand command and releasing the `/strip/expand` or `/select/expand` (setting it to 0 or false) will set the `/select` set of commands and feedback back to whichever strip the GUI has selected at that time. This could be used to switch between the GUI select and the local expand to compare two strips settings.



## 76.11 - OSC CONTROL FOR ARDOUR 4.7 AND PRIOR

This page is what was available before version 5.\* was added. It has not been updated to make sure all 4.7 functionality is accurately represented. This page will vanish soon.

OSC lets synthesizers and other devices communicate with Ardour. OSC devices can send commands relating to playback (such as play or stop), performance (such as volume, play, stop, and almost any other function (such as Edit, or Undo).

Ardour is probably one of the most OSC-controllable audio applications around, but as with all OSC-controllable apps, you can't do much without knowing what messages can be sent. This document describes the various categories of messages that Ardour understands. It is subject to change, particularly the "Actions" part below, since this relates to the GTK GUI for Ardour rather than the backend.

### 136.1 Connecting to Ardour via OSC

OSC support is not enabled by default, but can be turned on via Edit > Preferences > Control Surfaces. Once enabled, Ardour will listen on port 3819 by default. This port number can be changed by editing `$ARDOUR_CONFIG` and adding this line within the `<Config>` section:

```
<Option name="osc-port" value="Your choice here"/>
```

## 136.2 List of OSC messages

### 136.2.1 Transport Control

/ardour/transport_stop	
/ardour/transport_play	
/ardour/set_transport_speed <i>s</i>	where <i>s</i> is a float ranging from -8.0f to 8.0f
/ardour/ffwd	
/ardour/rewind	
/ardour/goto_start	
/ardour/goto_end	
/ardour/add_marker	(adds marker to the current transport position)
/ardour/next_marker	
/ardour/prev_marker	
/ardour/locate <i>spos</i> <i>roll</i>	where <i>spos</i> is the target position in samples and <i>roll</i> is a bool/integer defining whether you want transport to be kept rolling or not
/ardour/loop_toggle	

### 136.2.2 Editing-related

/ardour/undo	
/ardour/redo	
/ardour/save_state	(this is the regular Session > Save operation)

### 136.2.3 Recording control

/ardour/toggle_punch_in	
/ardour/toggle_punch_out	
/ardour/rec_enable_toggle	
/ardour/toggle_all_rec_enables	(toggles all tracks' recording state)

### 136.2.4 Track specific operations

For each of the following, *rid* is the remote ID or the track

/ardour/routes/mute <i>rid</i>	where <i>mute_st</i> is a bool/int representing the desired mute state of the track
/ardour/routes/solo <i>rid</i>	where <i>solo_st</i> is a bool/int representing the desired solo state of the track
/ardour/routes/recenable <i>rid rec_st</i>	where <i>rec_st</i> is a bool/int representing the desired rec state of the track
/ardour/routes/gainabs <i>rid gain_abs</i>	where <i>gain_abs</i> is a float ranging from 0 to 2 (0 being -infinite, 1 being 0dB and 2 being +6dB).
/ardour/routes/gaindB <i>rid gain_db</i>	where <i>gain_db</i> is a float ranging from -400 to 6 representing the desired gain of the track in dB.
/ardour/routes/trimabs <i>rid trim_abs</i>	where <i>trim_abs</i> is a float ranging from 0.1 to 10 (-20dB to +20dB). (since 4.1)
/ardour/routes/trimdB <i>rid trim_db</i>	where <i>trim_db</i> is a float ranging from -20 to 20 representing the desired trim of the track in dB. (since 4.1)

### 136.2.5 Menu actions

Every single menu item in Ardour's GUI is accessible via OSC. There is a single common syntax to trigger the action as if it was selected with the mouse (or keyboard):

```
/ardour/access_action action_name
```

The list below shows all available values of *action-name* as of mid-February 2014 for Ardour 3.5. You can get the current list at any time by running Ardour with the -b flag.

Action Name	Menu Name
Common/Chat	Chat
Common/KeepTearoffs	Show Toolbars
Common/Manual	Manual
Common/NewMIDITracer	MIDI Tracer
Common/Quit	Quit
Common/Reference	Reference
Common/Save	Save
Common/toggle-editor-mixer	Toggle Editor+Mixer
Common/ToggleMaximalEditor	Maximise Editor Space
Common/toggle-meterbridge	Meterbridge
Common/toggle-mixer	Mixer
Common/ToggleRecordEnableTrack10	Toggle Record Enable Track 10
Common/ToggleRecordEnableTrack11	Toggle Record Enable Track 11
Common/ToggleRecordEnableTrack12	Toggle Record Enable Track 12
Common/ToggleRecordEnableTrack13	Toggle Record Enable Track 13
Common/ToggleRecordEnableTrack14	Toggle Record Enable Track 14
Common/ToggleRecordEnableTrack15	Toggle Record Enable Track 15
Common/ToggleRecordEnableTrack16	Toggle Record Enable Track 16
Common/ToggleRecordEnableTrack17	Toggle Record Enable Track 17
Common/ToggleRecordEnableTrack18	Toggle Record Enable Track 18
Common/ToggleRecordEnableTrack19	Toggle Record Enable Track 19
Common/ToggleRecordEnableTrack1	Toggle Record Enable Track 1
Common/ToggleRecordEnableTrack20	Toggle Record Enable Track 20
Common/ToggleRecordEnableTrack21	Toggle Record Enable Track 21
Common/ToggleRecordEnableTrack22	Toggle Record Enable Track 22

Continued on next page

Table 1 – continued from previous page

Common/ToggleRecordEnableTrack23	Toggle Record Enable Track 23
Common/ToggleRecordEnableTrack24	Toggle Record Enable Track 24
Common/ToggleRecordEnableTrack25	Toggle Record Enable Track 25
Common/ToggleRecordEnableTrack26	Toggle Record Enable Track 26
Common/ToggleRecordEnableTrack27	Toggle Record Enable Track 27
Common/ToggleRecordEnableTrack28	Toggle Record Enable Track 28
Common/ToggleRecordEnableTrack29	Toggle Record Enable Track 29
Common/ToggleRecordEnableTrack2	Toggle Record Enable Track 2
Common/ToggleRecordEnableTrack30	Toggle Record Enable Track 30
Common/ToggleRecordEnableTrack31	Toggle Record Enable Track 31
Common/ToggleRecordEnableTrack32	Toggle Record Enable Track 32
Common/ToggleRecordEnableTrack3	Toggle Record Enable Track 3
Common/ToggleRecordEnableTrack4	Toggle Record Enable Track 4
Common/ToggleRecordEnableTrack5	Toggle Record Enable Track 5
Common/ToggleRecordEnableTrack6	Toggle Record Enable Track 6
Common/ToggleRecordEnableTrack7	Toggle Record Enable Track 7
Common/ToggleRecordEnableTrack8	Toggle Record Enable Track 8
Common/ToggleRecordEnableTrack9	Toggle Record Enable Track 9
Editor/addExistingAudioFiles	Import
Editor/addExternalAudioToRegionLi st	Import to Region List...
Editor/add-location-from-playhead	Add Mark from Playhead
Editor/center-edit-cursor	Center Edit Point
Editor/center-playhead	Center Playhead
Editor/crop	Crop
Editor/cycle-edit-point	Change Edit Point
Editor/cycle-edit-point-with-mark er	Change Edit Point Including Marker
Editor/cycle-snap-mode	Next Snap Mode
Editor/cycle-zoom-focus	Next Zoom Focus
Editor/deselect-all	Deselect All
Editor/duplicate-range	Duplicate Range
Editor/edit-at-mouse	Mouse
Editor/edit-at-playhead	Playhead
Editor/edit-at-selected-marker	Marker
Editor/edit-cursor-to-next-region -end	To Next Region End
Editor/edit-cursor-to-next-region -start	To Next Region Start
Editor/edit-cursor-to-next-region -sync	To Next Region Sync
Editor/edit-cursor-to-previous-re gion-end	To Previous Region End
Editor/edit-cursor-to-previous-re gion-start	To Previous Region Start
Editor/edit-cursor-to-previous-re gion-sync	To Previous Region Sync
Editor/edit-cursor-to-range-end	To Range End
Editor/edit-cursor-to-range-start	To Range Start
Editor/editor-copy	Copy
Editor/editor-crop	Crop
Editor/editor-cut	Cut
Editor/editor-delete	Delete
Editor/editor-paste	Paste
Editor/editor-separate	Separate
Editor/edit-to-playhead	Active Mark to Playhead
Editor/escape	Break drag or deselect all
Editor/expand-tracks	Expand Track Height

Continued on next page

Table 1 – continued from previous page

Editor/export-audio	Export Audio
Editor/export-range	Export Range
Editor/finish-add-range	Finish Add Range
Editor/finish-range	Finish Range
Editor/fit-tracks	Fit Selected Tracks
Editor/goto-mark-1	Locate to Mark 1
Editor/goto-mark-2	Locate to Mark 2
Editor/goto-mark-3	Locate to Mark 3
Editor/goto-mark-4	Locate to Mark 4
Editor/goto-mark-5	Locate to Mark 5
Editor/goto-mark-6	Locate to Mark 6
Editor/goto-mark-7	Locate to Mark 7
Editor/goto-mark-8	Locate to Mark 8
Editor/goto-mark-9	Locate to Mark 9
Editor/goto-visual-state-10	Goto View 10
Editor/goto-visual-state-11	Goto View 11
Editor/goto-visual-state-12	Goto View 12
Editor/goto-visual-state-1	Goto View 1
Editor/goto-visual-state-2	Goto View 2
Editor/goto-visual-state-3	Goto View 3
Editor/goto-visual-state-4	Goto View 4
Editor/goto-visual-state-5	Goto View 5
Editor/goto-visual-state-6	Goto View 6
Editor/goto-visual-state-7	Goto View 7
Editor/goto-visual-state-8	Goto View 8
Editor/goto-visual-state-9	Goto View 9
Editor/importFromSession	Import From Session
Editor/insert-time	Insert Time
Editor/invert-selection	Invert Selection
Editor/jump-backward-to-mark	Jump to Previous Mark
Editor/jump-forward-to-mark	Jump to Next Mark
Editor/main-menu-play-selected-regions	Play Selected Regions
EditorMenu/AlignMenu	Align
EditorMenu/Autoconnect	Autoconnect
EditorMenu/Crossfades	Crossfades
EditorMenu/EditCursorMovementOptions	Move Selected Marker
EditorMenu/Edit	Edit
EditorMenu/EditPointMenu	Edit Point
EditorMenu/EditSelectRangeOptions	Select Range Operations
EditorMenu/EditSelectRegionOptions	Select Regions
EditorMenu/FadeMenu	Fade
EditorMenu/LatchMenu	Latch
EditorMenu/Link	Link
EditorMenu/LocateToMarker	Locate to Markers
EditorMenu/MarkerMenu	Markers
EditorMenu/MeterFalloff	Meter falloff
EditorMenu/MeterHold	Meter hold
EditorMenu/MIDI	MIDI Options
EditorMenu/MiscOptions	Misc Options
EditorMenu/Monitoring	Monitoring

Continued on next page

Table 1 – continued from previous page

EditorMenu/MoveActiveMarkMenu	Active Mark
EditorMenu/MovePlayHeadMenu	Playhead
EditorMenu/PlayMenu	Play
EditorMenu/PrimaryClockMenu	Primary Clock
EditorMenu/Pullup	Pullup / Pulldown
EditorMenu/RegionEditOps	Region operations
EditorMenu/RegionGainMenu	Gain
EditorMenu/RegionMenuDuplicate	Duplicate
EditorMenu/RegionMenuEdit	Edit
EditorMenu/RegionMenuFades	Fades
EditorMenu/RegionMenuGain	Gain
EditorMenu/RegionMenu	Region
EditorMenu/RegionMenuLayering	Layering
EditorMenu/RegionMenuMIDI	MIDI
EditorMenu/RegionMenuPosition	Position
EditorMenu/RegionMenuRanges	Ranges
EditorMenu/RegionMenuTrim	Trim
EditorMenu/RulerMenu	Rulers
EditorMenu/SavedViewMenu	Views
EditorMenu/ScrollMenu	Scroll
EditorMenu/SecondaryClockMenu	Secondary Clock
EditorMenu>Select	Select
EditorMenu>SelectMenu	Select
EditorMenu/SeparateMenu	Separate
EditorMenu/SetLoopMenu	Loop
EditorMenu/SetPunchMenu	Punch
EditorMenu/Solo	Solo
EditorMenu/Subframes	Subframes
EditorMenu/SyncMenu	Sync
EditorMenu/TempoMenu	Tempo
EditorMenu/Timecode	Timecode fps
EditorMenu/Tools	Tools
EditorMenu/TrackHeightMenu	Height
EditorMenu/TrackMenu	Track
EditorMenu/VideoMonitorMenu	Video Monitor
EditorMenu/View	View
EditorMenu/ZoomFocus	Zoom Focus
EditorMenu/ZoomFocusMenu	Zoom Focus
EditorMenu/ZoomMenu	Zoom
Editor/move-range-end-to-next-region-boundary	Move Range End to Next Region Boundary
Editor/move-range-end-to-previous-region-boundary	Move Range End to Previous Region Boundary
Editor/move-range-start-to-next-region-boundary	Move Range Start to Next Region Boundary
Editor/move-range-start-to-previous-region-boundary	Move Range Start to Previous Region Boundary
Editor/move-selected-tracks-down	Move Selected Tracks Down
Editor/move-selected-tracks-up	Move Selected Tracks Up
Editor/next-snap-choice	Next Snap Choice
Editor/next-snap-choice-music-only	Next Musical Snap Choice
Editor/nudge-next-backward	Nudge Next Earlier
Editor/nudge-next-forward	Nudge Next Later
Editor/nudge-playhead-backward	Nudge Playhead Backward

Continued on next page

Table 1 – continued from previous page

Editor/nudge-playhead-forward	Nudge Playhead Forward
Editor/play-edit-range	Play Edit Range
Editor/play-from-edit-point-and-return	Play from Edit Point and Return
Editor/play-from-edit-point	Play From Edit Point
Editor/playhead-backward-to-grid	Playhead To Previous Grid
Editor/playhead-forward-to-grid	Playhead To Next Grid
Editor/playhead-to-edit	Playhead to Active Mark
Editor/playhead-to-next-region-boundary	Playhead to Next Region Boundary
Editor/playhead-to-next-region-boundary-noselection	Playhead to Next Region Boundary (No Track Selection)
Editor/playhead-to-next-region-end	Playhead to Next Region End
Editor/playhead-to-next-region-start	Playhead to Next Region Start
Editor/playhead-to-next-region-sync	Playhead to Next Region Sync
Editor/playhead-to-previous-region-boundary	Playhead to Previous Region Boundary
Editor/playhead-to-previous-region-boundary-noselection	Playhead to Previous Region Boundary (No Track Selection)
Editor/playhead-to-previous-region-end	Playhead to Previous Region End
Editor/playhead-to-previous-region-start	Playhead to Previous Region Start
Editor/playhead-to-previous-region-sync	Playhead to Previous Region Sync
Editor/playhead-to-range-end	Playhead to Range End
Editor/playhead-to-range-start	Playhead to Range Start
Editor/prev-snap-choice	Previous Snap Choice
Editor/prev-snap-choice-music-only	Previous Musical Snap Choice
Editor/redo	Redo
Editor/remove-last-capture	Remove Last Capture
Editor/remove-track	Remove
Editor/save-visual-state-10	Save View 10
Editor/save-visual-state-11	Save View 11
Editor/save-visual-state-12	Save View 12
Editor/save-visual-state-1	Save View 1
Editor/save-visual-state-2	Save View 2
Editor/save-visual-state-3	Save View 3
Editor/save-visual-state-4	Save View 4
Editor/save-visual-state-5	Save View 5
Editor/save-visual-state-6	Save View 6
Editor/save-visual-state-7	Save View 7
Editor/save-visual-state-8	Save View 8
Editor/save-visual-state-9	Save View 9
Editor/scroll-backward	Scroll Backward
Editor/scroll-forward	Scroll Forward
Editor/scroll-playhead-backward	Playhead Backward
Editor/scroll-playhead-forward	Playhead Forward
Editor/scroll-tracks-down	Scroll Tracks Down
Editor/scroll-tracks-up	Scroll Tracks Up
Editor/select-all-after-edit-cursor	Select All After Edit Point
Editor/select-all-before-edit-cursor	Select All Before Edit Point
Editor/select-all-between-cursors	Select All Overlapping Edit Range
Editor/select-all-in-loop-range	Select All in Loop Range
Editor/select-all-in-punch-range	Select All in Punch Range
Editor/select-all	Select All
Editor/select-all-within-cursors	Select All Inside Edit Range
Editor/selected-marker-to-next-region-boundary	To Next Region Boundary

Continued on next page

Table 1 – continued from previous page

Editor/selected-marker-to-next-region-boundary-noselection	To Next Region Boundary (No Track Selection)
Editor/selected-marker-to-previous-region-boundary	To Previous Region Boundary
Editor/selected-marker-to-previous-region-boundary-noselection	To Previous Region Boundary (No Track Selection)
Editor/select-next-route	Select Next Track or Bus
Editor/select-prev-route	Select Previous Track or Bus
Editor/select-range-between-cursors	Select Edit Range
Editor/separate-from-loop	Separate Using Loop Range
Editor/separate-from-punch	Separate Using Punch Range
Editor/set-edit-lock	Lock
Editor/set-edit-point	Active Marker to Mouse
Editor/set-edit-slide	Slide
Editor/set-edit-splice	Splice
Editor/set-loop-from>Edit Range	Set Loop from Edit Range
Editor/set-playhead	Playhead to Mouse
Editor/set-punch-from>Edit Range	Set Punch from Edit Range
Editor/set-tempo-from>Edit Range	Set Tempo from Edit Range = Bar
Editor/show-editor-list	Show Editor List
Editor/show-editor-mixer	Show Editor Mixer
Editor/show-marker-lines	Show Marker Lines
Editor/shrink-tracks	Shrink Track Height
Editor/snap-magnetic	Magnetic
Editor/SnapMode	Snap Mode
Editor/snap-normal	Grid
Editor/snap-off	No Grid
Editor/SnapTo	Snap to
Editor/sound-midi-notes	Sound Selected MIDI Notes
Editor/start-range	Start Range
Editor/step-mouse-mode	Step Mouse Mode
Editor/step-tracks-down	Step Tracks Down
Editor/step-tracks-up	Step Tracks Up
Editor/tab-to-transient-backwards	Move Earlier to Transient
Editor/tab-to-transient-forwards	Move Later to Transient
Editor/temporal-zoom-in	Zoom In
Editor/temporal-zoom-out	Zoom Out
Editor/toggle-edit-mode	Toggle Edit Mode
Editor/toggle-follow-playhead	Follow Playhead
Editor/ToggleGroupTabs	Show Group Tabs
Editor/ToggleJadeo	Video Monitor
Editor/ToggleLogoVisibility	Show Logo
Editor/toggle-log-window	Log
Editor/ToggleMeasureVisibility	Show Measures
Editor/toggle-midi-input-active	Toggle MIDI Input Active for Editor-Selected Tracks/Buses
Editor/toggle-stationary-playhead	Stationary Playhead
Editor/ToggleSummary	Show Summary
Editor/toggle-track-active	Toggle Active
Editor/toggle-vmon-frame	Frame number
Editor/toggle-vmon-fullscreen	Fullscreen
Editor/toggle-vmon-letterbox	Letterbox
Editor/toggle-vmon-on-top	Always on Top
Editor/toggle-vmon-osdbg	Timecode Background

Continued on next page

Table 1 – continued from previous page

Editor/toggle-vmon-timecode	Timecode
Editor/toggle-zoom	Toggle Zoom State
Editor/track-height-large	Large
Editor/track-height-larger	Larger
Editor/track-height-largest	Largest
Editor/track-height-normal	Normal
Editor/track-height-small	Small
Editor/track-mute-toggle	Toggle Mute
Editor/track-record-enable-toggle	Toggle Record Enable
Editor/track-solo-isolate-toggle	Toggle Solo Isolate
Editor/track-solo-toggle	Toggle Solo
Editor/undo	Undo
Editor/zoom-to-region-both-axes	Zoom to Region (Width and Height)
Editor/zoom-to-region	Zoom to Region
Editor/zoom-to-session	Zoom to Session
Editor/zoom-vmon-100	Original Size
Main/AddTrackBus	Add Track or Bus...
Main/CleanupUnused	Clean-up Unused Sources...
Main/Close	Close
Main/CloseVideo	Remove Video
Main/EditMetadata	Edit Metadata...
Main/ExportAudio	Export To Audio File(s)...
Main/Export	Export
Main/ExportVideo	Export To Video File
Main/FlushWastebasket	Flush Wastebasket
Main/ImportMetadata	Import Metadata...
Main_menu/AudioFileFormatData	Sample Format
Main_menu/AudioFileFormatHeader	File Type
Main_menu/AudioFileFormat	Audio File Format
Main_menu/Cleanup	Clean-up
Main_menu/ControlSurfaces	Control Surfaces
Main_menu/Denormals	Denormal Handling
Main_menu/Help	Help
Main_menu/KeyMouseActions	Misc. Shortcuts
Main_menu/MeteringFallOffRate	Fall Off Rate
Main_menu/MeteringHoldTime	Hold Time
Main_menu/Metering	Metering
Main_menu/Plugins	Plugins
Main_menu/Session	Session
Main_menu/Sync	Sync
Main_menu/TransportOptions	Options
Main_menu/WindowMenu	Window
Main_Metadata	Metadata
Main/New	New...
Main/Open	Open...
Main/OpenVideo	Open Video
Main/Recent	Recent...
Main/Rename	Rename...
Main/SaveAs	Save As...
Main/SaveTemplate	Save Template...

Continued on next page

Table 1 – continued from previous page

Main/Snapshot	Snapshot...
Main/StemExport	Stem export...
MIDI/panic	Panic
MouseMode/set-mouse-mode-audition	Audition Tool
MouseMode/set-mouse-mode-draw	Note Drawing Tool
MouseMode/set-mouse-mode-gain	Gain Tool
MouseMode/set-mouse-mode-object	Object Tool
MouseMode/set-mouse-mode-object-range	Smart Object Mode
MouseMode/set-mouse-mode-range	Range Tool
MouseMode/set-mouse-mode-timefx	Time FX Tool
MouseMode/set-mouse-mode-zoom	Zoom Tool
MouseMode/toggle-internal-edit	Edit MIDI
options/SendMidiClock	Send MIDI Clock
options/SendMIDIfeedback	Send MIDI Feedback
options/SendMMC	Send MMC
options/SendMTC	Send MTC
options/UseMMC	Use MMC
ProcessorMenu/ab_plugins	A/B Plugins
ProcessorMenu/activate_all	Activate All
ProcessorMenu/clear	Clear (all)
ProcessorMenu/clear_post	Clear (post-fader)
ProcessorMenu/clear_pre	Clear (pre-fader)
ProcessorMenu/controls	Controls
ProcessorMenu/copy	Copy
ProcessorMenu/cut	Cut
ProcessorMenu/deactivate_all	Deactivate All
ProcessorMenu/delete	Delete
ProcessorMenu/deselectall	Deselect All
ProcessorMenu/edit-generic	Edit with generic controls...
ProcessorMenu/edit	Edit...
ProcessorMenu/newaux	New Aux Send ...
ProcessorMenu/newinsert	New Insert
ProcessorMenu/newplugin	New Plugin
ProcessorMenu/newsend	New External Send ...
ProcessorMenu/paste	Paste
ProcessorMenu/rename	Rename
ProcessorMenu/selectall	Select All
ProcessorMenu/send_options	Send Options
Region/add-range-marker-from-region	Add Single Range Marker
Region/add-range-markers-from-region	Add Range Marker Per Region
Region/align-regions-end	Align End
Region/align-regions-end-relative	Align End Relative
Region/align-regions-start	Align Start
Region/align-regions-start-relative	Align Start Relative
Region/align-regions-sync	Align Sync
Region/align-regions-sync-relative	Align Sync Relative
Region/analyze-region	Spectral Analysis...
Region/boost-region-gain	Boost Gain
Region/bounce-regions-processed	Bounce (without processing)
Region/bounce-regions-unprocessed	Bounce (with processing)

Continued on next page

Table 1 – continued from previous page

Region/choose-top-region-context-menu	Choose Top...
Region/choose-top-region	Choose Top...
Region/close-region-gaps	Close Gaps
Region/combine-regions	Combine
Region/cut-region-gain	Cut Gain
Region/duplicate-region	Duplicate
Region/export-region	Export...
Region/fork-region	Unlink from other copies
Region/insert-patch-change-context	Insert Patch Change...
Region/insert-patch-change	Insert Patch Change...
Region/insert-region-from-region-list	Insert Region From Region List
RegionList/RegionListSort	Sort
RegionList/removeUnusedRegions	Remove Unused
RegionList/rlAudition	Audition
RegionList/rlHide	Hide
RegionList/rlShowAll	Show All
RegionList/rlShowAuto	Show Automatic Regions
RegionList/rlShow	Show
RegionList/SortAscending	Ascending
RegionList/SortByRegionEndinFile	By Region End in File
RegionList/SortByRegionLength	By Region Length
RegionList/SortByRegionName	By Region Name
RegionList/SortByRegionPosition	By Region Position
RegionList/SortByRegionStartinFile	By Region Start in File
RegionList/SortByRegionTimestamp	By Region Timestamp
RegionList/SortBySourceFileCreationDate	By Source File Creation Date
RegionList/SortBySourceFileLength	By Source File Length
RegionList/SortBySourceFileName	By Source File Name
RegionList/SortBySourceFilesystem	By Source Filesystem
RegionList/SortDescending	Descending
Region/loop-region	Loop
Region/lower-region	Lower
Region/lower-region-to-bottom	Lower to Bottom
Region/multi-duplicate-region	Multi-Duplicate...
Region/naturalize-region	Move to Original Position
Region/normalize-region	Normalize...
Region/nudge-backward-by-capture-offset	Nudge Earlier by Capture Offset
Region/nudge-backward	Nudge Earlier
Region/nudge-forward-by-capture-offset	Nudge Later by Capture Offset
Region/nudge-forward	Nudge Later
Region/pitch-shift-region	Pitch Shift...
Region/place-transient	Place Transient
Region/play-selected-regions	Play
Region/quantize-region	Quantize...
Region/raise-region	Raise
Region/raise-region-to-top	Raise to Top
Region/region-fill-track	Fill Track
Region/remove-region	Remove
Region/remove-region-sync	Remove Sync
Region/rename-region	Rename...

Continued on next page

Table 1 – continued from previous page

Region/reset-region-gain-envelope s	Reset Envelope
Region/reset-region-scale-amplitude	Reset Gain
Region/reverse-region	Reverse
Region/separate-under-region	Separate Under
Region/set-fade-in-length	Set Fade In Length
Region/set-fade-out-length	Set Fade Out Length
Region/set-loop-from-region	Set Loop Range
Region/set-punch-from-region	Set Punch
Region/set-region-sync-position	Set Sync Position
Region/set-selection-from-region	Set Range Selection
Region/set-tempo-from-region	Set Tempo from Region = Bar
Region/show-region-list-editor	List Editor...
Region/show-region-properties	Properties...
Region/show-rhythm-ferret	Rhythm Ferret...
Region/snap-regions-to-grid	Snap Position To Grid
Region/split-multichannel-region	Make Mono Regions
Region/split-region-at-transients	Split at Percussion Onsets
Region/split-region	Split
Region/strip-region-silence	Strip Silence...
Region/toggle-opaque-region	Opaque
Region/toggle-region-fade-in	Fade In
Region/toggle-region-fade-out	Fade Out
Region/toggle-region-fades	Fades
Region/toggle-region-gain-envelope-active	Envelope Active
Region/toggle-region-lock	Lock
Region/toggle-region-lock-style	Glue to Bars and Beats
Region/toggle-region-mute	Mute
Region/toggle-region-video-lock	Lock to Video
Region/transpose-region	Transpose...
Region/trim-back	Trim End at Edit Point
Region/trim-front	Trim Start at Edit Point
Region/trim-region-to-loop	Trim to Loop
Region/trim-region-to-punch	Trim to Punch
Region/trim-to-next-region	Trim to Next
Region/trim-to-previous-region	Trim to Previous
Region/uncombine-regions	Uncombine
Rulers/toggle-bbt-ruler	Bars & Beats
Rulers/toggle-cd-marker-ruler	CD Markers
Rulers/toggle-loop-punch-ruler	Loop/Punch
Rulers/toggle-marker-ruler	Markers
Rulers/toggle-meter-ruler	Meter
Rulers/toggle-minsec-ruler	Min:Sec
Rulers/toggle-range-ruler	Ranges
Rulers/toggle-samples-ruler	Samples
Rulers/toggle-tempo-ruler	Tempo
Rulers/toggle-timecode-ruler	Timecode
Rulers/toggle-video-ruler	Video
ShuttleActions/SetShuttleUnitsPercentage	Percentage
ShuttleActions/SetShuttleUnitsSemitones	Semitones
Snap/snap-to-asixteenthbeat	Snap to Sixteenths

Continued on next page

Table 1 – continued from previous page

Snap/snap-to-bar	Snap to Bar
Snap/snap-to-beat	Snap to Beat
Snap/snap-to-cd-frame	Snap to CD Frame
Snap/snap-to-eighths	Snap to Eighths
Snap/snap-to-fifths	Snap to Fifths
Snap/snap-to-fourteenths	Snap to Fourteenths
Snap/snap-to-halves	Snap to Halves
Snap/snap-to-mark	Snap to Mark
Snap/snap-to-minutes	Snap to Minutes
Snap/snap-to-onetwentyeighths	Snap to One Twenty Eighths
Snap/snap-to-quarters	Snap to Quarters
Snap/snap-to-region-boundary	Snap to Region Boundary
Snap/snap-to-region-end	Snap to Region End
Snap/snap-to-region-start	Snap to Region Start
Snap/snap-to-region-sync	Snap to Region Sync
Snap/snap-to-seconds	Snap to Seconds
Snap/snap-to-sevenths	Snap to Sevenths
Snap/snap-to-sixths	Snap to Sixths
Snap/snap-to-sixtyfourths	Snap to Sixty Fourths
Snap/snap-to-tenths	Snap to Tenths
Snap/snap-to-thirds	Snap to Thirds
Snap/snap-to-thirtyseconds	Snap to Thirty Seconds
Snap/snap-to-timecode-frame	Snap to Timecode Frame
Snap/snap-to-timecode-minutes	Snap to Timecode Minutes
Snap/snap-to-timecode-seconds	Snap to Timecode Seconds
Snap/snap-to-twelfths	Snap to Twelfths
Snap/snap-to-twentieths	Snap to Twentieths
Snap/snap-to-twentyeighths	Snap to Twenty Eighths
Snap/snap-to-twentyfourths	Snap to Twenty Fourths
Transport/focus-on-clock	Focus On Clock
Transport/ForwardFast	Forward (Fast)
Transport/Forward	Forward
Transport/ForwardSlow	Forward (Slow)
Transport/GotoEnd	Goto End
Transport/GotoStart	Goto Start
Transport/GotoWallClock	Goto Wall Clock
Transport/GotoZero	Goto Zero
Transport/Loop	Play Loop Range
Transport/PlayPreroll	Play Selection w/Preroll
Transport/PlaySelection	Play Selected Range
Transport/primary-clock-bbt	Bars & Beats
Transport/primary-clock-minsec	Minutes & Seconds
Transport/primary-clock-samples	Samples
Transport/primary-clock-timecode	Timecode
Transport/Record	Enable Record
Transport/record-roll	Start Recording
Transport/RewindFast	Rewind (Fast)
Transport/Rewind	Rewind
Transport/RewindSlow	Rewind (Slow)
Transport/Roll	Roll

Continued on next page

Table 1 – continued from previous page

Transport/secondary-clock-bbt	Bars & Beats
Transport/secondary-clock-minsec	Minutes & Seconds
Transport/secondary-clock-samples	Samples
Transport/secondary-clock-timecode	Timecode
Transport/Stop	Stop
Transport/ToggleAutoInput	Auto Input
Transport/ToggleAutoPlay	Auto Play
Transport/ToggleAutoReturn	Auto Return
Transport/ToggleClick	Click
Transport/ToggleExternalSync	
Transport/ToggleFollowEdits	Follow Edits
Transport/TogglePunchIn	Punch In
Transport/TogglePunch	Punch In/Out
Transport/TogglePunchOut	Punch Out
Transport/ToggleRollForgetCapture	Stop and Forget Capture
Transport/ToggleRoll	Start/Stop
Transport/ToggleRollMaybe	Start/Continue/Stop
Transport/ToggleTimeMaster	Time Master
Transport/ToggleVideoSync	Sync Startup to Video
Transport/TransitionToReverse	Transition To Reverse
Transport/TransitionToRoll	Transition To Roll
Transport/Transport	Transport
Window/toggle-about	About
Window/toggle-add-routes	Add Tracks/Busses
Window/toggle-add-video	Add Tracks/Busses
Window/toggle-audio-connection-manager	Audio Connections
Window/toggle-audio-midi-setup	Audio/MIDI Setup
Window/toggle-big-clock	Big Clock
Window/toggle-bundle-manager	Bundle Manager
Window/toggle-inspector	Tracks and Busses
Window/toggle-key-editor	Key Bindings
Window/toggle-locations	Locations
Window/toggle-midi-connection-manager	MIDI Connections
Window/toggle-rc-options-editor	Preferences
Window/toggle-session-options-editor	Properties
Window/toggle-speaker-config	Speaker Configuration
Window/toggle-theme-manager	Theme Manager
Zoom/zoom-focus-center	Zoom Focus Center
Zoom/zoom-focus-edit	Zoom Focus Edit Point
Zoom/zoom-focus-left	Zoom Focus Left
Zoom/zoom-focus-mouse	Zoom Focus Mouse
Zoom/zoom-focus-playhead	Zoom Focus Playhead
Zoom/zoom-focus-right	Zoom Focus Right

---

**CHAPTER  
SEVEN**

---

## **77 - CONTROLLING ARDOUR WITH MACKIE CONTROL DEVICES**

Since Mackie and Logic made the first Logic Control Surface, Mackie and other surface manufacturers have been making control surfaces that use the same protocol to communicate with a DAW. Ardour supports the MCP and will work with all these control surfaces.



## 77.1 - DEVICES USING MACKIE/LOGIC CONTROL PROTOCOL

This will walk you through the process of configuring and using a MIDI control surface with Ardour that uses the Mackie Control protocol (MCP) or Logic Control protocol. Devices that have been tested and are known to work include the SSL Nucleus, Mackie Control Pro (plus extenders), Behringer devices in Mackie/Logic mode, and Steinberg CMC devices.

### 138.1 Enabling Mackie Control in Ardour

Navigate to Edit > Preferences > Control Surfaces. Tick the Mackie option and click on the Show Protocol Settings button to see the setup dialog:

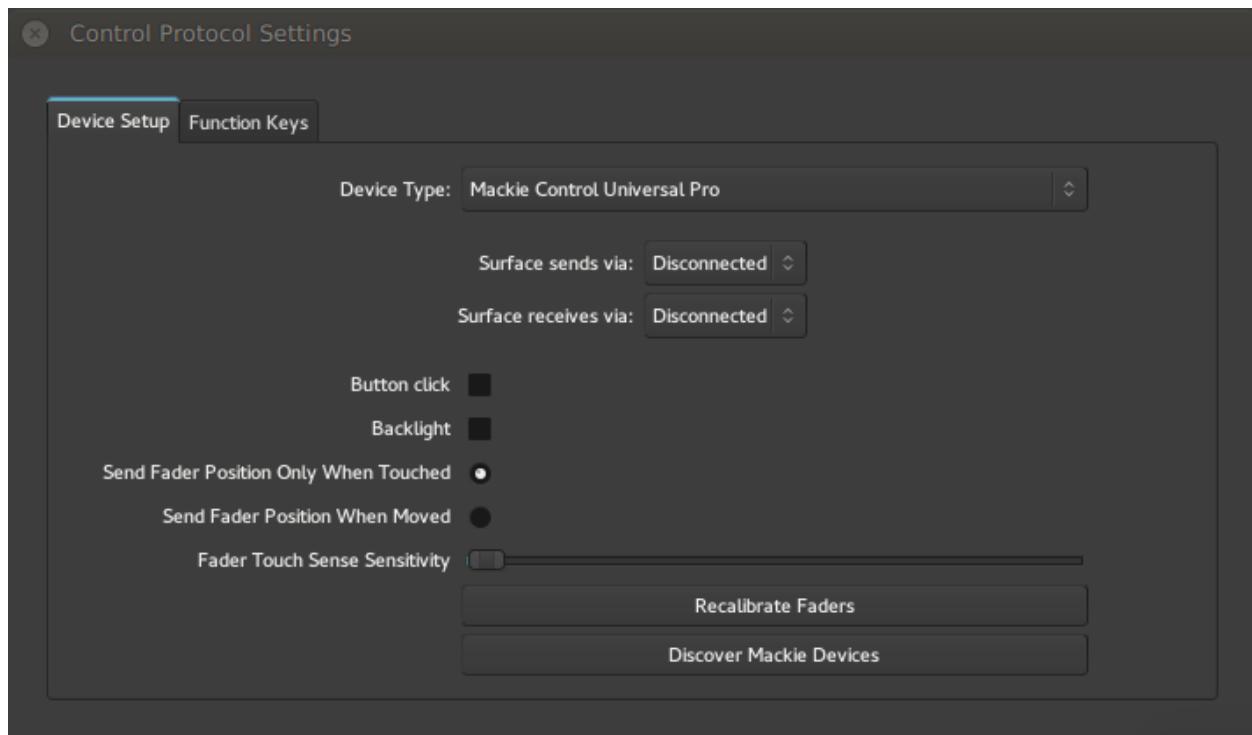


Fig. 1: The Mackie Control Setup Dialog

From the selector at the top, choose the type of device you are using. ([What to do if your device is not listed](#)).

Update the controls description

Once your setup is complete, click “OK” to close the dialog. Now click on the enable checkbox for “Mackie Control”.

## **138.2 Devices using ipMIDI**

If you are using a device that uses ipMIDI, such as the SSL Nucleus, no MIDI port connections are required—Ardour and your control surface will be able to talk to each other automatically so long as your control surface and computer are both connected to the same network.

## **138.3 Connecting control surface and Ardour MIDI ports**

Before attempting to use a Mackie Control device that communicates via a standard MIDI cable or a USB cable, you should ensure that *your Linux MIDI environment is setup*. If you are using a device that uses normal MIDI (via a standard MIDI or USB cable), you need to connect Ardour’s Mackie Control in and out ports to the MIDI ports leading to and coming from the control surface.

When you have made these connections once, Ardour will recreate them for you in the future, as long as you leave Mackie Control enabled.

## **138.4 Customizing your control surface**

Every possible global Mackie Control button can be bound to any *action* present in Ardour’s GUI. Please check your control surface page for suggestions.

## **138.5 Preparing your device for use with Ardour**

Most interfaces will require some configuration to send and respond to MCP.

When setting up the control surface, do *not* use “Pro Tools” mode. Pro Tools is the only DAW that still requires HUI. The rest of world uses Mackie Control Protocol. Ardour does not support HUI at this time.

## 77.2 - SSL NUCLEUS

The Nucleus, from Solid State Logic, is a 16 fader Mackie Control device that includes many buttons, separate meters, two LCD displays and other features. The device is not cheap (around US\$5000 at the time of writing), and has some *design features* (or lack thereof) which some Ardour developers find questionable. Nevertheless, it is a very flexible device, and makes a nice 16 fader surface without the need to somehow attach an extender to your main surface.

### 139.1 Pre-configuring the Nucleus

Your Nucleus comes complete with a number of “profiles” for a few well-known DAWs. At the time of writing it does not include one for Ardour (or related products such as Harrison Mixbus).

We have prepared a profile in which as many buttons as possible send Mackie Control messages, which makes the device maximally useful with Ardour (and Mixbus). You can download [the profile](#) and load it to your Nucleus using the `Edit Profiles` button in SSL’s Nucleus Remote application. Be sure to select it for the active DAW layer in order to make Ardour work as well as possible. *Note: unfortunately, the Nucleus Remote application only runs on OS X or Windows, so Linux users will need access to another system to load the profile. We will provide notes on the profile settings at a future time.*

### 139.2 Connecting the Nucleus

Unlike most Mackie Control devices, the Nucleus uses an ethernet connection to send and receive the MIDI messages that make up the Mackie Control protocol. Specifically, it uses a technology called “ipMIDI” which essentially “broadcasts” MIDI messages on a local area network, so that any connected devices (computers, control surfaces, tablets etc.) can participate.

All other DAWs so far that support the Nucleus have chosen to do so by using a 3rd party MIDI driver called “ipMIDI”, which creates a number of “virtual” MIDI ports on your computer. You, the user, tells the DAW which ports to connect to, and ipMIDI takes care of the rest.

Ardour has builtin ipMIDI support, with no need of any 3rd party packages, and no need to identify the “ports” to connect to in order to communicate with the Nucleus. This makes setting it up a bit easier than most other systems.

Unless ... you already installed the ipMIDI driver in order to use some other DAW with your Nucleus. If ipMIDI is configured to create any “ports”, it is not possible for Ardour’s own ipMIDI support to function. We decided to offer both methods of communicating with your Nucleus. If you regularly use other DAWs, and appreciate having ipMIDI permanently set up to communication with the Nucleus—that’s OK, you can tell Ardour to use the ipMIDI driver you already have. But if you’re not using other DAWs with the Nucleus (and thus have not installed the ipMIDI driver), then you can ignore the ipMIDI driver entirely, and let Ardour connect directly with no configuration.

### 139.2.1 Connecting via Ardour's own ipMIDI support

This is usable only on computers with no 3rd party ipMIDI driver software installed and configured. If you have the OS X or Windows ipMIDI driver from nerds.de, it **MUST** be configured to offer **ZERO** ports before using this method.

Open **Preferences > Control Surfaces**. Ensure that the Mackie protocol is enabled, then double-click on it to open the Mackie Control setup dialog.

Ensure that the device selected is “SSL Nucleus”. The dialog should show a single numerical selector control below it, defining the ipMIDI port number to use (it should almost always be left at the default value of 21928).

Communication is automatically established with the Nucleus and you need do nothing more.

If this does not work, then make sure your network cables are properly connected, and that you are **not** running other ipMIDI software on the computer.

### 139.2.2 Connecting via 3rd party ipMIDI support

This is usable only on computers with 3rd party ipMIDI driver software installed and configured for (at least) 2 ports.

Open **Preferences > Control Surfaces**. Ensure that the Mackie protocol is enabled, then double-click on it to open the Mackie Control setup dialog.

Ensure that the device selected is “SSL Nucleus (via platform MIDI)”. The dialog should show four combo/dropdown selectors, labelled (respectively):

- Main Surface receives via
- Main Surface sends via
- 1st extender receives via
- 1st extender sends via

You should choose “ipMIDI port 1”, “ipMIDI port 1”, “ipMIDI port 2” and “ipMIDI port 2” for each of the 4 combo/dropdown selectors.

Communication should be automatically established with the Nucleus.

If this does not work, then make sure your network cables are properly connected, and that you are running the appropriate ipMIDI driver and have configured it for 2 (or more) ports.

## 139.3 Nucleus Design Discussion

You might be reading this part of the manual seeking some guidance on whether the Nucleus would make a suitable control surface for your workflows. We don’t want to try to answer that question definitively, since the real answer depends on the very specific details of your workflow and situation, but we would like to point out a number of design features of the Nucleus that might change your opinion.

### 139.3.1 Cons

No Master Faster	It is not possible to control the level of the Master bus or Monitor section. Really don't know what SSL was thinking here.
No dedicated rec-enable buttons	You have to press the "Rec" button and convert the per-strip "Select" buttons into rec-enables
No dedicated automation buttons	You have to press the "Auto" button and convert the first 4 vports into 4 automation-related buttons, losing your current view of the session.
No buttons with Mackie-defined "Marker" functionality	Mackie's design intentions for the interoperation of the Marker, rewind and ffwd buttons requires profile editing in order to function properly.
No "Dyn" button	This is hard to assign in an edited profile. To be fair, other Mackie Control devices also lack this button.

### 139.3.2 Pros

Single cable connectivity	No need for multiple MIDI cables to get 16 faders
Broadcast connectivity	Connecting to multiple computers does not require recabling
16 faders from a single box	No need to figure out how to keep extenders together
Meters separated from displays	Contrast with the Mackie Control Universal Pro, where meters interfere with the display
DAW profiles	Easy to flip profiles for use by different DAWs.

### 139.3.3 Ambiguous

Ability to make buttons generate USB keyboard events	The extent to which this is useful reflects the target DAWs inability to manage all of its functionality via Mackie Control
Sophisticated “profile” editing	It is nice to be able to reassign the functionality of most buttons, but this is only necessary because of the relatively few global buttons on the surface.
Builtin analog signal path	SSL clearly expects users to route audio back from their computer via the Nucleus’ own 2 channel output path, and maybe even use the input path as well. They take up a significant amount of surface space with the controls for this signal path, space that could have been used for a master fader or more Mackie Control buttons. The USB audio device requires a proprietary driver, so Linux users can’t use this, and OS X/Windows users will have to install a device driver (very odd for a USB audio device these days). The analog path also no doubt adds notable cost to the Nucleus. There’s nothing wrong with this feature for users that don’t already have a working analog/digital signal path for their computers. But who is going to spend \$5000 on a Nucleus that doesn’t have this already?

## 77.3 - BEHRINGER DEVICES IN MACKIE/LOGIC CONTROL MODE

- *Behringer BCF-2000*
- *Behringer X-Touch*
- *Behringer X-Touch Compact*
- *Behringer X-Touch Mini*

### 140.1 Behringer BCF-2000 Faders Controller

The Behringer BCF-2000 Fader Controller is a control surface with 8 motorized faders, 8 rotary encoders and 30 push buttons. The device is a class compliant USB Midi Interface and also has standard Midi DIN IN/OUT/THRU ports. The device has included a Mackie/Logic Control Emulation Mode since firmware v1.06. Any devices with a firmware older than v1.06 will require an update before Mackie Control Emulation works as described here.

In order to put the controller into Mackie/Logic control mode, the unit must be turned on while holding the third button from the left in the top most row of buttons (under the rotary encoder row). The button must be held down until EG or edit global mode is displayed on the LCD screen of the unit. The global parameters can then be edited using the 8 rotary encoders in the top row.

- Encoder #1 sets the operating mode and should be set to U-1 or USB mode 1 if using with a USB cable connection.
- Encoder #3 sets the foot switch mode and should most likely be set to Auto to detect how the foot switch is wired.
- Encoder #5 sets the device id, if you are using only 1 device the id should be set to ID 1. If you are using multiple BCF/BCR2000 each device is required to be set up sequentially and one at a time.
- Encoder #7 controls the MIDI Dead Time or the amount of milliseconds after a move has been made that the device ignores further changes, this should be set to 100.
- Encoder #8 controls the MIDI message Send Interval in milliseconds and should be set to 10

To exit the EG mode press the Exit button. The device is now ready to use with Ardour.

#### 140.1.1 Modes of Operation

The four buttons arranged in a rectangle and located under the Behringer logo are the mode selection buttons in Logic Control Emulation Mode, currently Ardour has implemented support for two of these modes.

The surface can be broken into 8 groups of controls:

1. The rotary encoders at the top of the device
2. The first row of buttons under the encoders

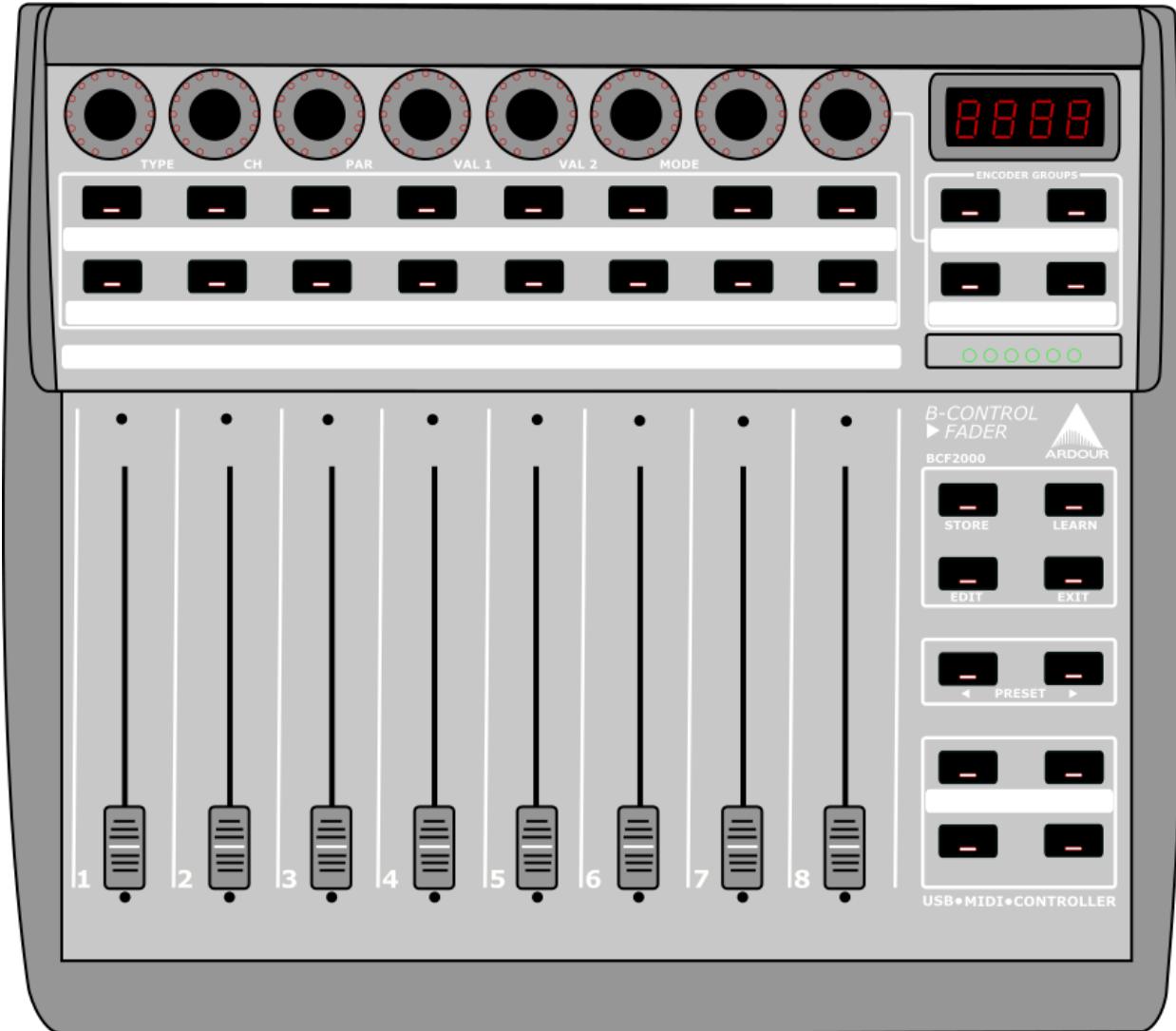


Fig. 1: Diagrammatic Image of the BCF2000 (click for a full-size view)

3. The second row of buttons under the encoders
- #. The row of motorized faders #. #. The group of 4 buttons at the top right that will be referred to here as the Shift Group
  1. The group of 4 buttons under the Shift Group referred to here as the Mode Group
  2. The group of 2 buttons under the Mode Group referred to here as the Select Group
  3. The group of 4 buttons under the Select Group referred to here as the Transport Group

### 140.1.2 Mixer Pan Mode

This is the standard work mode that organizes the control surface to emulate a standard mixer layout where controls for each track/bus are arranged vertically. The order of the faders is either controlled by the order of the tracks in the mixer or can be set manually by the user.

Encoders	Mixer Pans. The red LEDs show the amount of pan left or right
First Row of Buttons	Mixer Mutes. The button led lights up if the track is currently muted
Second Row of Buttons	Select Active Track/Bus. Currently selected track/bus is indicated by the button led
Faders	Mixer Gains
Shift Group	<ul style="list-style-type: none"> <li>The top and bottom left buttons are simply shifts to change the function of other buttons.</li> <li>The top right is the Fine Control button that allows the increment values sent by rotary encoders and faders to be a small value for more precise editing. This button can also act as a shift button.</li> <li>The bottom right is the Global Shift button that allows you to change back to the standard Mixer Pan view from other views and modes. This button can also act as a shift button.</li> </ul>
Mode Group	<ul style="list-style-type: none"> <li>The top two buttons functions are not currently implemented in Ardour.</li> <li>The bottom left button sets the device to Pan mode and should currently be lit</li> <li>The bottom right button sets the device to Send mode but will only allow the switch if the currently selected track/bus has a send or sends to control.</li> </ul>
Select Group	In this mode they function as bank select left and right. If the current session has more than 8 tracks the next set of 8 tracks is selected with the right button and the faders will move to match the current gain settings of that bank of 8 tracks/busses. If the last bank contains less than 8 tracks/busses the unused faders will move to the bottom and the pan lights will all turn off. An unlimited amount of tracks can be controlled with the device.
Transport Group	<ul style="list-style-type: none"> <li>The upper left button controls Rewind.</li> <li>The upper right button controls Fast Forward</li> <li>The lower left button controls Stop</li> <li>The lower right button controls Play</li> </ul>

### 140.1.3 Send Mode

Send mode allows for the top row of encoders to control the sends for a selected channel. One interesting option is to flip the controls from the encoders to the faders by pressing the shift 1 button and the global view button at the same time.

Encoders	In send mode, the encoders control sends from left to right instead of mixer pans. If there are less than 8 sends the behavior of the encoder will be to continue controlling the mixer pan. Visually it is indicated by the change in the LED from originating at the 12 o'clock position to originating at the 7 o'clock position. If FLIP is pressed the encoder will control the mixer gain for the selected track/bus.
First row of buttons	No Change
Second row of buttons	No Change.
Faders	No change unless FLIP is pressed then it controls the send for the selected track/bus.
Shift Group	No Change
Select Group	No Change
Transport Group	No Change

#### 140.1.4 Mixer Pan While Holding Shift 1

The operations of various buttons change while holding down the Shift 1 button:

Encoders	No Change
First row of buttons	These now control the Soloing of each track/bus in the current bank
Second row of buttons	These now control the Enable Record for each track
Faders	No Change
Shift Group	No change
Mode Group	No Change
Select Group	These now change the current bank of tracks being controlled over by one. So if tracks 1-8 were controlled, pushing the right button will change the controlled tracks to 2-9, and pressing the left would then shift back to controlling tracks 1-8.
Transport Group	<ul style="list-style-type: none"> <li>• The upper left now controls turning on and off Loop mode.</li> <li>• The upper right now toggles Click.</li> <li>• The lower left toggles Replace.</li> <li>• The lower right toggles Global Record.</li> </ul>

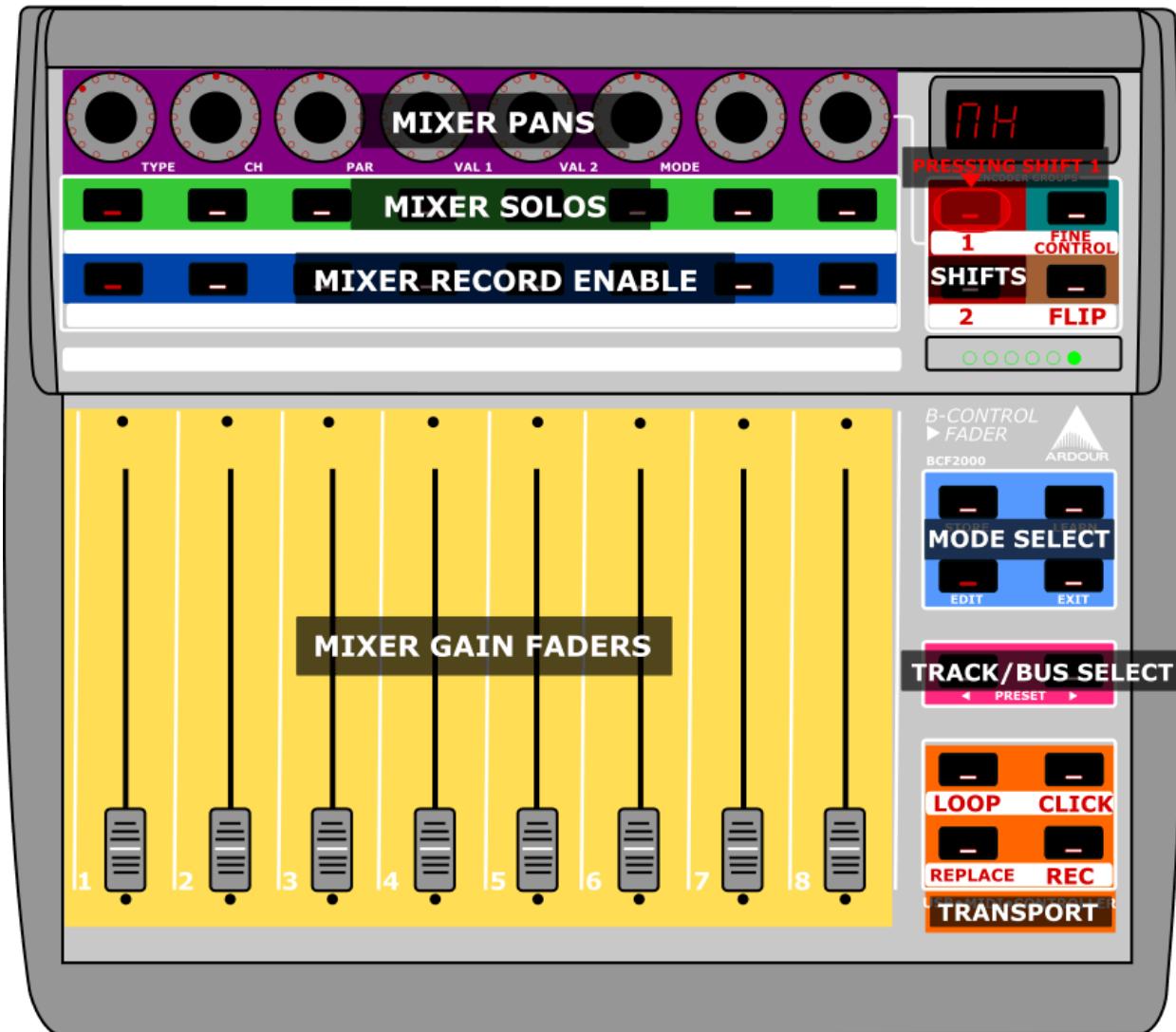


Fig. 2: Diagrammatic Image of the Mixer Mode while holding down Shift 1

### 140.1.5 Mixer Pan While Holding Shift 2

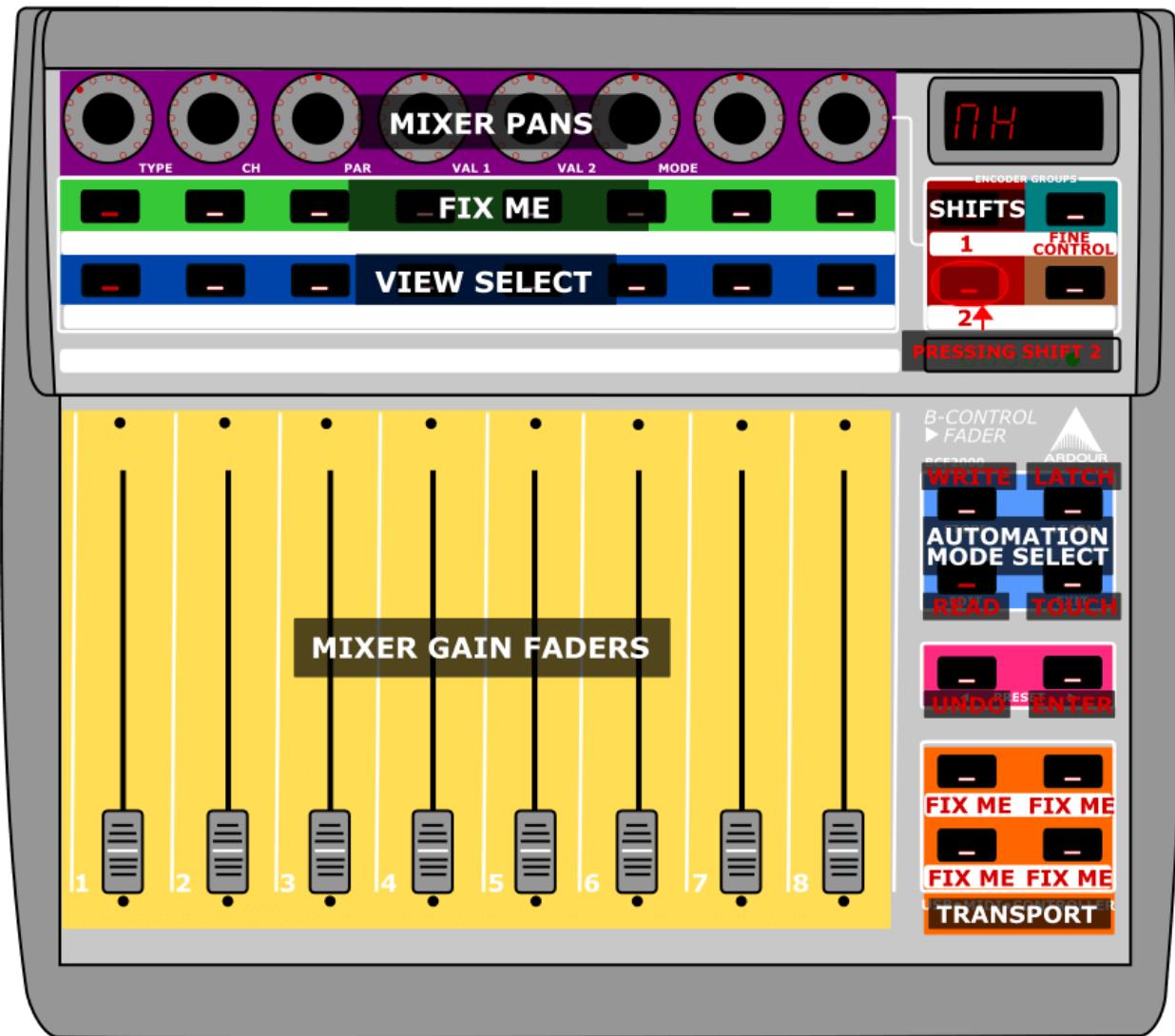


Fig. 3: Diagrammatic Image of the Mixer Mode while holding down Shift 2

The operations of various buttons change while holding down the Shift 2 button:

Encoders	No Change
First row of buttons	FIX ME
Second row of buttons	These now control setting up different Views. See below for more info
Faders	No Change
Shift Group	No change
Mode Group	No Change
Select Group	Left button controls Undo(FIXME: NEEDS VERIFICATION)
Transport Group	FIX ME



Fig. 4: Diagrammatic Image of the LED display for different Views

#### 140.1.6 Views

### 140.2 Behringer X-Touch

The Behringer X-Touch is a direct emulation of the Mackie Control and has all the buttons the Mackie device does. There is a “Behringer X-Touch” map included with Ardour. The X-Touch can be connected to the computer with USB or through a MIDI port. Using USB keeps MIDI ports free for other uses. The Ethernet port uses RTP MIDI which should show up as MIDI devices on Mac OS computers.

### 140.3 Behringer X-Touch Compact

The Behringer X-Touch Compact has a Mackie Control mode. From the device manual:

“To switch between standard operating mode and MC (Mackie Control) mode, press and hold down the MC button in the bottom left corner, and then turn on the unit’s power switch. Keep holding down the MC button until the MC MODE LED lights continuously to show that the unit is in MC mode.”

There is a “Behringer X-Touch Compact” map included with Ardour. The X-Touch can be connected to the computer with USB or through a MIDI port. Using USB keeps MIDI ports free for other uses.

The Behringer X-Touch Compact has fewer controls than the Mackie control and therefore less function as well. See pages 19-21 of the Behringer X-Touch Compact Quick Start Guide For an explanation of what controls on the Compact map to which Mackie Control buttons.

### 140.4 Behringer X-Touch Mini

The Behringer X-Touch Mini says it can emulate Mackie control as well. There is a “Behringer X-Touch Mini” map included with Ardour. The control layout in Mackie Control mode is shown below.

Missing content

## MC MODE

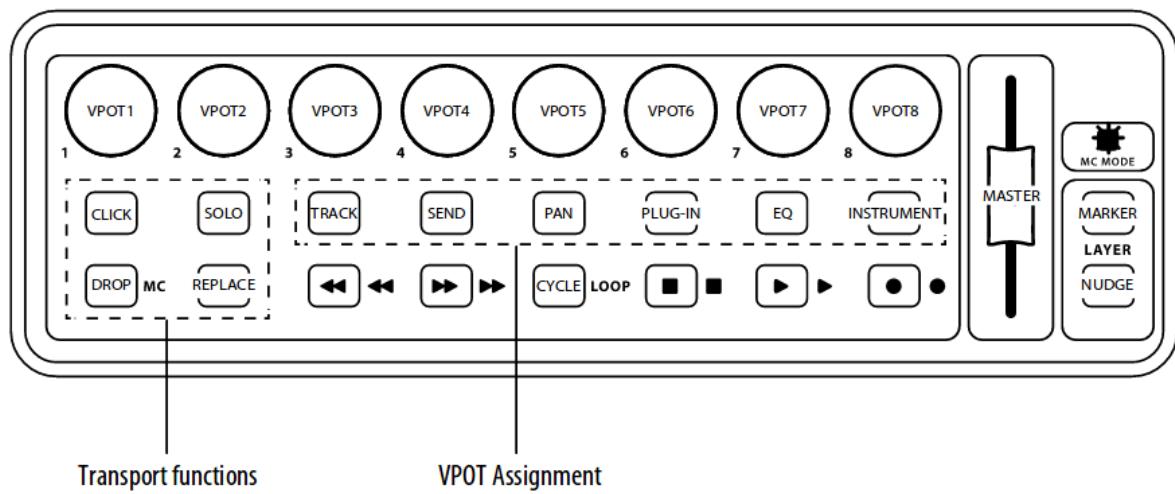


Fig. 5: X-Touch Mini MC mode layout



## **77.4 - WHAT TO DO IF YOUR DEVICE IS NOT LISTED**

All Mackie Control devices are based on the original Logic Control and the documentation in the user manual that came with it. The Mackie Control and the Mackie Control Pro and so on, all use this same protocol. Any units from other manufactures will also use the same encoding as best the hardware will allow. If the unit in use has more than one Mackie Control option, it is best to choose Logic Control or LC. Any Templates for the buttons should be chosen the same way as the Function key Editor uses these button names. The "Mackie Control" option should be considered default and should be tried with any unlisted device before attempting to create a custom definition file.



## 77.5 - WORKING WITH EXTENDERS

There are currently 5 devices pre-configured to work with extenders. Two of them are for one master and one extender with the master on the right side or master on the left side. There are three presets for a master and two extenders with the master on the left, in the center and on the right. While these files will work for many uses there may be cases where a custom device profile makes more sense. The best way is to start with the \*.device file in the [Source Tree](#) that matches your master device and copy it to a new name such as `xt+mc.device` in the [user config](#) sub directory `mcp` and then edit that file. It is best to name the file with the order the devices are expected to be used in as the position of the master device is specified in this file.

The three lines of interest are:

```
<Name value="Device name"/>
<Extenders value="0"/>
<MasterPosition value="0"/>
```

Add any lines that are not present.

The `Name` value should be a unique name so it is obvious in the list of devices (so change it).

The `Extenders` value is the number of extenders used and should not include the master in that number.

When an `Extenders` value of greater than 0 is used, extra midi ports will appear for the extenders to be connected to. The MIDI ports for the controllers will be named `mackie control in/out` for the master, `mackie control in/out ext #*` where \* is the position of the extender from left to right. So for a master in the middle with an extender on either side, the ports from left to right will be `mackie control in/out ext #1`, `mackie control in/out` and `mackie control in/out ext #3`.

If using the MCP GUI to connect surfaces the top surface is the leftmost and the bottom is the rightmost. The GUI shows explicitly the position of the main or master surface within the group of surfaces.

The `MasterPosition` value is the position the master unit (with the master fader) is located at within the group of surfaces. The surfaces are numbered from 1 at the left side and up. So if there are three surfaces, `<MasterPosition value="1"/>` will expect the master on the left, `<MasterPosition value="2"/>` would be master in the middle and `<MasterPosition value="3"/>` would be master on the right.

The default value of `<MasterPosition value="0"/>` has the same effect as `<MasterPosition value="1"/>`.

If the `MasterPosition` value does not properly match the physical position and MIDI port, the master fader and global controls will not work. The master unit will act like an extender.

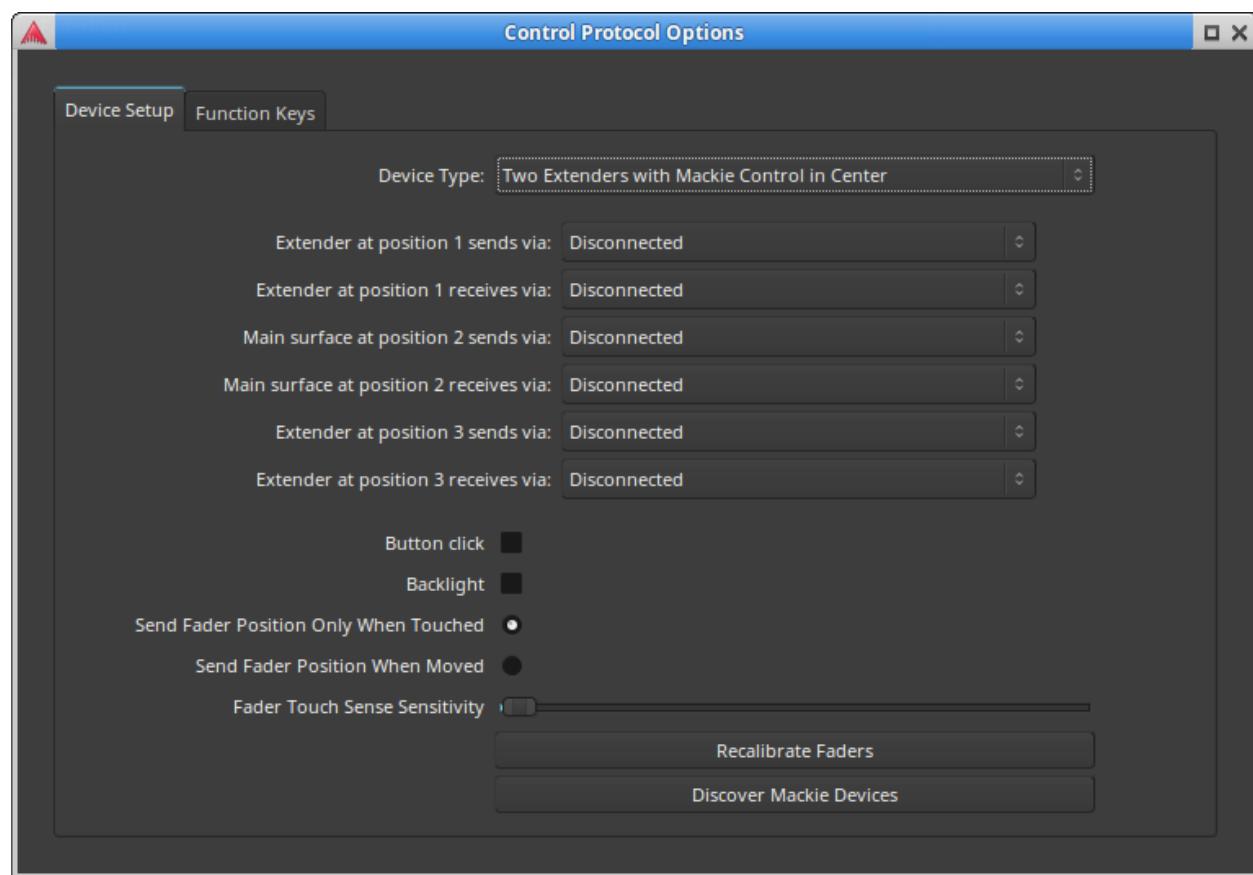


Fig. 1: The Mackie Control Device Dialog

**78 - GENERIC MIDI**

Generic Midi allows synthesizers and other devices communicate with Ardour. MIDI devices can send commands relating to playback (such as play or stop), performance (such as volume, play, stop, and almost any other function (such as Edit, or Undo).

Many MIDI control surfaces use predefined MIDI protocols such as the “Mackie Control Protocol”. In such cases it is best to use Ardour’s implementation of that protocol as it is likely more feature complete.



## 78.1 - GENERIC MIDI BINDING MAPS

Ardour 2.X supported *MIDI learning* for more or less any control. This was a nice feature that quite a few other DAWs are providing by now, but it didn't allow Ardour to work "out of the box" with sensible defaults for existing commercial MIDI controllers. In Ardour 3 and later versions, we have augmented the MIDI learn feature with the ability to load a MIDI binding map for a given controller, which can set up an arbitrary number of physical controls with anything inside Ardour that can be controlled.

Currently (August 2016), we have presets for the following devices/modes:

- AKAI MPD-32
- AKAI MPK61
- AKAI MPKmini
- Behringer BCF2000
- Behringer BCF2000 (Mackie Emulation mode; better to use Ardour's actual Mackie Control Protocol support)
- Behringer DDX3216
- Korg nanoKONTROL (2 layouts)
- Korg nanoKONTROL 2 (2 layouts)
- Korg Taktile
- M-Audio Axiom 25 (2 layouts)
- M-Audio Axiom 61
- M-Audio Oxygen 49
- M-Audio Oxygen 61v3
- M-Audio Oxygen 25
- M-Audio Oxygen 8v2
- Novation Impulse 49
- Novation Impulse 61
- Novation LaunchControl XL
- Novation LaunchKey 25
- Roland SI-24
- Roland V Studio 20
- Yamaha KX25

At this time, new binding maps need to be created with a text editor.

MIDI binding maps are accessible by double-clicking Edit > Preferences > Control Surfaces > Generic MIDI. Ardour will retain your selection after you choose one.

## 144.1 Creating new MIDI maps

### 144.1.1 The Basic Concept

Since the beginning of time (well, sometime early in the 2.X series), Ardour has had the concept of identifying each track and bus with a remote control ID. This ID uniquely identifies a track or bus so that when messages arrive from elsewhere via MIDI or OSC , we can determine which track or bus they are intended to control. See *remote control IDs* for more information. You just need to know that there is a “first track” and its remote control ID is 1, and so on.

### 144.1.2 Getting Started

MIDI bindings are stored in files with the suffix “.map” attached to their name. The minimal content looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<ArdourMIDIBindings version="1.0.0" name="The name of this set of
bindings">
</ArdourMIDIBindings>
```

So, to start, create a file with that as the initial contents.

The file should be located in the midi\_maps sub directory located in the *Ardour configuration directory*

### 144.1.3 Finding out what your MIDI control surface sends

This is the most complex part of the job, but its still not very hard. You need to connect the control surface to an application that will show you the information that the device sends each time you modify a knob, slider, button etc. There are a variety of such applications (notably **gmidimon** and **kmidimon**, but you can actually use Ardour for this if you want. Start Ardour in a terminal window, connect MIDI ports up, and in the Preferences window, enable “Trace Input” on the relevant MIDI port. A full trace of the MIDI data received will show up in the terminal window. (Note: in Ardour3, you get a dedicated, custom dialog for this kind of tracing.)

### 144.1.4 Types of Bindings

There are two basic kinds of bindings you can make between a MIDI message and something inside Ardour. The first is a binding to a specific parameter of a track or bus. The second is a binding to a function that will change Ardour’s state in some way.

#### Binding to Track/Bus controls

A track/bus binding has one of two basic structures

```
<Binding msg specification uri="... control address ..." /> <Binding msg specification
function="... function name ..." />
```

## Message specifications

You can create a binding for either 3 types of channel messages, or for a system exclusive (“sysex”) message. A channel message specification looks like this:

```
<Binding channel="1" ctl="13" ...
```

This defines a binding for a MIDI Continuous Controller message involving controller 13, arriving on channel 1. There are 16 MIDI channels, numbered 1 to 16. Where the example above says `ctl`, you can alternatively use `note` (to create binding for a Note On message) or `pgm` (to create a binding for a Program Change message).

As of Ardour 4.2, `enc-r`, `enc-l`, `enc-2` and `enc-b` may be used for surfaces that have encoders that send offsets rather than values. These accept Continuous Controller messages but treat them as offsets. These are good for banked controls as they are always at the right spot to start adjusting. (*Learn more about working with encoders*)

- <Binding channel="1" enc-r="13" ...
- <Binding channel="1" enc-l="13" ...
- <Binding channel="1" enc-2="13" ...
- <Binding channel="1" enc-b="13" ...

The `enc-*` value is the CC number used by the encoder. Encoders only work with CC messages.

Ardour 5.12 has a bug with the encoder detection where the first encoder message resets the control to 0. Setting “Enable Feedback” on allows encoders to work as expected.

You can also bind sysex messages:

```
<Binding sysex="f0 0 0 e 9 0 5b f7" .... <Binding sysex="f0 7f 0 6 7 f7" ....
```

The string after the `sysex=` part is the sequence of MIDI bytes, as hexadecimal values, that make up the sysex message.

Finally, you can bind a totally arbitrary MIDI message:

```
<Binding msg="f0 0 0 e 9 0 5b f7" .... <Binding msg="80 60 40" ....
```

The string after the `msg=` part is the sequence of MIDI bytes, as hexadecimal values, that make up the message you want to bind. Using this is slightly less efficient than the other variants shown above, but is useful for some oddly designed control devices.

As of Ardour 4.6 it is possible to use multi-event MIDI strings such as two event CC messages, RPN or NRPN.

The `sysex=` and `msg=` bindings will only work with `function=` or `action=` control addresses. They will *not* work with the `uri=` control addresses. Controls used with `uri=` require a *Value* which is only available in a known place with channel mode MIDI events.

## Control address

A control address defines what the binding will actually control. There are quite a few different things that can be specified here:

Enable Feedback applies to these “Control Addresses” only.

/route/gain	the gain control (“fader”) for the track/bus
/route/trim	the trim control for the track/bus (new in 4.1)
/route/solo	a toggleable control for solo (and listen) of the track/bus
/route/mute	a toggleable control to mute/unmute the track/bus
/route/recenable	a toggleable control to record-enable the track
/route/panwidth	interpreted by the track/bus panner, should control image “width”
/route/pandirection	interpreted by the track/bus panner, should control image “direction”
/route/plugin/parameter	the Mth parameter of the Nth plugin of a track/bus
/route/send/gain	the gain control (“fader”) of the Nth send of a track/bus

Each of the specifications needs an address, which takes various forms too. For track-level controls (solo/gain/mute/recenable), the address is one the following:

a number, e.g. “1”	identifies a track or bus by its remote control ID
B, followed by a number	identifies a track or bus by its remote control ID within the current bank (see below for more on banks)
S, followed by a number	identifies a selected track in order they have been selected, S1 should be the same track as the Editor Mixer
one or more words	identifies a track or bus by its name

For send/insert/plugin controls, the address consists of a track/bus address (as just described) followed by a number identifying the plugin/send (starting from 1). For plugin parameters, there is an additional third component: a number identifying the plugin parameter number (starting from 1).

One additional feature: for solo and mute bindings, you can also add `momentary="yes"` after the control address. This is useful primarily for NoteOn bindings—when Ardour gets the NoteOn it will solo or mute the targetted track or bus, but then when a NoteOff arrives, it will un-solo or un-mute it.

### Bindings to Ardour “functions”

There is currently no feedback available for functions.

Rather than binding to a specific track/bus control, it may be useful to have a MIDI controller able to alter some part of Ardour’s state. A binding definition that does this looks like this:

```
<Binding channel="1" note="13" function="transport-roll"/>
```

In this case, a NoteOn message for note number 13 (on channel 1) will start the transport rolling. The following function names are available:

<code>transport-stop</code>	stop the transport
<code>transport-roll</code>	start the transport “rolling”
<code>transport-zero</code>	move the playhead to the zero position
<code>transport-start</code>	move the playhead to the start marker
<code>transport-end</code>	move the playhead to the end marker
<code>loop-toggle</code>	turn on loop playback
<code>rec-enable</code>	enable the global record button
<code>rec-disable</code>	disable the global record button
<code>next-bank</code>	Move track/bus mapping to the next bank (see Banks below)
<code>prev-bank</code>	Move track/bus mapping to the previous bank (see Banks below)

### Binding to Ardour “actions”

It is not possible to have feedback available for actions because these represent keyboard shortcuts which are input only.

You can also bind a sysex or arbitrary message to any of the items that occur in Ardour’s main menu (and its submenus). The [list of actions](#) shows all available values of *action-name*.

To create a binding between an arbitrary MIDI message (we’ll use a note-off on channel 1 of MIDI note 60 (hex) with release velocity 40 (hex)), the binding file would contain:

```
<Binding msg="80 60 40" action="Editor/temporal-zoom-in"/>
```

The general rule, when taken an item from the keybindings file and using it in a MIDI binding is to simply strip the <Action> prefix of the second field in the keybinding definition.

### 144.1.5 Banks and Banking

Because many modern control surfaces offer per-track/bus controls for far fewer tracks & busses than many users want to control, Ardour offers the relatively common place concept of banks. Banks allow you to control any number of tracks and/or busses easily, regardless of how many faders/knobs etc. your control surface has.

To use banking, the control addresses must be specified using the bank relative format mentioned above (“B1” to identify the first track of a bank of tracks, rather than “1” to identify the first track).

One very important extra piece of information is required to use banking: an extra line near the start of the list of bindings that specifies how many tracks/busses to use per bank. If the device has 8 faders, then 8 would be a sensible value to use for this. The line looks like this:

```
<DeviceInfo bank-size="8"/>
```

In addition, you probably want to ensure that you bind something on the control surface to the `next-bank` and `prev-bank` functions, otherwise you and other users will have to use the mouse and the GUI to change banks, which rather defeats the purpose of the bindings.

### 144.1.6 The Selected Strip

Often times one wants to just deal with the strip currently selected by the GUI (or the control surface). In the same way as with banks above the selected strip can be designated with *S1*.

## 144.2 A Complete (though muddled) Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ArdourMIDIBindings version="1.0.0" name="pc1600x transport controls">
  <DeviceInfo bank-size="16"/>
  <Binding channel="1" ctl="1" uri="/route/gain B1"/>
  <Binding channel="1" ctl="2" uri="/route/gain B2"/>
  <Binding channel="1" ctl="3" uri="/route/send/gain B1 1"/>
  <Binding channel="1" ctl="4" uri="/route/plugin/parameter B1 1 1"/>
  <Binding channel="1" ctl="6" uri="/bus/gain master"/>
```

(continues on next page)

(continued from previous page)

```
<Binding channel="1" note="1" uri="/route/solo B1"/>
<Binding channel="1" note="2" uri="/route/solo B2" momentary="yes"/>

<Binding channel="1" note="15" uri="/route/mute B1" momentary="yes"/>
<Binding channel="1" note="16" uri="/route/mute B2" momentary="yes"/>

<Binding channel="1" enc-r="11" uri="/route/pandirection B1"/>
<Binding channel="1" enc-r="12" uri="/route/pandirection B2"/>

<Binding sysex="f0 0 0 e 9 0 5b f7" function="transport-start"/>
<Binding sysex="f0 7f 0 6 7 f7" function="rec-disable"/>
<Binding sysex="f0 7f 0 6 6 f7" function="rec-enable"/>
<Binding sysex="f0 0 0 e 9 0 53 0 0 f7" function="loop-toggle"/>

<Binding channel="1" note="13" function="transport-roll"/>
<Binding channel="1" note="14" function="transport-stop"/>
<Binding channel="1" note="12" function="transport-start"/>
<Binding channel="1" note="11" function="transport-zero"/>
<Binding channel="1" note="10" function="transport-end"/>
</ArdourMIDIBindings>
```

Please note that channel, controller and note numbers are specified as decimal numbers in the ranges 1-16, 0-127 and 0-127 respectively (the channel range may change at some point).

## **78.2 - GENERIC MIDI LEARN**

### **145.1 Philosophy**

There are no “best” ways to map an arbitrary MIDI controller for controlling Ardour. There may be very legitimate reasons for different users to prefer quite different mappings.

On every platform that Ardour runs on, there are excellent free-of-charge tools for making connections between MIDI hardware and “virtual” MIDI ports like the ones that Ardour creates and uses. Rather than waste precious developer time replicating these connection/patch managers, we prefer to leverage their existence by having users rely on them to actually connect Ardour to other MIDI devices and software. On OS X, we recommend Pete Yandell’s MIDI Patchbay. On Linux, a wide variety of tools are available including QJackCtl, aconnect, Patchage, and more.

### **145.2 Basics**

1. Enable Generic MIDI control: Edit > Preferences > Control Surfaces > Generic MIDI
2. Connect Ardour’s MIDI port named control to whatever hardware or software you want (using a MIDI patchbay app)
3. Middle-click on whatever on-screen fader, plugin parameter control, button etc. you want to control
4. A small window appears that says “Operate Controller now”
5. Move the hardware knob or fader, or press the note/key.
6. The binding is complete. Moving the hardware should control the Ardour fader etc.

### **145.3 Cancelling a Learned MIDI Binding**

To unlearn a learned MIDI binding, Middle-click on the control in the same way as you did to learn it in the first place, but click on the popup to cancel it.

### **145.4 Avoiding work in the future**

If you want the bindings you set up to be used automatically in every session, the simplest thing to do is to use Session > Save Template. Then, when creating new sessions, select that template and all the bindings will be automatically set up for you.



## **78.3 - GENERIC MIDI AND ENCODERS**

Encoders are showing up more frequently on controllers. However, they use the same MIDI events as Continuous Controllers and they have no standard way of sending that information as MIDI events. Ardour 4.2 has implemented 4 of the more common ways of sending encoder information.

Encoders that send the same continuous values as a pot would be not discussed here as they are already supported by `ctl`.

Encoders as this page talks about them send direction and offset that the DAW will add to or subtract from the current value.

The 4 kinds of encoder supported are:

- enc-r: On the bcr/bcf2000 this is called “Relative Signed Bit”. The most significant bit sets positive and the lower 6 significant bits are the offset.
- enc-l: The bcr2000 calls this “Relative Signed Bit 2”. The most significant bit sets negative and the lower 6 significant bits are the offset. If you are using one of these two and the values are right but reversed, use the other. This one is the one the Mackie Control Protocol uses.
- enc-2: The bcr2000 calls this one “Relative 2s Complement”. Positive offsets are sent as normal from 1 to 64 and negative offsets are sent as 2s complement negative numbers.
- enc-b: The bcr2000 calls this one “Relative Binary Offset”. Positive offsets are sent as offset plus 64 and negative offsets are sent as 64 minus offset.

If the wrong one is chosen, either the positive or negative side will act incorrectly. It is not really possible to auto detect which one the controller is using. Trial and error is the only way if the specification of the controller is not known.

Many controllers have more than one choice as well, check the manual for the surface.



## 79 - USING THE PRESONUS FADERPORT

Since version 4.5, Ardour has had full support for the Presonus Faderport. This is a compact control surface featuring a single motorized fader, a single knob (encoder) and 24 buttons with fixed labels. It is a relatively low-cost device that functions very well to control a single (selected) track or bus, along with a variety of other “global” settings and conditions.

### 147.1 Connecting the Faderport

The Faderport comes with a single USB socket on the back. Connect a suitable USB cable from there to a USB port on your computer. As of the end of 2015, you should avoid USB3 ports—these cause erratic behaviour with the device. This issue might get fixed by Presonus in the future.

Ardour uses the Faderport in what Presonus calls “native” mode. You do not need to do anything to enable this—Ardour will set the device to be in the correct mode. In native mode, the Faderport sends and receives ordinary MIDI messages to/from the host, and the host understands the intended meaning of these messages. We note this detail to avoid speculation about whether Ardour supports the device via the HUI protocol—it does not.

The Faderport will be automatically recognized by your operating system, and will appear in any of the lists of possible MIDI ports in both Ardour and other similar software.

To connect the Faderport to Ardour, open the Preferences dialog, and then click on “Control Surfaces”. Click on the “Enable” button in the line that says “Faderport” in order to activate Ardour’s Faderport support. Then double click on the line that says “Faderport”. A new dialog will open, containing (among other things) two dropdown selectors that will allow you to identify the MIDI ports where your Faderport is connected.

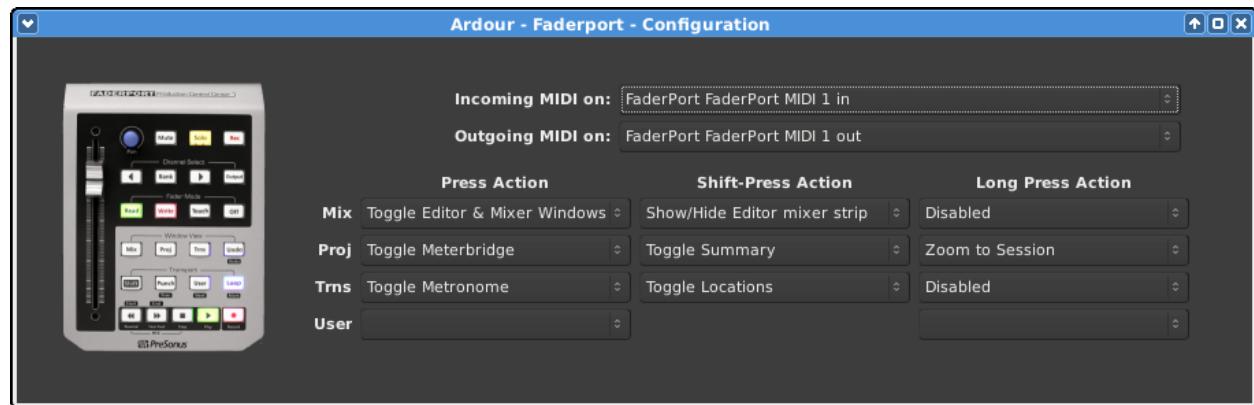


Fig. 1: The Faderport configuration dialog

Once you select the input and output port, Ardour will initialize the Faderport and it will be ready to use. You only need do this once: once these ports are connected and your session has been saved, the connections will be made automatically in this and other future sessions.

You do not need to use the power supply that comes with the Faderport but without it, the fader will not be motorized. This makes the overall experience of using the Faderport much less satisfactory, since the fader will not move when Ardour tells it to, leading to very out-of-sync conditions between the physical fader position and the “fader position” inside the program.

## 147.2 Using the Faderport

The Faderport’s controls can be divided into three groups:

1. Global controls such as the transport buttons
2. Controls which change the settings for particular track or bus
3. Controls which alter which track or bus is modified by the per-track/bus controls.

Because the Faderport has only a single set of per-track controls, by default those controls operate on the first selected track or bus. If there is no selected track or bus, the controls will do nothing.

### 147.2.1 Transport Buttons

The transport buttons all work as you would expect.

Rewind	When pressed on its own, starts the transport moving backwards. Successive presses speed up the “rewind” behaviour. If pressed while also holding the Stop button, the playhead will return to the zero position on the timeline. If pressed while also holding the Shift button, the playhead will move to the session start marker.
Fast Forward	When pressed on its own, starts the transport moving faster than normal. Successive presses speed up the “fast forward” behaviour. If pressed while also holding the Shift button, the playhead will move to the session end marker.
Stop	Stops the transport. Also used in combination with the Rewind button to “return to zero”.
Play	Starts the transport. If pressed while the transport is already rolling at normal speed, causes the playhead to jump to the start of the last “roll” and continue rolling (“Poor man’s looping”).
Record Enable	Toggles the global record enable setting

### 147.2.2 Other Global Controls

The Mix, Proj, Trns buttons do not obviously correspond any particular functions or operations in Ardour. We have therefore allowed users to choose from a carefully curated set of possible actions that seem related to the button labels in some clear way. This can be done via the Faderport configuration dialog accessed via **Preferences > Control Surfaces**. Each button has 3 possible actions associated with it:

- Plain Press: action to be taken when the button is pressed on its own.
- Shift-Press: action to be taken when the button is pressed in conjunction with the Shift button.
- Long Press: action to be taken when the button is pressed on its own and held down for more than 0.5 seconds.

Click on the relevant drop-down selector to pick an action as you prefer.

The User button also has no obvious mapping to specific Ardour functionality, so we allow users to choose from *any* possible GUI action. The menu for selecting the action is somewhat confusing and it can be hard to find what you're looking for. However, all possible actions are there, so keep looking!

Mix	Possible actions include: <ul style="list-style-type: none"> <li>• Toggle Editor &amp; Mixer visibility</li> <li>• Show/Hide the Editor mixer strip</li> </ul>
Proj	Possible actions include: <ul style="list-style-type: none"> <li>• Toggle Meterbridge visibility</li> <li>• Toggle Session Summary visibility</li> <li>• Toggle Editor Lists visibility</li> <li>• Zoom to session</li> <li>• Zoom in</li> <li>• Zoom out</li> </ul>
Trns	Possible actions include: <ul style="list-style-type: none"> <li>• Toggle Locations window visibility</li> <li>• Toggle Metronome</li> <li>• Toggle external sync</li> <li>• Set Playhead at current pointer position</li> </ul>
Undo/Redo	Undo Causes the last operation carried out in the editor to be undone. When pressed in conjunction with the Shift button, it causes the most recent undone operation to be re-done.
Punch	When pressed on its own, toggles punch recording. If there is no punch range set for the session, this will do nothing. When pressed in conjunction with the Shift button, this moves the playhead to the previous Marker
User	See above. Any and all GUI-initiated actions can be driven with by pressing this button on its own, or with a “long” press. When pressed in conjunction with the Shift button, this will move the playhead to the next marker.
Loop	When pressed on its own, this toggles loop playback. If the Ardour preference “Loop-is-mode” is enabled, this does nothing to the current transport state. If that preference is disabled, then engaging loop playback will also start the transport. When pressed in conjunction with the Shift button, this will create a new (unnamed) marker at the current playhead position.

### 147.2.3 Per-track Controls

Mute	This toggles the mute setting of the currently controlled track/bus. The button will be lit if the track/bus is muted.
Solo	This toggles the solo (or listen) setting of the currently controlled track/bus. The button will be lit if the track/bus is soloed (or set to listen mode).
Rec	This toggles the record-enabled setting of the currently controlled track/bus. The button will be lit if the track is record-enabled. This button will do nothing if the Faderport is controlling a bus.
Fader	The fader controls the gain applied to the currently controlled track/bus. If the Faderport is powered, changing the gain in Ardour's GUI or via another control surface, or via automation, will result in the fader moving under its own control.
Knob	<p><b>Dial</b> <small>Encoder</small> controls 1 or 2 pan settings for the current controlled track/bus. When used alone, turning the knob controls the “azimuth” or “direction” (between left and right) for the panner in the track/bus (if any). This is all you need when controlling tracks/busses with 1 input and 2 outputs.</p> <p>If controlling a 2 input/2 output track/bus, Ardour’s panner has two controls: azimuth (direction) and width. The width must be reduced to less than 100% before the azimuth can be changed. Pressing the “Shift” button while turning the knob will alter the width setting.</p> <p>The knob can also be turned while the “User” button is held, in order to modify the input gain for the currently controlled track.</p>
Read	Enables playback/use of fader automation data by the controlled track/bus.
Write	Puts the fader for the controlled track/bus into automation write mode. While the transport is rolling, all fader changes will be recorded to the fader automation lane for the relevant track/bus.
Touch	Puts the fader for the controlled track/bus into automation touch mode. While the transport is rolling, touching the fader will initiate recording all fader changes until the fader is released. When the fader is not being touched, existing automation data will be played/used to control the gain level.
Off	This disables all automation modes for the currently controlled track/bus. Existing automation data will be left unmodified by any fader changes, and will not be used for controlling gain.

### 147.2.4 Track Selection Controls

You can manually change the track/bus controlled by the Faderport by changing the selected track in Ardour's editor window. If you select more than 1 track, the Faderport will control the first selected track and *only* that track/bus.

Left (ar- row)	This causes the Ardour GUI to select the previous track/bus (using the current visual order in the editor window), which will in turn cause the Faderport to control that track. If there is no previous track/bus, the selected track/bus is left unchanged, and the Faderport continues to control it.
Right (ar- row)	This causes the Ardour GUI to select the next track/bus (using the current visual order in the editor window), which will in turn cause the Faderport to control that track. If there is no next track/bus, the selected track/bus is left unchanged, and the Faderport continues to control it.
Out- put	<p>Pressing the Output button causes the Faderport to control the fader, pan, mute and solo settings of the Master bus. If your session does not contain a Master bus, it does nothing. This is a toggle button—pressing it again returns Faderport to controlling whichever track/bus was selected before the first press of the Output button.</p> <p>If your session uses Ardour's monitor section, you can use Shift-Output to assign it to the Faderport in the same way that Output assigns the Master bus. This is also a toggle setting, so the second Shift-Output will return the Faderport to controlling whichever track/bus was selected before.</p> <p>If you press Shift-Output after a single press to Output (i.e. control the Monitor Section while currently controlling the Master bus) or vice versa (i.e. control the Master bus while currently controlling the Monitor Section), the press will be ignored. This avoids getting into a tricky situation where it is no longer apparent what is being controlled and what will happen if you try to change it.</p>
Bank	The “Bank” button is currently not used by Ardour



## **80 - USING THE PRESONUS FADERPORT 8**

Since version 5.8-290, Ardour supports for the Presonus FaderPort™ 8.

The FaderPort™ 8 is a production control surface with 8 touch-sensitive, motorized faders, monochromatic digital scribble strips and more than 60 buttons with fixed labels.

### **148.1 Connecting the FaderPort 8**

The FaderPort 8 (FP8) comes with a USB socket on the back. Connect a suitable USB cable from there to a USB port on your computer. The FP8 will be automatically recognized by your operating system, and will appear in any of the lists of possible MIDI ports in both Ardour and other similar software.

Ardour uses the FaderPort 8 in what PreSonus calls “Studio One” or “native” mode. To use the FaderPort8 with Ardour’s FP8 Control Surface, make sure that the device is in “Studio One” mode. (If you would like to change the mode at any point, power on the unit while holding down the two leftmost Select buttons, see the FaderPort 8 manual for further details. Also note that at least firmware version 1.01 is required.

NB. “factory default” resets the firmware, see the PreSonus FaderPort8 Owner’s manual chapter 9.4.)

While the FaderPort provides a Mackie Control Universal (MCU) mode, which works with Ardour’s Mackie Control Surface, MCU does not support various elements available on the FP8 (e.g. colored buttons, and the custom mode scribble strips).

To connect the FP8 to Ardour, open the Preferences dialog, select “Control Surfaces” and enable “PreSonus FaderPort 8”. Then open the “Protocol Settings” dialog for the FP8. Which (among other things) allows to select the the MIDI ports corresponding to the FP8.

Once you select the input and output port, Ardour will initialize the FP8 and it will be ready to use. You only need do this once: Once these ports are connected and your session has been saved, the connections will be made automatically in this and other future sessions.

### **148.2 Using the FaderPort 8**

The FaderPort’s controls can be divided into five groups:

- Transport buttons
- Session Navigation controls
- Fader modes



Fig. 1: FaderPort8 Control Surface Settings Dialog

- Mix management
- Channel strip

In general the control mapping described in the FaderPort 8 Owner's Manual for Studio One (chapter 2) applies to Ardour as well. There are however subtle differences where the DAWs differ.

Buttons generally act on release (not press), with exception of transport-control (since 6.0pre) and individual exceptions mentioned below.

### 148.2.1 Transport Buttons

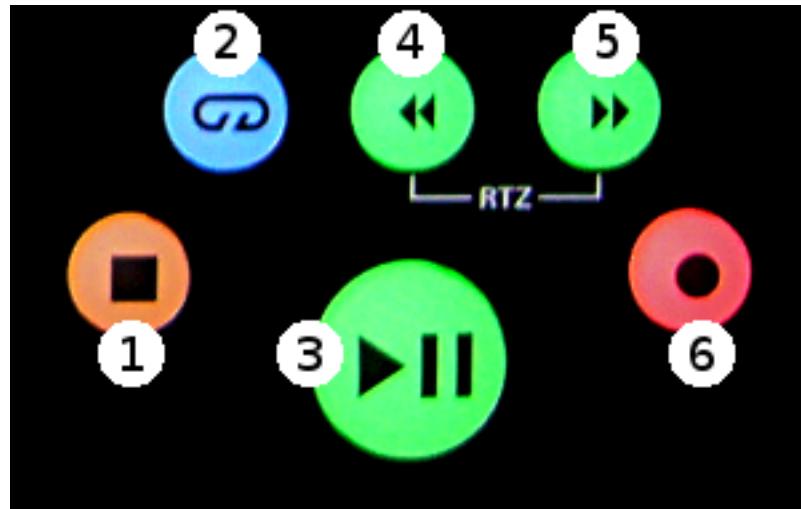


Fig. 2: FaderPort8 Transport Buttons

1. Stop: Stops the transport. Press twice to return to session start.
2. Loop: Toggles loop playback. A loop-range needs to be defined in the session for looping to be engaged.
3. Play/Pause: Roll/Stop the transport (note that Ardour has no “pause” mode: Pause is equivalent to stop). During vari-speed playback, pressing “play” resets to 100% forward speed.

4. Rewind: Rewind, roll backwards. Successive presses or holding the button incrementally changes the speed.
5. Fast Forward: Roll forward. Successive presses or holding the button accumulate speed. Pressing (Rewind and Fast Forward) simultaneously stops playback and returns the playhead to zero.
6. Record: Toggles the global record enable setting.

### 148.2.2 Session Navigation

Session Navigation allows quick navigation and provides access to session-wide controls. Each of the eight buttons alters the function of the push-button encoder and the Prev, Next buttons. With exception of Click the seven modes are exclusive (radio buttons).



Fig. 3: FaderPort8 Navigation Buttons

1. Channel: The Prev / Next buttons select the previous/next mixer-strip. If no strip is selected, Next selects the first, Prev the last mixer-strip in the session. Pressing the encoder knob moves the most recently selected mixer-strip into view on the FP8. The encoder scrolls the editor-canva up/down.
2. Master: The encoder controls the master-bus level. If a session includes a monitor-section, the encoder controls monitor-out by default. Hold the button to control the master-bus level. Press the encoder knob to reset the gain to 0dB. The Prev / Next navigation buttons bank the visible strips on the FP8 by one track left/right.
3. Zoom: The encoder controls horizontal zoom of the editor. Press the encoder to *zoom to fit* the session. Prev / Next navigation buttons zoom selected track(s) vertically (or all tracks if none are selected).
4. Click: Toggle the metronome on/off. While holding the Click button, the encoder modifies the volume of the metronome click (press the encoder while holding Click to reset the metronome level to 0dBFS).

5. Scroll: The encoder scrolls the timeline (hold Shift for finer steps). Pressing the encoder *zooms to fit* the session. The Prev / Next navigation buttons bank the visible strips on the FP8 by one track left/right.
6. Section: The Prev / Next navigation buttons nudge the selected region by the time configured in the nudge-clock. If no region is selected the playhead position is nudged. The encoder always nudges the playhead position.
7. Bank: Encoder and navigation buttons scroll through mixer-strips in banks of eight. Pressing the encoder moves the most recently selected mixer-strip into view on the FP8.
8. Marker: The encoder scrolls the timeline (hold Shift for finer steps). The Prev / Next navigation buttons jump to prev/next markers. Press the encoder to drop a new marker.

When combined with Shift, the eight buttons will access custom functions, which can be configured in the Preference Dialog. The buttons will light up if an action has been assigned to a button.

The following tables shows a condensed overview of the session-navigation modes:

	Prev / Next	Encoder knob	Encoder Press
Channel	Select prev/next mixer-strip	Scroll Editor up/down	Bank to show selected strip on FP8
Master	Bank visible strips on FP8 by 1	Adjust master/monitor level	Reset master/monitor to 0dB
Zoom	Vertical zoom (editor track-height)	Horizontal timeline zoom (time)	Horizontal zoom to session
Scroll	Bank visible strips on FP8 by 1	Scroll the timeline (move playhead)	Horizontal zoom to session
Section	Nudge the selected region	Nudge the playhead	•
Bank	Bank visible strips on FP8 by 8	Bank visible strips on FP8 by 1	Bank to show selected strip on FP8
Marker	Move to prev/next marker	Scroll the timeline (move playhead)	Drop a new marker
Press and hold Click	(mode dependent)	Adjust metronome Level	Reset metronome level to 0dBFS

### 148.2.3 Shift Button



Fig. 4: The FaderPort8 Shift Button

The two Shift buttons are identical, they're copied to provide convenient access to the modifiers. Pressing

and holding the Shift button updates the lights (and colors on RGB buttons) to indicate the modified control.

Pressing and holding the Shift button for one second without pressing any other button enters shift-lock mode. Press Shift again to reset. The Shift button engages directly on press. Activating an action while the button is held will void the shift-lock mode.

#### 148.2.4 Fader Modes

The eight faders on the FP8 can be assigned to various automatable controls present in the current session. The four fader-mode buttons change the behavior of the mixer-strip and scribble strip displays. (Note: with the 1.01 firmware these buttons always act on press.)

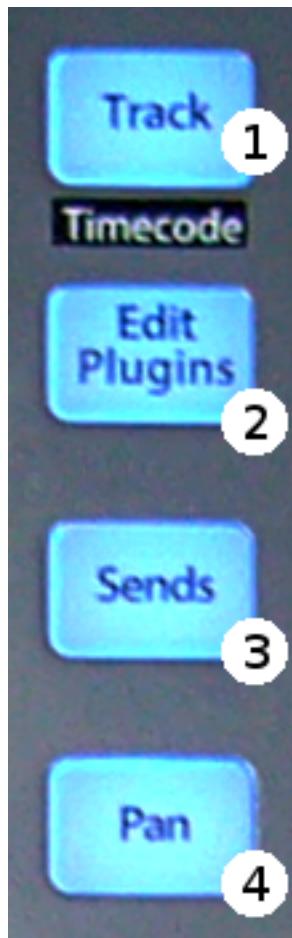


Fig. 5: FaderPort8 Fader Mode Buttons

1. Track: In Track-mode, the motorized faders display and control a mixer-strip's signal level. The Pan/Param encoder modifies the azimuth of the panner (hold Shift to control the width, if the track's panner supports it). Mute and Solo affect the respective mixer-strip.
2. Edit Plugins: When Edit Plug-ins mode is active, the motorized faders will control the parameter settings of a given plugin-insert. Press the Edit Plugins button to view all available plugin-inserts on a strip. If no plugins are available, Edit Plugins will not engage and the FP8 automatically switches back to Track-mode. **Select Plugin Mode:** Use the Select buttons under the scribble strip to pick a plugin to edit. The Select button color indicates the bypass/enable state of the plugin (red: bypassed,

green: enabled). Use Shift + Select to toggle the bypass state. Selecting a plugin enters **Parameter Edit Mode**: The faders and the Select buttons will respectively control the parameters and toggle controls of the selected plugin (once a plugin has been selected, it stays in edit mode regardless of track selection). If there are more than eight parameters, the Pan/Param encoder allows to scroll through available control-parameters (hold Shift to bank by 8). If the plugin has any presets, pressing the Pan/Param encoder switches to the **plugin-preset display**: Plugin preset names are displayed on the scribble-displays, the Select button below each loads the preset. The “Pan/Param” encoder can be used to scroll through presets if there are more than seven (right-most, 8th, slot is reserved to unload/clear a loaded preset, hold Shift to bank by 7). The Select button color is used to indicate the currently loaded preset (if any) and blinks if a parameter has been modified since loading the preset. Loading a preset or pressing the Pan/Param encoder again switches back to the Plugin Parameter Edit Mode. In Parameter Edit Mode, the “Open” (Shift + Macro) allows to toggle the Plugin GUI visibility. Press the Edit Plugins button again to return to the Select Plugin Mode.

3. Sends: In Sends mode, each of the faders is mapped to the send-level of aux-sends of the selected track. If there are more than eight sends on a given track, the Pan/Param encoder can scroll through them. Send-mode follow the selection. If there are no sends on a given track, the FP8 automatically switches back to Track-mode.
4. Pan: When Pan mode is active, the motorized faders will display and control the panner’s azimuth. The Pan/Param controls the pan-width of the selected mixer-strip.

Shift + Track toggles timecode display on/off (middle row of the scribble-strip). The timecode format can be configured in the Control Surface Preference Dialog (Timecode, musical-time: bar/beat/tick).

#### 148.2.5 Channel Strip

- **Touch-Sensitive Fader:** The fader can be used to control volume levels, aux send levels, panning, or plugin parameters, depending on the fader-mode (see above).
- **Pan/Param:** The encoder controls panning in Track and Pan mode. In Plugin and Send fader-modes, the encoder banks parameters. See Fader modes above for details. When “Link” is engaged, the encoder can control any automatable parameter (see Miscellaneous below).
- Mute: Toggle the mute-control of the corresponding mixer-strip. Mute engages on press, and disengages on release. Press and hold the button for at least 0.5sec for momentary.
- Solo: Toggle the solo or listen (AFL,PFL) control of the corresponding mixer-strip. Solo engages on press, and disengages on release. Press and hold the button for at least 500ms for momentary.
- Select: In Track and Sends and Pan mode the Select button select/de-select a given mixer-strip. Since selection is not limited to a single mixer-strip, the button acts in tri-state. A mixer-strip light indicates selection:
  - **Any Selected Track:** The select button is lit with the track’s color.
  - **Any Not Selected Track:** The select button is off (dimly showing the track’s color).
  - **Most Recently Selected Track:** Only one track at a time. The select button blinks with the track’s color.

Operations such as Edit Plugins or Sends use the most-recently-selected (focused) track. To modify the selection, the button’s action depends on the current selection:

1. **Select:** The track is exclusively selected and also becomes the most-recently selected.
2. **Shift + Select any selected track:** Deselect the track.
3. **Shift + Select any unselected track:** Adds the given track to the selection and make it most-recently selected).



Fig. 6: The FaderPort8 Channel Strip

In Track-mode, pressing the Select button of the most recently selected track (blinking Select button) will reset the fader-gain to unity (0dB). (since Ardour 5.11-207, Mixbus 4.2-66)

While holding the ARM button the Select button lights change to red and the Select buttons controls the record-arm of the given track. Mixer-strips that cannot be record-armed have a dim white light.

Shift + ARM record-arms all tracks in the session.

### 148.2.6 Mix Management

These buttons allows to select which mixer-strips are spilled on the FP8 channel-strips.



Fig. 7: FaderPort8 Mix Management Buttons

1. Audio: View Audio Tracks only.
2. VI: Show tracks with virtual instrument plugins.

3. Bus: Display only Busses.
4. VCA: Show VCAs.
5. All: Display all Tracks, Busses (incl master-bus) and VCAs.

In combination with the Shift modifier ten total filters are available:

- Shift + Audio **Inputs**: shows all record-armed tracks (Audio and MIDI).
- Shift + VI **MIDI**: View all MIDI tracks.
- Shift + Bus **Outputs**: Show the Master and Monitor Bus.
- Shift + VCA **FX**: Shows Aux-Busess.
- Shift + All **User**: Display all currently selected mixer-strips only.

### 148.2.7 Automation Controls

The Automation Controls provide access to the currently selected mixer-strips. The automation enable lights indicates the mode of the most recently selected mixer-strip (blinking selection button). The action affects all selected mixer-strips. The automation controls are currently only available in Track and Pan fader modes where they affect the fader and pan automation modes respectively.

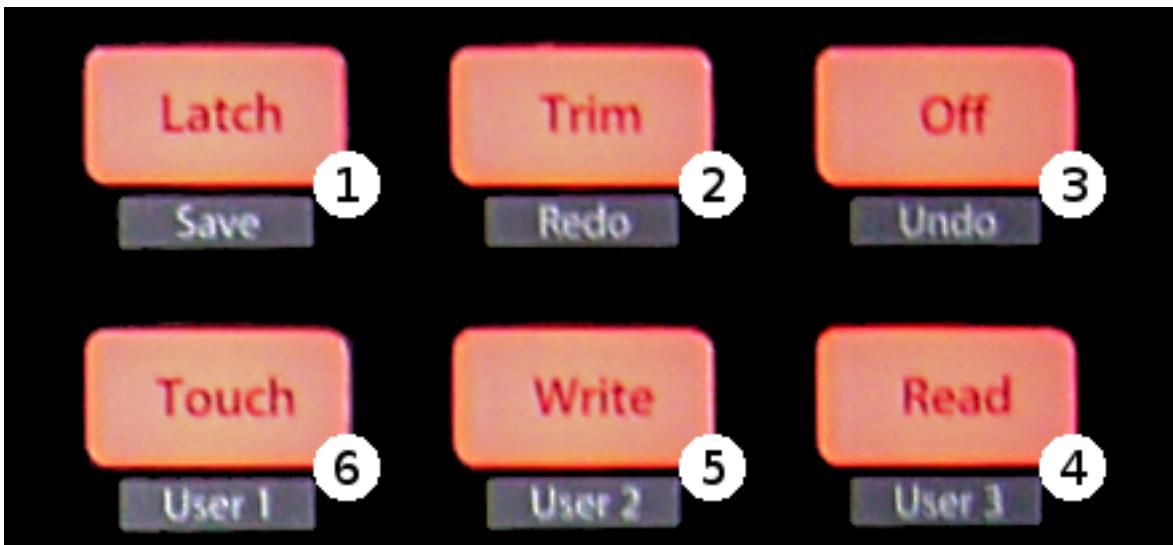


Fig. 8: FaderPort8 Automation Buttons

1. Latch: Currently not available in Ardour.
2. Trim: Currently not available in Ardour.
3. Off: Select “Manual” automation mode.
4. Read: Select “Play” automation mode.
5. Write: Select “Write” automation mode (note at the end of a write pass, Ardour automatically puts the track into “Touch” mode).
6. Touch Select “Touch” automation mode.

The Automation Controls also double as session state controls when combined with Shift.

1. Shift + Latch **Save**: Save the session. The button lights up red if the session is modified.

2. Shift + Trim **Redo**: Redo a previously undone operation. The button lights up green if redo is possible.
  3. Shift + Off **Undo**: Undo the most recent operation. The button lights up green if undo is possible.
- With Shift, the bottom row allows to bind three custom user actions.

#### 148.2.8 Miscellaneous

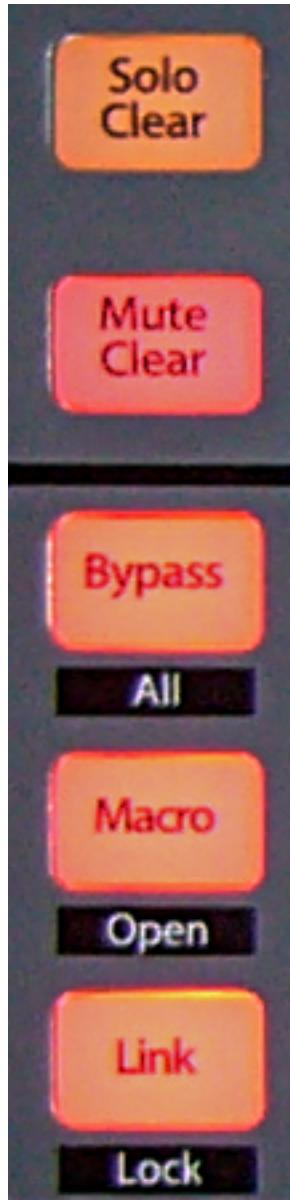


Fig. 9: FaderPort8 Misc Buttons

- Solo Clear: Reset all solo controls in the session. If the FP8 was used to clear solo-state, pressing the button again will restore the previous state (unless solo state was modified manually since).
- Mute Clear: Unmute all mixer-strips in the session. If the FP8 was used to clear mute-state, pressing the button again will restore the previous mute state (unless mute-state was changed manually since).

- Bypass: The behavior depends on the edit-mode:
  - **Track + Pan Mode:** A/B bypass toggle any plugins on all selected mixer-strips.
  - **Edit Plugin Parameter:** Toggle bypass of the plugin that is currently being edited. Bypass state is indicated by color: red for bypassed, green for enabled (not bypassed).
- Shift + Bypass **Bypass All:** A/B bypass toggle any plugins on all selected mixer-strips.
- Macro: Toggle Editor and Mixer Windows/Tabs.
- Shift + Macro **Open:** The behavior depends on the edit-mode:
  - **Edit Plugin Parameter:** Toggle Plugin GUI visibility (if it has a GUI) of the plugin that is currently being edited.
  - **all other modes:** Show the Import Audio Dialog.
- Link: Activate Control-Link Mode (only available in Track and Pan modes). The Pan/Param encoder controls the element over which the mouse-cursor hovers in the GUI. One can access any parameter which can be automated. Pressing the Pan/Param encoder resets the control-parameter to the default value. The buttons color is used to indicate the link-state:
  - **orange:** Link is enabled, but the mouse-cursor is not over an element which can be controlled.
  - **yellow:** Link is enabled, and the cursor is hovering over a controllable element.
  - **green:** Link is locked to a given element (see below).
  - **turquoise:** Link lock is possible (when pressing Shift while link-mode is enabled).
  - **red:** Link-lock is not possible (only when pressing Shift while link-mode is enabled without a valid element to control).
- Shift + Link **Lock:** When in Link-mode (see above), this allows to lock the current control to the Pan/Param encoder. Link will no longer follow the GUI mouse-cursor. If Link-mode is not enabled, Lock, locks the GUI (alike Session > Lock) to prevent accidental changes.

Link and Link-Lock mode will automatically disengage when entering Sends or Edit Plugins mode.

### 148.2.9 Harrison Mixbus

The above also applies to Ardour-derivatives Harrison-Mixbus and Mixbus 32C with a few subtle differences:

- Mix Management **Bus** shows Mixbusses only, while **FX** spills Aux-busses.
- The Mixbus built-in EQ and Compressor are present on every track and bus and always available. They are displayed as special plugins on right-side in **Select Plugin Mode**. When editing those processors, the parameters follows track selection (for other plugins this is not possible since they may not be present).
- Fader mode **Sends** shows mixbus-assigns first (before any optional aux-sends). The master-bus-assign is available on the “S”olo button of the right-most strip.



## 81 - USING THE ABLETON PUSH 2



Fig. 1: The Ableton Push 2 surface

Since version 5.4, Ardour has had extensive support for the Ableton Push2. This is an expensive but beautifully engineered control surface primarily targeting the workflow found in Ableton's Live software and other similar tools such as Bitwig. As of version 5.4, Ardour does not offer the same kind of workflow, so we have designed our support for the Push 2 around mixing and editing and musical performance, without the clip/scene oriented approach in Live. This may change in future versions of Ardour.

## 149.1 Connecting the Push 2

Plug the USB cable from the Push 2 into a USB2 or USB3 port on your computer. For brighter backlighting, also plug in the power supply (this is not necessary for use).

The Push 2 will be automatically recognized by your operating system, and will appear in any of the lists of possible MIDI ports in both Ardour and other similar software.

To connect the Push 2 to Ardour, open the Preferences dialog, and then click on “Control Surfaces”. Click on the “Enable” button in the line that says “Ableton Push 2” in order to activate Ardour’s Push 2 support.

Once you select the input and output port, Ardour will initialize the Push 2 and it will be ready to use. You only need do this once: once these ports are connected and your session has been saved, the connections will be made automatically in this and other future sessions.

## 149.2 Push 2 Configuration

The only configuration option at this time is whether the pads send aftertouch or polyphonic pressure messages. You can alter this setting via the Push 2 GUI, accessed by double-clicking on the “Push 2” entry in the control surfaces list.



Fig. 2: The Push 2 configuration dialog

## 149.3 Basic Concepts

With the Push 2 support in Ardour 5.4, you can do the following things:

Perform using the 8 x 8 pad “grid”	The Push 2 has really lovely pressure-sensitive pads that can also generate either aftertouch or note (polyphonic) pressure.
Global Mixing	See many tracks at once, and control numerous parameters for each.
Track/Bus Mixing	View a single track/bus, with even more parameters for the track.
Choose the mode/scale, root note and more for the pads	37 scales are available. Like Live, Ardour offers both “in-key” and “chromatic” pad layouts.

... plus a variety of tasks related to transport control, selection, import, click track control and more.

## 149.4 Musical Performance

Messages sent from the 8x8 pad grid and the “pitch bend bar” are routed to a special MIDI port within Ardour called “Push 2 Pads” (no extra latency is incurred from this routing). Although you can manually connect this port to whatever you wish, the normal behaviour of Ardour’s Push 2 support is to connect the pads to the most recently selected MIDI track.

This means that to play a soft-synth/instrument plugin in a given MIDI track with the Push 2, you just need to select that track.

If multiple MIDI tracks are selected at once, the first selected track will be used. Note that messages originating from all other controls on the Push 2 will *not* be delivered to the “Push 2 Pads” port. This makes no difference in practice, because the other controls do not send messages that are useful for musical performance.

## 149.5 Global Mix

This is the default mode that Ardour will start the Push 2 in. In this mode, the 8 knobs at the top of the device, the 8 buttons below them, the video display and the 8 buttons below that are combined to provide a global view of the session mix.



Fig. 3: Global mix mode on Push2 screen

The upper buttons are labelled by text in the video display just below them. Pressing one of the buttons changes the function of the knobs, and the parameters that will shown for each track/bus in the display.

As of Ardour 5.4, the possible parameters are:

Volumes	The display shows a knob and text displaying the current gain setting for the track, and a meter that corresponds precisely to the meter shown in the Ardour GUI for that track. Changing the meter type (e.g. from Peak to K12) in the GUI will also change it in the Push 2 display. The physical knob will alter track/bus gain.
Pans	The display shows a knob indicating the pan direction/azimuth for the corresponding track/bus. Turning the physical knob will pan the track left and right. If the track/bus has no panner (i.e. it has only a single output), no knob is shown and the physical knob will do nothing.
Pan Width	For tracks with 2 outputs, the display will show a knob indicating the pan width setting for the corresponding track/bus. The physical knob can be turned to adjust the width. Unlike many DAWs, Ardour's stereo panners have "width" parameter that defaults to 100%. You cannot change the pan direction/azimuth of a track with 100% width, but must first reduce the width in order to pan it. Similarly, a track panned anywhere other than dead center has limits on the maximum width setting. If these concepts are not familiar to you, please be aware than many DAWs use a "panner" that actually implement "balance" and not "panning", hence the difference.
A Sends	The display shows a knob indicating the gain level for the first send in that track. If the track has no send, no knob will be shown, and the physical knob for that track will do nothing.
B Sends, C Sends, D Sends	Like "A Sends", but for the 2nd, 3rd and 4th sends of a track/bus respectively.

To change which tracks are shown while in global mix mode, use the left and right arrow/cursor keys just below and to the right of the display. Tracks and busses that are hidden in Ardour's GUI will also be hidden from display on the Push 2.

To select a track/bus directly from the Push 2, press the corresponding button below the display. The track name will be highlighted, and the selection will change in Ardour's GUI as well (and also any other control surfaces).

#### 149.5.1 Soloing and Muting in Global Mix mode

The Solo and Mute buttons to the left of the video display can be used to solo and mute tracks while in Global Mix mode. The operation will be applied to the *first* currently selected track(s).

There are two indications that one or more tracks are soloed:

1. The solo button will blink red
2. Track names will be prefixed by "\*" if they are soloed, and "-" if they are muted due to soloing.

To cancel solo, either:

- Select the soloed track(s) and press the solo button again
- Press and hold the solo button for more than 1 second

#### 149.6 Track Mix

Track Mix mode allows you to focus on a single track in more detail than is possible in Global Mix mode. To enter (or leave) Track Mix mode, press the "Mix" button.

In Track Mix mode, various aspects of the state of the first selected track/bus will be displayed on the Push 2. Above the display, the first 4 knobs control track volume (gain), pan direction/azimuth, pan width, and where appropriate, track input trim.

Below the display, 7 buttons provide immediate control of mute, solo, rec-enable, monitoring (input or disk or automatic), solo isolate and solo safe state. When a track is muted due to other track(s) soloing, the mute button will flash (to differentiate from its state when it is explicitly muted).

The video display also shows meters for the track, which as in Global Mix mode, precisely match the meter type shown in Ardour's GUI. There are also two time displays showing the current playhead position in both musical (beats|bars|ticks) format, and as hours:minutes:seconds.

To change which track is visible in Track Mix mode, use the left/right arrow/cursor keys just below and to the right of the video display.

## 149.7 Scale Selection

Press the Scale button to enter Scale mode. The display will look like this:

Close	C	G	D	A	E	B	>
	Dorian Ionian (Major) Minor	Phrygian Lydian Mixolydian	Pentatonic Minor Chromatic BluesScale	Double Harmonic Enigmatic Hirajoshi	Iwato Hindu Spanish 8 Tone	Leading Whole Tone Arabian Balinese	
	Harmonic Minor Melodic Minor Asc. Melodic Minor Desc.	Aeolian Locrian	Neapolitan Minor Neapolitan Major	Hungarian Minor Hungarian Major	Pelog Hungarian Gypsy	Gypsy Mohammedan	
InKey Chromatic	Pentatonic Major	Oriental	Kumoi	Overtone		Javanese	
	F	B♭/A♯	E♭/D♯	A♭/G♯	D♭/C♯	G♭/F♯	

Fig. 4: Track mix mode on Push2 screen

In the center, 37 scales are presented. Scroll through them by either using the cursor/arrow keys to the lower right of the display, or the knobs above the display. The scale will change dynamically as you scroll. You can also scroll in whole pages using the upper right and upper left buttons above the display (they will display “<” and “>” if scrolling is possible).

To change the root note of the scale, press the corresponding button above or below the video display. The button will be lit to indicate your selection (and the text will be highlighted).

By default, Ardour configures the Push 2 pads to use “in-key” mode, where all pads correspond to notes “in” the chosen scale. Notes corresponding to the root note, or the equivalent note in higher octaves, are highlighted with the color of the current target MIDI track.

In “chromatic” mode, the pads correspond to a continuous sequence of notes starting with your selected root note. Pads corresponding to notes in the scale are illuminated; those corresponding to the root note are lit with the color of the current target MIDI track. Other pads are left dark, but you can still play them.

To switch between them, press button on the lower left of the video display; the text above it will display the current mode (though it is usually visually self-evident from the pad lighting pattern).

To leave Scale mode, press the “Scale” button again. You may also use the upper left button above the display, though if you have scrolled left, it may require more than one press.

## 149.8 Specific Button/Knob Functions

In addition to the layouts described above, many (but not all) of the buttons and knobs around the edges of the Push 2 will carry out various functions related to their (illuminated) label. As of Ardour 5.4, this

includes:

Metronome (button and adjacent knob)	Enables/disables the click (metronome). The knob directly above it will control the volume (gain) of the click.
Undo/Redo	Undo or redo the previous editing operation.
Delete	Deletes the currently selected region, or range, or note. Equivalent to using Ctrl/Cmd-x on the keyboard.
Quantize	If a MIDI region is selected in Ardour, this will open the quantize dialog.
Duplicate	Duplicates the current region or range selection.
Rec-Enable	Enables and disables Ardour's global record enable state.
Play	Starts and stops the transport. Press Shift-Play to return to the session start.
Add Track	Opens Ardour's Add Track/Bus dialog.
Browse	Open's Ardour's import dialog to select and audition existing audio and MIDI files.
Master	Pressing this button jumps directly to Track Mix mode, with the master out bus displayed.
Cursor arrows	These are used by some modes to navigate within the display (e.g Scale mode). In other modes, the up/down cursor arrows will scroll the GUI display up and down, while the left/right cursor arrows will generally scroll within the Push 2 display itself.
Repeat	Enables/disables loop playback. This will follow Ardour's "loop is mode" preference, just like the loop button in the Ardour GUI.
Octave buttons	These shift the root note of the current pad scale up or down by 1 octave.
Page buttons	These scroll Ardour's editor display left and right along the timeline.
Master (top right) knob	This knob controls the gain/volume of Ardour's main output. If the session has a monitor section.

**PART XII - SCRIPTING**



## **82 - LUA SCRIPTING**

Starting with version 4.7.213, Ardour supports Lua scripts.

This Documentation is Work in Progress and far from complete. Also the documented API may be subject to change.

### **151.1 Preface**

There are cases that Ardour cannot reasonably cater to with core functionality alone, either because they're session specific or user specific edge cases.

Examples for these include voice-activate (record-arm specific tracks and roll transport depending on signal levels), rename all regions after a specific timecode, launch an external application when a certain track is soloed, generate automation curves or simply provide a quick shortcut for a custom batch operation.

Cases like this call for means to extend the DAW without actually changing the DAW itself. This is where scripting comes in.

“Scripting” refers to tasks that could alternatively be executed step-by-step by a human operator.

Lua is a tiny and simple language which is easy to learn, yet allows for comprehensive solutions. Lua is also a glue language it allows to tie existing component in Ardour together in unprecedented ways, and most importantly Lua is one of the few scripting-languages which can be safely used in a real-time environment.

A good introduction to Lua is the book [Programming in Lua](#). The first edition is available online, but if you have the means buy a copy of the book, it not only helps to support the Lua project, but provides for a much nicer reading and learning experience.

### **151.2 Overview**

The core of Ardour is a real-time audio engine that runs and processes audio. One interfaces with an engine by sending it commands. Scripting can be used to interact with or modify the active Ardour session, just like a user uses the Editor/Mixer GUI to modify the state or parameters of the session.

Doing this programmatically requires some knowledge about the objects used internally. Most Ardour C++ objects and their methods are directly exposed to Lua and one can call functions or modify variables:

```
C++ session->set_transport_speed (1.0);
```

```
Lua Session:set_transport_speed (1.0)
```

You may notice that there is only a small syntactic difference in this case. While C++ requires recompiling the application for every change, Lua script can be loaded, written or modified while the application is

running. Lua also abstracts away many of the C++ complexities such as object lifetime, type conversion and null-pointer checks.

Close ties with the underlying C++ components is where the power of scripting comes from. A script can orchestrate interaction of lower-level components which take the bulk of the CPU time of the final program.

At the time of writing Ardour integrates Lua 5.3.2: [Lua 5.3 reference manual](#).

## 151.3 Integration

Like Control surfaces and the GUI, Lua Scripts are confined to certain aspects of the program. Ardour provides the framework and runs Lua (not the other way around).

In Ardour's case Lua is available:

Editor Action Scripts	User initiated actions (menu, shortcuts) for batch processing
Editor Hooks/Callbacks	Event triggered actions for the Editor/Mixer GUI
Session Scripts	Scripts called at the start of every audio cycle (session, real-time)
DSP Scripts	Audio/Midi processor - plugins with access to the Ardour session (per track/bus, real-time)
Script Console	Action Script commandline

There are also a special mode:

Commandline Tool	Replaces the complete Editor GUI, direct access to libardour (no GUI) from the commandline. <i>Be aware that the vast majority of complex functionality is provided by the Editor UI.</i>
------------------	--

## 151.4 Managing Scripts

Ardour searches for Lua scripts in the `scripts` folder in `$ARDOUR_DATA_PATH`, Apart from scripts included directly with Ardour, this includes

GNU/Linux	<code>\$HOME/.config/ardour5/scripts</code>
Mac OS X	<code>\$HOME/Library/Preferences/Ardour5/scripts</code>
Windows	<code>%localappdata%\ardour5\scripts</code>

Files must end with `.lua` file extension.

Scripts are managed via the GUI

Editor Action Scripts	Menu → Edit → Scripted Actions → Manage
Editor Hooks/Callbacks	Menu → Edit → Scripted Actions → Manage
Session Scripts	Menu → Session → Scripting → Add/Remove Script
DSP Scripts	Mixer-strip → context menu (right click) → New Lua Proc
Script Console	Menu → Window → Scripting

## 151.5 Script Layout

- Every script must include an `ardour` descriptor table. Required fields are “Name” and “Type”.
- A script must provide a *Factory method*: A function with optional instantiation parameters which returns the actual script.
- [optional]: list of parameters for the “factory”.
- in case of DSP scripts, an optional list of automatable parameters and possible audio/midi port configurations, and a `dsp_run` function, more on that later.

A minimal example script looks like:

```
ardour {
    ["type"]      = "EditorAction",
    name         = "Rewind",
}

function factory (unused_params)
    return function ()
        Session:goto_start() -- rewind the transport
    end
end
```

The common part for all scripts is the “Descriptor”. It’s a Lua function which returns a table (key/values) with the following keys (the keys are case-sensitive):

type [required]	one of “DSP”, “Session”, “EditorHook”, “EditorAction” (the type is not case-sensitive)
name [re-quired]	Name/Title of the script
author	Your Name
license	The license of the script (e.g. “GPL” or “MIT”)
description	A longer text explaining to the user what the script does

Scripts that come with Ardour (currently mostly examples) can be found in the Source Tree.

### 151.5.1 Action Scripts

Action scripts are the simplest form. An anonymous Lua function is called whenever the action is triggered. A simple action script is shown above.

There are 10 action script slots available, each of which is a standard GUI action available from the menu and hence can be bound to a keyboard shortcut.

### 151.5.2 Session Scripts

Session scripts similar to Actions Scripts, except the anonymous function is called periodically every process cycle. The function receives a single parameter - the number of audio samples which are processed in the given cycle

```

ardour {
    ["type"]      = "session",
    name         = "Example Session Script",
    description  = [[
        An Example Ardour Session Script.
        This example stops the transport after rolling for a specific time.]]
}

-- instantiation options, these are passed to the "factory" method below
function sess_params ()
    return
{
    ["print"]   = { title = "Debug Print (yes/no)", default = "no", optional = true },
    ["time"]    = { title = "Timeout (sec)", default = "90", optional = false },
}
end

function factory (params)
    return function (n_samples)
        local p = params["print"] or "no"
        local timeout = params["time"] or 90
        a = a or 0
        if p ~= "no" then print (a, n_samples, Session:frame_rate (), Session:transport_
rolling ()) end -- debug output (not rt safe)
        if (not Session:transport_rolling()) then
            a = 0
            return
        end
        a = a + n_samples
        if (a > timeout * Session:frame_rate()) then
            Session:request_transport_speed(0.0, true)
        end
    end
end

```

### 151.5.3 Action Hooks

Action hook scripts must define an additional function which returns a *Set* of Signal that which trigger the callback (documenting available slots and their parameters remains to be done).

```

ardour {
    ["type"]      = "EditorHook",
    name         = "Hook Example",
    description  = "Rewind On Solo Change, Write a file when regions are moved.",
}

function signals ()
    s = LuaSignal.Set()
    s:add (
    {

```

(continues on next page)

(continued from previous page)

```

        [LuaSignal.SoloActive] = true,
        [LuaSignal.RegionPropertyChanged] = true
    }
)
return s
end

function factory (params)
    return function (signal, ref, ...)
        -- print (signal, ref, ...)

        if (signal == LuaSignal.SoloActive) then
            Session:goto_start()
        end

        if (signal == LuaSignal.RegionPropertyChanged) then
            obj,pch = ...
            file = io.open ("/tmp/test" , "a")
            io.output (file
            io.write (string.format ("Region: '%s' pos-changed: %s, length-changed: %s\n",
                obj:name (),
                tostring (pch:containsFramePos (ARDOUR.Properties.Start)),
                tostring (pch:containsFramePos (ARDOUR.Properties.Length))
            )))
            io.close (file)
        end
    end
end

```

### 151.5.4 DSP Scripts

See the scripts folder for examples for now.

Some notes for further doc:

- required function: `dsp_ioconfig ()`: return a list of possible audio I/O configurations - follows Audio Unit conventions.
- optional function: `dsp_dsp_midi_input ()`: return true if the plugin can receive midi input
- optional function: `dsp_params ()`: return a table of possible parameters (automatable)
- optional function: `dsp_init (samplerate)`: called when instantiation the plugin with given samplerate.
- optional function: `dsp_configure (in, out)`: called after instantiation with configured plugin i/o.
- required function: `dsp_run (ins, outs, n_samples)` OR `dsp_runmap (bufs, in_map, out_map, n_samples, offset)`: DSP process callback. The former is a convenient abstraction that passes mapped buffers (as table). The latter is a direct pass-through matching Ardour's internal `::connect_and_run()` API, which requires the caller to map and offset raw buffers.
- plugin parameters are handled via the global variable `CtrlPorts`.
- midi data is passed via the global variable `mididata` which is valid during `dsp_run` only. (`dsp_runmap` requires the script to pass raw data from the buffers according to `in_map`)

- The script has access to the current session via the global variable `Session`, but access to the session methods are limited to realtime safe functions

## 151.6 Accessing Ardour Objects

The top most object in Ardour is the `ARDOUR::Session`. Fundamentally, a Session is just a collection of other things: Routes (tracks, busses), Sources (Audio/Midi), Regions, Playlists, Locations, Tempo map, Undo/Redo history, Ports, Transport state and controls, etc.

Every Lua interpreter can access it via the global variable `Session`.

GUI context interpreters also have an additional object in the global environment: The Ardour `Editor`. The Editor provides access to high level functionality which is otherwise triggered via GUI interaction such as undo/redo, open/close windows, select objects, drag/move regions. It also holds the current UI state: snap-mode, zoom-range, etc. The Editor also provides complex operations such as “import audio” which under the hood, creates a new Track, adds a new Source Objects (for every channel) with optional resampling, creates both playlist and regions and loads the region onto the Track all the while displaying a progress information to the user.

Documenting the bound C++ methods and class hierarchy is somewhere on the ToDo list. Meanwhile `luabindings.cc` is the best we can offer.

## 151.7 Concepts

- There are no bound constructors: Lua asks Ardour to create objects (e.g. add a new track), then receives a reference to the object to modify it.
- Scripts, once loaded, are saved with the Session (no reference to external files). This provides for portable Sessions.
- Lua Scripts are never executed directly. They provide a “factory” method which can have optional instantiation parameters, which returns a Lua closure.
- No external Lua modules/libraries can be used, scripts need to be self contained (portable across different systems (libs written in Lua can be used, and important c-libs/functions can be included with Ardour if needed).

Ardour is a highly multithreaded application and interaction between the different threads, particularly real-time threads, needs to be done with care. This part has been abstracted away by providing separate Lua interpreters in different contexts and restricting available interaction:

- Editor Actions run in a single instance interpreter in the GUI thread.
- Editor Hooks connect to libardour signals. Every Callback uses a dedicated Lua interpreter which is in the GUI thread context.
- All Session scripts run in a single instance in the main real-time thread (audio callback)
- DSP scripts have a separate instance per script and run in one of the DSP threads.

The available interfaces differ between contexts. For example, it is not possible to create new tracks or import audio from real-time context; while it is not possible to modify audio buffers from the GUI thread.

## 151.8 Current State

Fully functional, yet still in a prototyping stage:

- The GUI to add/configure scripts is rather minimalistic.
- The interfaces may change (particularly DSP, and Session script `run()`).
- Further planned work includes:
  - Built-in Script editor (customize/modify Scripts in-place)
  - convenience methods (wrap more complex Ardour actions into a library). e.g set plugin parameters, write automation lists from a Lua table
  - Add some useful scripts and more examples
  - Documentation (Ardour API), also usable for tab-expansion, syntax highlighting
  - bindings for GUI Widgets (plugin UIs, message boxes, etc)
- 

## 151.9 Examples

Apart from the scripts included with the source-code here are a few examples without further comments...

### 151.9.1 Editor Console Examples

```
print (Session:route_by_remote_id(1):name())

a = Session:route_by_remote_id(1);
print (a:name());

print(Session:get_tracks():size())

for i, v in ipairs(Session:unknown_processors():table()) do print(v) end
for i, v in ipairs(Session:get_tracks():table()) do print(v:name()) end

for t in Session:get_tracks():iter() do print(t:name()) end
for r in Session:get_routes():iter() do print(r:name()) end

Session:tempo_map():add_tempo(ARDOUR.Tempo(100,4), Timecode.BBT_TIME(4,1,0))

Editor:set_zoom_focus(Editting.ZoomFocusRight)
print(Editting.ZoomFocusRight);
Editor:set_zoom_focus(1)

files = C.StringVector();
files:push_back("/home/rgareus/data/coding/ltc-tools/smpTE.wav")
pos = -1
```

(continues on next page)

(continued from previous page)

```

Editor:do_import(files, Editing.ImportDistinctFiles, Editing.ImportAsTrack, ARDOUR.
    ↵SrcQuality.SrcBest, pos, ARDOUR.PluginInfo())

# or in one line:
Editor:do_import(C.StringVector():add={"/path/to/file.wav"}), Editing.
    ↵ImportDistinctFiles, Editing.ImportAsTrack, ARDOUR.SrcQuality.SrcBest, -1, ARDOUR.
    ↵PluginInfo()

# called when a new session is loaded:
function new_session (name) print("NEW SESSION:", name) end

# read/set/describe a plugin parameter
route = Session:route_by_remote_id(1)
processor = route:nth_plugin(0)
plugininsert = processor:to_insert()

plugin = plugininsert:plugin(0)
print (plugin:label())
print (plugin:parameter_count())

x = ARDOUR.ParameterDescriptor ()
_, t = plugin:get_parameter_descriptor(2, x) -- port #2
paramdesc = t[2]
print (paramdesc.lower)

ctrl = Evoral.Parameter(ARDOUR.AutomationType.PluginAutomation, 0, 2)
ac = plugininsert:automation_control(ctrl, false)
print (ac:get_value ())
ac:set_value(1.0, PBD.GroupControlDisposition.NoGroup)

# the same using a convenience wrapper:
route = Session:route_by_remote_id(1)
proc = t:nth_plugin (i)
ARDOUR.LuaAPI.set_processor_param (proc, 2, 1.0)

```

### 151.9.2 Commandline Session

The standalone tool `luasession` allows one to access an Ardour session directly from the commandline. Interaction is limited by the fact that most actions in Ardour are provided by the Editor GUI.

`luasession` provides only two special functions `load_session` and `close_session` and exposes the `AudioEngine` instance as global variable.

```

for i,_ in AudioEngine:available_backends():iter() do print (i.name) end

backend = AudioEngine:set_backend("ALSA", "", "")
print (AudioEngine:current_backend_name())

for i,_ in backend:enumerate_devices():iter() do print (i.name) end

```

(continues on next page)

(continued from previous page)

```
backend:set_input_device_name("HDA Intel PCH")
backend:set_output_device_name("HDA Intel PCH")

print (backend:buffer_size())
print (AudioEngine:get_last_backend_error())

s = load_session ("/home/rgareus/Documents/ArdourSessions/lua2/", "lua2")
s:request_transport_speed (1.0)
print (s:transport_rolling())
s:goto_start()
close_session()
```



## 83 - LUA BINDINGS CLASS REFERENCE

This documentation is far from complete may be inaccurate and subject to change.

### Overview

The top-level entry point are *ARDOUR:Session* and *ArdourUI:Editor*. Most other Classes are used indirectly starting with a Session function. e.g. *Session:get\_routes()*.

A few classes are dedicated to certain script types, e.g. Lua DSP processors have exclusive access to *ARDOUR.DSP* and *ARDOUR:ChanMapping*. Action Hooks Scripts to *LuaSignal:Set* etc.

Detailed documentation (parameter names, method description) is not yet available. Please stay tuned.

### Short introduction to Ardour classes

Ardour's structure is object oriented. The main object is the Session. A Session contains Audio Tracks, Midi Tracks and Busses. Audio and Midi tracks are derived from a more general "Track" Object, which in turn is derived from a "Route" (aka Bus). (We say "An Audio Track *is-a* Track *is-a* Route"). Tracks contain specifics. For Example a track *has-a* diskstream (for file i/o).

Operations are performed on objects. One gets a reference to an object and then calls a method. e.g `obj = Session:route_by_name("Audio") obj:set_name("Guitar")`.

Lua automatically follows C++ class inheritance. e.g one can directly call all *SessionObject* and *Route* methods on *Track* object. However lua does not automatically promote objects. A *Route* object which just happens to be a *Track* needs to be explicitly cast to a *Track*. Methods for casts are provided with each class. Note that the cast may fail and return a *nil* reference.

Likewise multiple inheritance is a [non-trivial issue](#) in lua. To avoid performance penalties involved with lookups, explicit casts are required in this case. One example is *ARDOUR:SessionObject* which *is-a* *StatefulDestructible* which inherits from both *Stateful* and *Destructible*.

Object lifetimes are managed by the Session. Most Objects cannot be directly created, but one asks the Session to create or destroy them. This is mainly due to realtime constrains: you cannot simply remove a track that is currently processing audio. There are various *factory* methods for object creation or removal.

### Pass by Reference

Since lua functions are closures, C++ methods that pass arguments by reference cannot be used as-is. All parameters passed to a C++ method which uses references are returned as Lua Table. If the C++ method also returns a value it is prefixed. Two parameters are returned: the value and a Lua Table holding the parameters.

C++

```
void set_ref (int& var, long& val)
{
    printf ("%d %ld\n", var, val);
    var = 5;
    val = 7;
}
```

Lua

```
local var = 0;
ref = set_ref (var, 2);
-- output from C++ printf()
0 2
-- var is still 0 here
print (ref[1], ref[2])
5 7
```

```
int set_ref2 (int &var, std::string unused)
{
    var = 5;
    return 3;
}
```

```
rv, ref = set_ref2 (0, "hello");
print (rv, ref[1], ref[2])
3 5 hello
```

## Pointer Classes

Libardour makes extensive use of reference counted `boost::shared_ptr` to manage lifetimes. The Lua bindings provide a complete abstraction of this. There are no pointers in lua. For example a `ARDOUR:Route` is a pointer in C++, but lua functions operate on it like it was a class instance.

`shared_ptr` are reference counted. Once assigned to a lua variable, the C++ object will be kept and remains valid. It is good practice to assign references to lua local variables or reset the variable to `nil` to drop the ref.

All pointer classes have a `isnil()` method. This is for two cases: Construction may fail. e.g. `ARDOUR.LuaAPI.newplugin()` may not be able to find the given plugin and hence cannot create an object.

The second case is for `boost::weak_ptr`. As opposed to `boost::shared_ptr` weak-pointers are not reference counted. The object may vanish at any time. If lua code calls a method on a nil object, the interpreter will raise an exception and the script will not continue. This is not unlike `a = nil a:test()` which results in an error “attempt to index a nil value”.

From the lua side of things there is no distinction between weak and shared pointers. They behave identically. Below they’re indicated in orange and have an arrow to indicate the pointer type. Pointer Classes cannot be created in lua scripts. It always requires a call to C++ to create the Object and obtain a reference to it.

## Class Documentation

## ARDOUR

Methods

*RCCConfiguration*

config()

## 152.1 ARDOUR:Amp

*C#:* boost::shared\_ptr< ARDOUR::Amp >, boost::weak\_ptr< ARDOUR::Amp >

is-a: *ARDOUR:Processor*

Applies a declick operation to all audio inputs, passing the same number of audio outputs, and passing through any other types unchanged.

Methods

*GainControl*

gain\_control()

bool

isnil()

### 152.1.1 Inherited from ARDOUR:Processor

Methods

void

activate()

bool

active()

void

deactivate()

std::string

display\_name()

bool

display\_to\_user()

*ChanCount*

input\_streams()

*ChanCount*

output\_streams()

Cast

*Amp*

to\_amp()

*Automatable*

to\_automatable()

*PluginInsert*

to\_insert()

*IOProcessor*

to\_ioprocessor()

*PeakMeter*

to\_meter()

*MonitorProcessor*

to\_monitorprocessor()

*PeakMeter*

to\_peakmeter()

*PluginInsert*

to\_plugininsert()

*SideChain*

to\_sidechain()

*UnknownProcessor*

to\_unknownprocessor()

### 152.1.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.2 ARDOUR:AudioBackend

C#: boost::shared\_ptr< ARDOUR::AudioBackend >, boost::weak\_ptr< ARDOUR::AudioBackend >

AudioBackend is an high-level abstraction for interacting with the operating system's audio and midi I/O.

Methods

unsigned int

buffer\_size()

```
std::string  
device_name()  
std::string  
driver_name()  
override this if this implementation returns true from requires_driver_selection()  
float  
dsp_load()  
return the fraction of the time represented by the current buffer size that is being used for each buffer process cycle, as a value from 0.0 to 1.0
```

E.g. if the buffer size represents 5msec and current processing takes 1msec, the returned value should be 0.2.

Implementations can feel free to smooth the values returned over time (e.g. high pass filtering, or its equivalent).

#### *DeviceStatus Vector*

```
enumerate_devices()
```

Returns a collection of DeviceStatuses identifying devices discovered by this backend since the start of the process.

Any of the names in each DeviceStatus may be used to identify a device in other calls to the backend, though any of them may become invalid at any time.

#### *StringVector*

```
enumerate_drivers()
```

If the return value of requires\_driver\_selection() is true, then this function can return the list of known driver names.

If the return value of requires\_driver\_selection() is false, then this function should not be called. If it is called its return value is an empty vector of strings.

#### *DeviceStatus Vector*

```
enumerate_input_devices()
```

Returns a collection of DeviceStatuses identifying input devices discovered by this backend since the start of the process.

Any of the names in each DeviceStatus may be used to identify a device in other calls to the backend, though any of them may become invalid at any time.

#### *DeviceStatus Vector*

```
enumerate_output_devices()
```

Returns a collection of DeviceStatuses identifying output devices discovered by this backend since the start of the process.

Any of the names in each DeviceStatus may be used to identify a device in other calls to the backend, though any of them may become invalid at any time.

#### *AudioBackendInfo*

```
info()
```

Return the AudioBackendInfo object from which this backend was constructed.

```
unsigned int
input_channels()
std::string
input_device_name()
bool
isnil()
unsigned int
output_channels()
std::string
output_device_name()
unsigned int
period_size()
float
sample_rate()
int
set_buffer_size(unsigned int)
Set the buffer size to be used.

The device is assumed to use a double buffering scheme, so that one buffer's worth of data can be processed by hardware while software works on the other buffer. All known suitable audio APIs support this model (though ALSA allows for alternate numbers of buffers, and CoreAudio doesn't directly expose the concept).

int
set_device_name(std::string)
Set the name of the device to be used

int
set_driver(std::string)
Returns zero if the backend can successfully use

Should not be used unless the backend returns true from requires_driver_selection()

name as the driver, non-zero otherwise.

int
set_input_device_name(std::string)
Set the name of the input device to be used if using separate input/output devices.

use_separate_input_and_output_devices()
int
set_output_device_name(std::string)
Set the name of the output device to be used if using separate input/output devices.

use_separate_input_and_output_devices()
int
```

set\_period\_size(unsigned int)

Set the period size to be used. must be called before starting the backend.

int

set\_sample\_rate(float)

Set the sample rate to be used

bool

use\_separate\_input\_and\_output\_devices()

An optional alternate interface for backends to provide a facility to select separate input and output devices.

If a backend returns true then enumerate\_input\_devices() and enumerate\_output\_devices() will be used instead of enumerate\_devices() to enumerate devices. Similarly set\_input/output\_device\_name() should be used to set devices instead of set\_device\_name().

## 152.3 ARDOUR:AudioBackendInfo

*C++:* ARDOUR::AudioBackendInfo

Data Members

char\*

name

## 152.4 ARDOUR:AudioBuffer

*C++:* ARDOUR::AudioBuffer

Buffer containing audio data.

Methods

void

apply\_gain(float, long)

bool

check\_silence(unsigned int, unsigned int&)

check buffer for silence

**nframes** number of frames to check

**n** first non zero sample (if any)

Returns true if all samples are zero

*FloatArray*

data(long)

void

read\_from(*FloatArray*, long, long, long)

void

`silence(long, long)`  
silence buffer  
`len` number of samples to clear  
`offset` start offset

## 152.5 ARDOUR:AudioEngine

*C++:* ARDOUR::AudioEngine  
is-a: *ARDOUR:PortManager*

Methods

*BackendVector*

`available_backends()`  
std::string

`current_backend_name()`  
float

`get_dsp_load()`  
std::string

`get_last_backend_error()`

*AudioBackend*

`set_backend(std::string, std::string, std::string)`  
int

`set_buffer_size(unsigned int)`  
int

`set_device_name(std::string)`  
int

`set_sample_rate(float)`  
bool

`setup_required()`  
int

`start(bool)`  
int

`stop(bool)`

### 152.5.1 Inherited from ARDOUR:PortManager

Methods

int

connect(std::string, std::string)

bool

connected(std::string)

int

disconnect(std::string, std::string)

int

disconnect\_port(*Port*)

*LuaTable*(int, ...)

get\_backend\_ports(std::string, *Data Type*, *PortFlags*, *StringVector*&)

*LuaTable*(int, ...)

get\_connections(std::string, *StringVector*&)

void

get\_physical\_inputs(*Data Type*, *StringVector*&, *MidiPortFlags*, *MidiPortFlags*)

void

get\_physical\_outputs(*Data Type*, *StringVector*&, *MidiPortFlags*, *MidiPortFlags*)

*Port*

get\_port\_by\_name(std::string)

**name** Full or short name of port

Returns Corresponding Port or 0.

*LuaTable*(int, ...)

get\_ports(*Data Type*, *PortList*&)

std::string

get\_pretty\_name\_by\_name(std::string)

*ChanCount*

n\_physical\_inputs()

*ChanCount*

n\_physical\_outputs()

bool

physically\_connected(std::string)

*PortEngine*

port\_engine()

bool

port\_is\_physical(std::string)

## 152.6 ARDOUR:AudioPlaylist

*C<sub>++</sub>*: boost::shared\_ptr< ARDOUR::AudioPlaylist >, boost::weak\_ptr< ARDOUR::AudioPlaylist >

is-a: *ARDOUR:Playlist*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

bool

isnil()

long

read(*FloatArray*,*FloatArray*,*FloatArray*, long, long, unsigned int)

### 152.6.1 Inherited from ARDOUR:Playlist

Methods

void

add\_region(*Region*, long, float, bool, int, double, bool)

Note: this calls set\_layer (... , DBL\_MAX) so it will reset the layering index of region

*Region*

combine(*RegionList*)

unsigned int

count\_regions\_at(long)

*Playlist*

cut(*AudioRangeList*&, bool)

*DataType*

data\_type()

void

duplicate(*Region*, long, long, float)

**gap** from the beginning of the region to the next beginning

void

duplicate\_range(*AudioRange*&, float)

void

duplicate\_until(*Region*, long, long, long)

**gap** from the beginning of the region to the next beginning

**end** the first frame that does not contain a duplicated frame

*Region*`find_next_region(long, RegionPoint, int)``long``find_next_region_boundary(long, int)``long``find_next_transient(long, int)``void``lower_region(Region)``void``lower_region_to_bottom(Region)``unsigned int``n_regions()``void``raise_region(Region)``void``raise_region_to_top(Region)`*Region*`region_by_id(ID)``RegionListPtr``region_list()``RegionListPtr``regions_at(long)``RegionListPtr``regions_touched(long, long)`

**start** Range start.

**end** Range end.

Returns regions which have some part within this range.

`RegionListPtr``regions_with_end_within(Range)``RegionListPtr``regions_with_start_within(Range)``void``remove_region(Region)``void``split(long)``void`

split\_region(*Region*,*MusicFrame*)

*Region*

top\_region\_at(long)

*Region*

top\_unmuted\_region\_at(long)

void

uncombine(*Region*)

Cast

*AudioPlaylist*

to\_audioplaylist()

*MidiPlaylist*

to\_midiplaylist()

## 152.6.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.7 ARDOUR:AudioPort

*C++:* boost::shared\_ptr< ARDOUR::AudioPort >, boost::weak\_ptr< ARDOUR::AudioPort >

is-a: *ARDOUR:Port*

Methods

bool

isnil()

### 152.7.1 Inherited from ARDOUR:Port

Methods

int

connect(std::string)

bool

connected()

Returns true if this port is connected to anything

bool

connected\_to(std::string)

o Port name

Returns true if this port is connected to o, otherwise false.

int

disconnect(std::string)

int

disconnect\_all()

std::string

name()

Returns Port short name

std::string

pretty\_name(bool)

Returns Port human readable name

bool

receives\_input()

Returns true if this Port receives input, otherwise false

bool

sends\_output()

Returns true if this Port sends output, otherwise false

Cast

*AudioPort*

to\_audioport()

*MidiPort*

to\_midiport()

## 152.8 ARDOUR:AudioRange

C<sub>#</sub>: ARDOUR::AudioRange

Constructor

ARDOUR.AudioRange(long, long, unsigned int)

Methods

bool

equal(*AudioRange*)

long

length()

Data Members

long

end

unsigned int

id

long

start

## 152.9 ARDOUR:AudioRangeList

*C<sub>r</sub>t*: std::list<ARDOUR::AudioRange >

Constructor

ARDOUR.AudioRangeList()

Methods

*AudioRange*

back()

bool

empty()

*AudioRange*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.10 ARDOUR:AudioRegion

*C<sub>++</sub>*: boost::shared\_ptr< ARDOUR::AudioRegion >, boost::weak\_ptr< ARDOUR::AudioRegion >  
is-a: *ARDOUR:Region*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

*AudioSource*

audio\_source(unsigned int)

bool

isnil()

double

maximum\_amplitude(*Progress*)

Returns the maximum (linear) amplitude of the region, or a -ve number if the Progress object reports that the process was cancelled.

double

rms(*Progress*)

Returns the maximum (rms) signal power of the region, or a -1 if the Progress object reports that the process was cancelled.

float

scale\_amplitude()

*LuaTable*(int, ...)

separate\_by\_channel(*RegionVector*&)

void

set\_scale\_amplitude(float)

### 152.10.1 Inherited from ARDOUR:Region

Methods

bool

at\_natural\_position()

bool

automatic()

bool

can\_move()

bool

captured()

void

clear\_sync\_position()

*Control*

control(*Parameter*, bool)

bool

covers(long)

void

cut\_end(long, int)

void

cut\_front(long, int)

*Data Type*

data\_type()

bool

external()

bool

has\_transients()

bool

hidden()

bool

import()

bool

is\_compound()

unsigned int

layer()

long

length()

bool

locked()

void

lower()

void

lower\_to\_bottom()

*StringVector*

master\_source\_names()

*SourceList*

master\_sources()

void

```
move_start(long, int)
void
move_to_natural_position()
bool
muted()
unsigned int
n_channels()
void
nudge_position(long)
bool
opaque()
long
position()
```

How the region parameters play together:

POSITION: first frame of the region along the timeline  
START: first frame of the region within its source(s)  
LENGTH: number of frames the region represents

```
bool
position_locked()
double
quarter_note()
void
raise()
void
raise_to_top()
void
set_hidden(bool)
void
set_initial_position(long)
```

A gui may need to create a region, then place it in an initial position determined by the user. When this takes place within one gui operation, we have to reset `_last_position` to prevent an implied move.

```
void
set_length(long, int)
void
set_locked(bool)
void
set_muted(bool)
void
```

```
set_opaque(bool)
void
set_position(long, int)
void
set_position_locked(bool)
void
set_start(long)
void
set_sync_position(long)
Set the region's sync point.
absolute_pos Session time.
void
set_video_locked(bool)
float
shift()
Source
source(unsigned int)
long
start()
float
stretch()
bool
sync_marked()
LuaTable(long, ...)
sync_offset(int&)
long
sync_position()
Returns Sync position in session time
Int64List
transients()
void
trim_end(long, int)
void
trim_front(long, int)
void
trim_to(long, long, int)
```

bool  
video\_locked()  
bool  
whole\_file()  
Cast  
*AudioRegion*  
to\_audioregion()  
*MidiRegion*  
to\_midiregion()  
*Readable*  
to\_readable()

### 152.10.2 Inherited from ARDOUR:SessionObjectPtr

Methods  
std::string  
name()  
Cast  
*Stateful*  
to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.11 ARDOUR: AudioSource

*C#:* boost::shared\_ptr< ARDOUR::

is-a: *ARDOUR:Source*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods  
std::string  
captured\_for()  
bool  
empty()  
bool  
isnil()  
bool

```
isnil()
long
length(long)
unsigned int
n_channels()
unsigned int
n_channels()
long
read(FloatArray, long, long, int)
long
readable_length()
long
readable_length()
float
sample_rate()
Cast
Readable
to_readable()
```

### 152.11.1 Inherited from ARDOUR:Source

```
Methods
std::string
ancestor_name()
bool
can_be_analysed()
bool
destructive()
bool
has_been_analysed()
long
natural_position()
long
timeline_position()
long
timestamp()
int
```

```
use_count()
bool
used()
bool
writable()
Cast

```

### 152.11.2 Inherited from ARDOUR:SessionObjectPtr

Methods  
std::string  
name()  
Cast  
*Stateful*  
to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.12 ARDOUR:AudioTrack

*C++:* boost::shared\_ptr< ARDOUR::AudioTrack >, boost::weak\_ptr< ARDOUR::AudioTrack >

is-a: *ARDOUR:Track*

A track is an route (bus) with a recordable diskstream and related objects relevant to tracking, playback and editing.

Specifically a track has regions and playlist objects.

Methods  
bool  
isnil()

### 152.12.1 Inherited from ARDOUR:Track

Methods

*Region*

bounce(*InterThreadInfo&*)

bounce track from session start to session end to new region

**itt** asynchronous progress report and cancel

Returns a new audio region (or nil in case of error)

*Region*

bounce\_range(long, long,*InterThreadInfo&*, *Processor*, bool)

Bounce the given range to a new audio region.

**start** start time (in samples)

**end** end time (in samples)

**itt** asynchronous progress report and cancel

**endpoint** the processor to tap the signal off (or nil for the top)

**include\_endpoint** include the given processor in the bounced audio.

Returns a new audio region (or nil in case of error)

bool

bounceable(*Processor*, bool)

Test if the track can be bounced with the given settings. If sends/inserts/returns are present in the signal path or the given track has no audio outputs bouncing is not possible.

**endpoint** the processor to tap the signal off (or nil for the top)

**include\_endpoint** include the given processor in the bounced audio.

Returns true if the track can be bounced, or false otherwise.

bool

can\_record()

*Playlist*

playlist()

bool

set\_name(std::string)

Cast

*AudioTrack*

to\_audio\_track()

*MidiTrack*

to\_midi\_track()

### 152.12.2 Inherited from ARDOUR:Route

Methods

bool

active()

int

add\_processor\_by\_index(*Processor*, int, *ProcessorStreams*, bool)

Add a processor to a route such that it ends up with a given index into the visible processors.

**index** Index to add the processor at, or -1 to add at the end of the list.

Returns 0 on success, non-0 on failure.

bool

add\_sidechain(*Processor*)

*Amp*

amp()

std::string

comment()

bool

customize\_plugin\_insert(*Processor*, unsigned int, *ChanCount*, *ChanCount*)

enable custom plugin-insert configuration

**proc** Processor to customize

**count** number of plugin instances to use (if zero, reset to default)

**outs** output port customization

**sinks** input pins for variable-I/O plugins

Returns true if successful

*IO*

input()

*Delivery*

main\_outs()

the signal processor at the end of the processing chain which produces output

bool

muted()

*ChanCount*

n\_inputs()

*ChanCount*

n\_outputs()

*Processor*

nth\_plugin(unsigned int)

*Processor*

nth\_processor(unsigned int)

*Processor*

nth\_send(unsigned int)

*IO*

output()

*PannerShell*

panner\_shell()

*PeakMeter*

peak\_meter()

\*\*\*\*\* Pure interface begins  
here\*\*\*\*\*

int

remove\_processor(*Processor*,*ProcessorStreams*, bool)

remove plugin/processor

**proc** processor to remove

**err** error report (index where removal failed, channel-count why it failed) may be nil

**need\_process\_lock** if locking is required (set to true, unless called from RT context with lock)

Returns 0 on success

int

remove\_processors(*ProcessorList*,*ProcessorStreams*)

bool

remove\_sidechain(*Processor*)

int

reorder\_processors(*ProcessorList*,*ProcessorStreams*)

int

replace\_processor(*Processor*,*Processor*,*ProcessorStreams*)

replace plugin/processor with another

**old** processor to remove

**sub** processor to substitute the old one with

**err** error report (index where removal failed, channel-count why it failed) may be nil

Returns 0 on success

bool

reset\_plugin\_insert(*Processor*)

reset plugin-insert configuration to default, disable customizations.

This is equivalent to calling

```
customize_plugin_insert (proc, 0, unused)
```

**proc** Processor to reset

Returns true if successful

void

set\_active(bool, void\*)

void

set\_comment(std::string, void\*)

void

set\_meter\_point(*MeterPoint*, bool)

bool

set\_strict\_io(bool)

bool

soloed()

bool

strict\_io()

*Processor*

the\_instrument()

Return the first processor that accepts has at least one MIDI input and at least one audio output. In the vast majority of cases, this will be “the instrument”. This does not preclude other MIDI->audio processors later in the processing chain, but that would be a special case not covered by this utility function.

*Amp*

trim()

Cast

*Automatable*

to\_automatable()

*Slavable*

to\_slavable()

*Track*

to\_track()

### 152.12.3 Inherited from ARDOUR:Stripable

Methods

*AutomationControl*

comp\_enable\_control()

*AutomationControl*

comp\_makeup\_control()

*AutomationControl*

comp\_mode\_control()  
std::string  
comp\_mode\_name(unsigned int)

*ReadOnlyControl*

comp\_redux\_control()

*AutomationControl*

comp\_speed\_control()  
std::string  
comp\_speed\_name(unsigned int)

*AutomationControl*

comp\_threshold\_control()

unsigned int

eq\_band\_cnt()

std::string

eq\_band\_name(unsigned int)

*AutomationControl*

eq\_enable\_control()

*AutomationControl*

eq\_freq\_control(unsigned int)

*AutomationControl*

eq\_gain\_control(unsigned int)

*AutomationControl*

eq\_q\_control(unsigned int)

*AutomationControl*

eq\_shape\_control(unsigned int)

*AutomationControl*

filter\_enable\_controllable(bool)

*AutomationControl*

filter\_freq\_controllable(bool)

*AutomationControl*

filter\_slope\_controllable(bool)

*GainControl*

gain\_control()

bool

is\_auditioner()

```
bool  
is_hidden()  
bool  
is_master()  
bool  
is_monitor()  
bool  
is_selected()  
AutomationControl  
master_send_enable_control()  
MonitorProcessor  
monitor_control()  
MuteControl  
mute_control()  
AutomationControl  
pan_azimuth_control()  
AutomationControl  
pan_elevation_control()  
AutomationControl  
pan_frontback_control()  
AutomationControl  
pan_lfe_control()  
AutomationControl  
pan_width_control()  
PhaseControl  
phase_control()  
PresentationInfo  
presentation_info_ptr()  
AutomationControl  
rec_enable_control()  
AutomationControl  
rec_safe_control()  
AutomationControl  
send_enable_control(unsigned int)  
AutomationControl  
send_level_control(unsigned int)
```

```
std::string  
send_name(unsigned int)  
void  
set_presentation_order(unsigned int)  
SoloControl  
solo_control()  
SoloIsolateControl  
solo_isolate_control()  
SoloSafeControl  
solo_safe_control()  
GainControl  
trim_control()  
Cast  
Route  
to_route()  
VCA  
to_vca()
```

#### 152.12.4 Inherited from ARDOUR:SessionObjectPtr

Methods

```
std::string  
name()  
Cast  
Stateful  
to_stateful()  
StatefulDestructible  
to_statefuldestructible()
```

### 152.13 ARDOUR:AudioTrackList

*C++:* std::list<boost::shared\_ptr<ARDOUR::AudioTrack>>  
Constructor

ARDOUR.AudioTrackList()

Methods

*LuaTable*

add(LuaTable {*AudioTrack*})

*AudioTrack*

back()

bool

empty()

*AudioTrack*

front()

*LuaIter*

iter()

void

push\_back(*AudioTrack*)

void

reverse()

unsigned long

size()

*LuaTable*

table()

void

unique()

## 152.14 ARDOUR:Automatable

*C<sub>t</sub>*: boost::shared\_ptr< ARDOUR::Automatable >, boost::weak\_ptr< ARDOUR::Automatable >

is-a: *Evoral:ControlSet*

Methods

*AutomationControl*

automation\_control(*Parameter*, bool)

bool

isnil()

Cast

*Slavable*

to\_slavable()

## 152.15 ARDOUR:AutomatableSequence

*C<sub>++</sub>:* boost::shared\_ptr< ARDOUR::AutomatableSequence<Evoral::Beats> >, boost::weak\_ptr< ARDOUR::AutomatableSequence<Evoral::Beats> >

is-a: *ARDOUR:Automatable*

Methods

bool

isnil()

Cast

*Sequence*

to\_sequence()

### 152.15.1 Inherited from ARDOUR:Automatable

Methods

*AutomationControl*

automation\_control(*Parameter*, bool)

Cast

*Slavable*

to\_slavable()

## 152.16 ARDOUR:AutomationControl

*C<sub>++</sub>:* boost::shared\_ptr< ARDOUR::AutomationControl >, boost::weak\_ptr< ARDOUR::AutomationControl >

is-a: *PBD:Controllable*

A PBD::Controllable with associated automation data (AutomationList)

Methods

*AutomationList*

alist()

*AutoState*

automation\_state()

double

get\_value()

Get the current effective ‘user’ value based on automation state

bool

isnil()

void

```
set_automation_state(AutoState)
void
set_value(double,GroupControlDisposition)
Get and Set ‘internal’ value
All derived classes must implement this.
Basic derived classes will ignore
group_override, but more sophisticated children, notably those that proxy the value setting logic via an
object that is aware of group relationships between this control and others, will find it useful.
void
start_touch(double)
void
stop_touch(double)
bool
writable()
Cast
Control
to_ctrl()
SlavableAutomationControl
to_slavable()
```

### 152.16.1 Inherited from PBD:Controllable

Methods

```
std::string
name()
```

### 152.16.2 Inherited from PBD:StatefulPtr

Methods

```
void
clear_changes()
Forget about any changes to this object’s properties
ID
id()
OwnedPropertyList
properties()
```

## 152.17 ARDOUR:AutomationList

*C<sub>++</sub>:* boost::shared\_ptr< ARDOUR::AutomationList >, boost::weak\_ptr< ARDOUR::AutomationList >  
is-a: *Evoral:ControlList*

AutomationList is a stateful wrapper around Evoral::ControlList. It includes session-specifics (such as automation state), control logic (e.g. touch, signals) and acts as proxy to the underlying ControlList which holds the actual data.

Methods

*XMLNode*

get\_state()

bool

isnil()

*Command*

memento\_command(*XMLNode*,*XMLNode*)

bool

touch\_enabled()

bool

touching()

bool

writing()

Cast

*ControlList*

list()

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

### 152.17.1 Inherited from Evoral:ControlList

Methods

void

add(double, double, bool, bool)

add automation events

**when** absolute time in samples

**value** parameter value

**with\_guards** if true, add guard-points

**with\_initial** if true, add an initial point if the list is empty

void  
clear(double, double)  
remove all automation events between the given time range  
**start** start of range (inclusive) in audio samples  
**end** end of range (inclusive) in audio samples  
void  
clear\_list()  
double  
eval(double)  
query value at given time (takes a read-lock, not safe while writing automation)  
**where** absolute time in samples  
Returns parameter value  
*EventList*  
events()  
bool  
in\_write\_pass()  
*InterpolationStyle*  
interpolation()  
query interpolation style of the automation data  
Returns Interpolation Style  
*LuaTable*(double, ...)  
rt\_safe\_eval(double, bool&)  
realtime safe version of eval, may fail if read-lock cannot be taken  
**where** absolute time in samples  
**ok** boolean reference if returned value is valid  
Returns parameter value  
bool  
set\_interpolation(*InterpolationStyle*)  
set the interpolation style of the automation data.  
This will fail when asking for Logarithmic scale and min,max crosses 0 or Exponential scale with min != 0.  
**is** interpolation style  
Returns true if style change was successful  
void  
thin(double)  
Thin the number of events in this list.

The thinning factor corresponds to the area of a triangle computed between three points in the list (time-difference \* value-difference). If the area is large, it indicates significant non-linearity between the points.

Time is measured in samples, value is usually normalized to 0..1.

During automation recording we thin the recorded points using this value. If a point is sufficiently co-linear with its neighbours (as defined by the area of the triangle formed by three of them), we will not include it in the list. The larger the value, the more points are excluded, so this effectively measures the amount of thinning to be done.

**thinning\_factor** area-size (default: 20)

void

truncate\_end(double)

truncate the event list after the given time

**last\_coordinate** last event to include

void

truncate\_start(double)

truncate the event list to the given time

**overall\_length** overall length

## 152.18 ARDOUR:BackendVector

*C++:* std::vector<ARDOUR::AudioBackendInfo const\*>

Constructor

ARDOUR.BackendVector()

Methods

*AudioBackendInfo*

at(unsigned long)

bool

empty()

*LuaIter*

iter()

unsigned long

size()

*LuaTable*

table()

## 152.19 ARDOUR:BeatsFramesConverter

*C<sub>7</sub>:* ARDOUR::BeatsFramesConverter

Converter between quarter-note beats and frames. Takes distances in quarter-note beats or frames from some origin (supplied to the constructor in frames), and converts them to the opposite unit, taking tempo changes into account.

Constructor

ARDOUR.BeatsFramesConverter(*TempoMap*, long)

Methods

*Beats*

from(long)

Convert B time to A time (A from B)

long

to(*Beats*)

Convert A time to B time (A to B)

## 152.20 ARDOUR:BufferSet

*C<sub>7</sub>:* ARDOUR::BufferSet

A set of buffers of various types.

These are mainly accessed from Session and passed around as scratch buffers (e.g. as parameters to run() methods) to do in-place signal processing.

There are two types of counts associated with a BufferSet - available, and the ‘use count’. Available is the actual number of allocated buffers (and so is the maximum acceptable value for the use counts).

The use counts are how things determine the form of their input and inform others the form of their output (e.g. what they did to the BufferSet). Setting the use counts is realtime safe.

Methods

*ChanCount*

count()

*AudioBuffer*

get\_audio(unsigned long)

*MidiBuffer*

get\_midi(unsigned long)

## 152.21 ARDOUR:ChanCount

*C<sub>7</sub>:* ARDOUR::ChanCount

A count of channels, possibly with many types.

Operators are defined so this may safely be used as if it were a simple (single-typed) integer count of channels.

Constructor

ARDOUR.ChanCount(*Data Type*, unsigned int)

Convenience constructor for making single-typed streams (mono, stereo, midi, etc)

**type** data type

**count** number of channels

Methods

unsigned int

get(*Data Type*)

query channel count for given type

**type** data type

Returns channel count for given type

unsigned int

n\_audio()

query number of audio channels

Returns number of audio channels

unsigned int

n\_midi()

query number of midi channels

Returns number of midi channels

unsigned int

n\_total()

query total channel count of all data types

Returns total channel count (audio + midi)

void

reset()

zero count of all data types

void

set(*Data Type*, unsigned int)

set channel count for given type

**count** number of channels

**type** data type

## 152.22 ARDOUR:ChanMapping

*C $\ddagger$* : ARDOUR::ChanMapping

A mapping from one set of channels to another. The general form is 1 source (from), many sinks (to). numeric IDs are used to identify sources and sinks.

for plugins this is used to map “plugin-pin” to “audio-buffer”

Constructor

ARDOUR.ChanMapping()

Methods

*ChanCount*

count()

unsigned int

get(*Data Type*, unsigned int)

get buffer mapping for given data type and pin

**from** numeric source id

**type** data type

Returns mapped buffer number (or ChanMapping::Invalid)

bool

is\_monotonic()

Test if this mapping is monotonic (useful to see if inplace processing is feasible)

Returns true if the map is a strict monotonic set

unsigned int

n\_total()

void

set(*Data Type*, unsigned int, unsigned int)

set buffer mapping for given data type

**from** numeric source id

**to** buffer

**type** data type

## 152.23 ARDOUR:ControlList

*C $\ddagger$* : std::list<boost::shared\_ptr<ARDOUR::AutomationControl>>

Constructor

ARDOUR.ControlList()

Methods

*LuaTable*

add(LuaTable { *AutomationControl* })

*AutomationControl*

back()

bool

empty()

*AutomationControl*

front()

*LuaIter*

iter()

void

push\_back(*AutomationControl*)

void

reverse()

unsigned long

size()

*LuaTable*

table()

void

unique()

## 152.24 ARDOUR:ControlListPtr

*C++:* boost::shared\_ptr<std::list<boost::shared\_ptr<ARDOUR::AutomationControl>>>

Constructor

ARDOUR.ControlListPtr()

Methods

*LuaTable*

add(LuaTable { *AutomationControl* })

bool

empty()

*LuaIter*

iter()

void

```
push_back(AutomationControl)
void
reverse()
unsigned long
size()
LuaTable
table()
void
unique()
```

## 152.25 ARDOUR.DSP

Methods

```
float
accurate_coefficient_to_dB(float)
void
apply_gain_to_buffer(FloatArray, unsigned int, float)
float
compute_peak(FloatArray, unsigned int, float)
void
copy_vector(FloatArray,FloatArray, unsigned int)
float
dB_to_coefficient(float)
float
fast_coefficient_to_dB(float)
void
find_peaks(FloatArray, unsigned int,FloatArray,FloatArray)
float
log_meter(float)
non-linear power-scale meter deflection
power signal power (dB)
>Returns deflected value
float
log_meter_coeff(float)
non-linear power-scale meter deflection
coeff signal value
```

Returns deflected value  
void  
memset(*FloatArray*, float, unsigned int)  
lua wrapper to memset()  
void  
mix\_buffers\_no\_gain(*FloatArray*,*FloatArray*, unsigned int)  
void  
mix\_buffers\_with\_gain(*FloatArray*,*FloatArray*, unsigned int, float)  
void  
mmult(*FloatArray*,*FloatArray*, unsigned int)  
matrix multiply multiply every sample of ‘data’ with the corresponding sample at ‘mult’.  
**data** multiplicand  
**mult** multiplicand  
**n\_samples** number of samples in data and mmult  
*LuaTable*(...)  
peaks(*FloatArray*, float&, float&, unsigned int)  
void  
process\_map(*BufferSet*,*ChanMapping*,*ChanMapping*, unsigned int, long,*Data Type*)

## 152.26 ARDOUR:DSP:Biquad

*C*#: ARDOUR::DSP::Biquad

Biquad Filter

Constructor

ARDOUR.DSP.Biquad(double)

Instantiate Biquad Filter

**samplerate** Samplerate

Methods

void

compute(*Type*, double, double, double)

setup filter, compute coefficients

**type** filter type (LowPass, HighPass, etc)

**freq** filter frequency

**Q** filter quality

**gain** filter gain

```

void
configure(double, double, double, double, double)
setup filter, set coefficients directly
float
dB_at_freq(float)
filter transfer function (filter response for spectrum visualization)
freq frequency
Returns gain at given frequency in dB (clamped to -120..+120)
void
reset()
reset filter state
void
run(FloatArray, unsigned int)
process audio data
data pointer to audio-data
n_samples number of samples to process

```

## 152.27 ARDOUR:DSP:DspShm

*C<sub>++</sub>*: ARDOUR::DSP::DspShm

C/C++ Shared Memory

A convenience class representing a C array of float[] or int32\_t[] data values. This is useful for lua scripts to perform DSP operations directly using C/C++ with CPU Hardware acceleration.

Access to this memory area is always 4 byte aligned. The data is interpreted either as float or as int.

This memory area can also be shared between different instances or the same lua plugin (DSP, GUI).

Since memory allocation is not realtime safe it should be allocated during dsp\_init() or dsp\_configure(). The memory is free()ed automatically when the lua instance is destroyed.

Constructor

ARDOUR.DSP.DspShm(unsigned long)

Methods

void

allocate(unsigned long)

[re] allocate memory in host's memory space

**s** size, total number of float or integer elements to store.

int  
atomic\_get\_int(unsigned long)  
atomically read integer at offset  
This involves a memory barrier. This call is intended for buffers which are shared with another instance.  
**off** offset in shared memory region  
Returns value at offset  
void  
atomic\_set\_int(unsigned long, int)  
atomically set integer at offset  
This involves a memory barrier. This call is intended for buffers which are shared with another instance.  
**off** offset in shared memory region  
**val** value to set  
void  
clear()  
clear memory (set to zero)  
*FloatArray*  
to\_float(unsigned long)  
access memory as float array  
**off** offset in shared memory region  
Returns float[]  
*IntArray*  
to\_int(unsigned long)  
access memory as integer array  
**off** offset in shared memory region  
Returns int\_32\_t[]

## 152.28 ARDOUR:DSP:FFTSpectrum

*C<sup>#</sup>*: ARDOUR::DSP::FFTSpectrum

Constructor

ARDOUR.DSP.FFTSpectrum(unsigned int, double)

Methods

void

execute()

process current data in buffer

```
float
freq_at_bin(unsigned int)
float
power_at_bin(unsigned int, float)
query
b the frequency bin 0 .. window_size / 2
norm gain factor (set equal to for 1/f normalization)
Returns signal power at given bin (in dBFS)
void
set_data_hann(FloatArray, unsigned int, unsigned int)
```

## 152.29 ARDOUR:DSP:LowPass

*C++:* ARDOUR::DSP::LowPass

1st order Low Pass filter

Constructor

ARDOUR.DSP.LowPass(double, float)

instantiate a LPF

**samplerate** samplerate

**freq** cut-off frequency

Methods

void

ctrl(*FloatArray*, float, unsigned int)

filter control data

This is useful for parameter smoothing.

**data** pointer to control-data array

**val** target value

**array** length

void

proc(*FloatArray*, unsigned int)

process audio data

**data** pointer to audio-data

**n\_samples** number of samples to process

```
void
reset()
reset filter state
void
set_cutoff(float)
update filter cut-off frequency
freq cut-off frequency
```

## 152.30 ARDOUR:DataType

*C#:* ARDOUR::DataType

A type of Data Ardour is capable of processing.

The majority of this class is dedicated to conversion to and from various other type representations, simple comparison between them, etc. This code is deliberately ‘ugly’ so other code doesn’t have to be.

Constructor

ARDOUR.DataType(std::string)

Methods

*DataType*

audio()

convenience constructor for DataType::AUDIO with managed lifetime

Returns DataType::AUDIO

*DataType*

midi()

convenience constructor for DataType::MIDI with managed lifetime

Returns DataType::MIDI

*DataType*

null()

convenience constructor for DataType::NIL with managed lifetime

Returns DataType::NIL

char\*

to\_string()

Inverse of the from-string constructor

## 152.31 ARDOUR:Delivery

*C<sub>t</sub>*: boost::shared\_ptr< ARDOUR::Delivery >, boost::weak\_ptr< ARDOUR::Delivery >  
is-a: *ARDOUR:IOProcessor*

A mixer strip element (Processor) with 1 or 2 IO elements.

Methods

bool

isnil()

*PannerShell*

panner\_shell()

### 152.31.1 Inherited from ARDOUR:IOProcessor

Methods

*IO*

input()

*ChanCount*

natural\_input\_streams()

*ChanCount*

natural\_output\_streams()

*IO*

output()

### 152.31.2 Inherited from ARDOUR:Processor

Methods

void

activate()

bool

active()

void

deactivate()

std::string

display\_name()

bool

display\_to\_user()

*ChanCount*

input\_streams()

*ChanCount*  
output\_streams()

Cast

*Amp*  
to\_amp()

*Automatable*  
to\_automatable()

*PluginInsert*  
to\_insert()

*IOProcessor*  
to\_ioprocessor()

*PeakMeter*  
to\_meter()

*MonitorProcessor*  
to\_monitorprocessor()

*PeakMeter*  
to\_peakmeter()

*PluginInsert*  
to\_plugininsert()

*SideChain*  
to\_sidechain()

*UnknownProcessor*  
to\_unknownprocessor()

### 152.31.3 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*  
to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.32 ARDOUR:DeviceStatus

*C<sup>#</sup>:* ARDOUR::AudioBackend::DeviceStatus

used to list device names along with whether or not they are currently available.

Data Members

bool

available

std::string

name

## 152.33 ARDOUR:DeviceStatusVector

*C<sup>#</sup>:* std::vector<ARDOUR::AudioBackend::DeviceStatus >

Constructor

ARDOUR.DeviceStatusVector()

ARDOUR.DeviceStatusVector()

Methods

*LuaTable*

add(*LuaTable* {*DeviceStatus*})

*DeviceStatus*

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(*DeviceStatus*)

unsigned long

size()

*LuaTable*

table()

## 152.34 ARDOUR:DoubleBeatsFramesConverter

*C<sub>7</sub>*: ARDOUR::DoubleBeatsFramesConverter

Converter between quarter-note beats and frames. Takes distances in quarter-note beats or frames from some origin (supplied to the constructor in frames), and converts them to the opposite unit, taking tempo changes into account.

Constructor

ARDOUR.DoubleBeatsFramesConverter(*TempoMap*, long)

Methods

double

from(long)

Convert B time to A time (A from B)

long

to(double)

Convert A time to B time (A to B)

## 152.35 ARDOUR:EventList

*C<sub>7</sub>*: std::list<Evoral::ControlEvent\*>

Constructor

ARDOUR.EventList()

Methods

*ControlEvent*

back()

bool

empty()

*ControlEvent*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.36 ARDOUR:FileSource

*C<sub>7</sub>*: boost::shared\_ptr< ARDOUR::FileSource >, boost::weak\_ptr< ARDOUR::FileSource >

is-a: *ARDOUR:Source*

A source associated with a file on disk somewhere

Methods

unsigned short

channel()

float

gain()

bool

isnil()

std::string

origin()

std::string

path()

std::string

take\_id()

bool

within\_session()

### 152.36.1 Inherited from ARDOUR:Source

Methods

std::string

ancestor\_name()

bool

can\_be\_analysed()

bool

destructive()

bool

empty()

bool

has\_been\_analysed()

long

length(long)  
long  
natural\_position()  
long  
timeline\_position()  
long  
timestamp()  
int  
use\_count()  
bool  
used()  
bool  
writable()  
Cast  
*AudioSource*  
to\_audiosource()  
*FileSource*  
to\_filesource()  
*MidiSource*  
to\_midisource()

## 152.36.2 Inherited from ARDOUR:SessionObjectPtr

Methods  
std::string  
name()  
Cast  
*Stateful*  
to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.37 ARDOUR:FluidSynth

*C<sub>++</sub>*: ARDOUR::FluidSynth  
Constructor

---

```

ARDOUR.FluidSynth(float, int)
instantiate a Synth
samplerate samplerate
Methods
bool
load_sf2(std::string)
bool
midi_event(unsigned char*, unsigned long)
void
panic()
unsigned int
program_count()
std::string
program_name(unsigned int)
bool
select_program(unsigned int, unsigned char)
bool
synth(FloatArray,FloatArray, unsigned int)

```

## 152.38 ARDOUR:GainControl

*C++:* boost::shared\_ptr< ARDOUR::GainControl >, boost::weak\_ptr< ARDOUR::GainControl >

is-a: *ARDOUR:SlavableAutomationControl*

A PBD::Controllable with associated automation data (AutomationList)

Methods

```

bool
isnil()

```

### 152.38.1 Inherited from ARDOUR:SlavableAutomationControl

Methods

```

void
add_master(AutomationControl)
void
clear_masters()
int
get_boolean_masters()

```

```
double  
get_masters_value()  
void  
remove_master(AutomationControl)  
bool  
slaved()  
bool  
slaved_to(AutomationControl)
```

## 152.38.2 Inherited from ARDOUR:**AutomationControl**

Methods

*AutomationList*

alist()

*AutoState*

automation\_state()

double

get\_value()

Get the current effective ‘user’ value based on automation state

void

set\_automation\_state(*AutoState*)

void

set\_value(double, *GroupControlDisposition*)

Get and Set ‘internal’ value

All derived classes must implement this.

Basic derived classes will ignore

**group\_override**, but more sophisticated children, notably those that proxy the value setting logic via an object that is aware of group relationships between this control and others, will find it useful.

void

start\_touch(double)

void

stop\_touch(double)

bool

writable()

Cast

*Control*

to\_ctrl()

*SlavableAutomationControl*

to\_slavable()

### 152.38.3 Inherited from PBD:Controllable

Methods

std::string

name()

### 152.38.4 Inherited from PBD:StatefulPtr

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.39 ARDOUR:IO

*C<sub>7</sub>*: boost::shared\_ptr< ARDOUR::IO >, boost::weak\_ptr< ARDOUR::IO >

is-a: *ARDOUR:SessionObjectPtr*

A collection of ports (all input or all output) with connections.

An IO can contain ports of varying types, making routes/inserts/etc with varied combinations of types (e.g. MIDI and audio) possible.

Methods

bool

active()

int

add\_port(std::string, void\*, *Data Type*)

Add a port.

**destination** Name of port to connect new port to.

**src** Source for emitted ConfigurationChanged signal.

**type** Data type of port. Default value (NIL) will use this IO's default type.

*AudioPort*

```
audio(unsigned int)
int
connect(Port, std::string, void*)
int
disconnect(Port, std::string, void*)
int
disconnect_all(void*)
bool
has_port(Port)
bool
isnil()
```

*MidiPort*

```
midi(unsigned int)
```

*ChanCount*

```
n_ports()
```

*Port*

```
nth(unsigned int)
```

```
bool
```

```
physically_connected()
```

*Port*

```
port_by_name(unsigned int)
```

```
int
```

```
remove_port(Port, void*)
```

### 152.39.1 Inherited from ARDOUR:SessionObjectPtr

Methods

```
std::string
```

```
name()
```

Cast

*Stateful*

```
to_stateful()
```

*StatefulDestructible*

```
to_statefuldestructible()
```

## 152.40 ARDOUR:IOProcessor

*C<sub>t</sub>*: boost::shared\_ptr< ARDOUR::IOProcessor >, boost::weak\_ptr< ARDOUR::IOProcessor >  
is-a: *ARDOUR:Processor*

A mixer strip element (Processor) with 1 or 2 IO elements.

Methods

*IO*

input()

bool

isnil()

*ChanCount*

natural\_input\_streams()

*ChanCount*

natural\_output\_streams()

*IO*

output()

### 152.40.1 Inherited from ARDOUR:Processor

Methods

void

activate()

bool

active()

void

deactivate()

std::string

display\_name()

bool

display\_to\_user()

*ChanCount*

input\_streams()

*ChanCount*

output\_streams()

Cast

*Amp*

to\_amp()

*Automatable*

to\_automatable()

*PluginInsert*

to\_insert()

*IOProcessor*

to\_ioprocessor()

*PeakMeter*

to\_meter()

*MonitorProcessor*

to\_monitorprocessor()

*PeakMeter*

to\_peakmeter()

*PluginInsert*

to\_plugininsert()

*SideChain*

to\_sidechain()

*UnknownProcessor*

to\_unknownprocessor()

## 152.40.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.41 ARDOUR:InterThreadInfo

*C<sub>7</sub>*: ARDOUR::InterThreadInfo

Constructor

ARDOUR.InterThreadInfo()

Data Members

bool  
done  
float  
progress

## 152.42 ARDOUR:Location

*C++:* ARDOUR::Location

is-a: *PBD:StatefulDestructible*

Location on Timeline - abstract representation for Markers, Loop/Punch Ranges, CD-Markers etc.

Methods

long  
end()  
*Flags*  
flags()  
bool  
is\_auto\_loop()  
bool  
is\_auto\_punch()  
bool  
is\_cd\_marker()  
bool  
is\_hidden()  
bool  
is\_mark()  
bool  
is\_range\_marker()  
bool  
is\_session\_range()

long  
length()  
void  
lock()  
bool  
locked()  
bool

matches(*Flags*)  
int  
move\_to(long, unsigned int)  
std::string  
name()  
int  
set\_end(long, bool, bool, unsigned int)  
Set end position.  
**force** true to force setting, even if the given new end is before the current start.  
**allow\_beat\_recompute** True to recompute BEAT end time from the new given end time.  
**s** New end.  
int  
set\_length(long, long, bool, unsigned int)  
int  
set\_start(long, bool, bool, unsigned int)  
Set start position.  
**s** New start.  
**force** true to force setting, even if the given new start is after the current end.  
**allow\_beat\_recompute** True to recompute BEAT start time from the new given start time.  
long  
start()  
void  
unlock()

### 152.42.1 Inherited from PBD:Stateful

Methods  
void  
clear\_changes()  
Forget about any changes to this object's properties  
*ID*  
id()  
*OwnedPropertyList*  
properties()

## 152.43 ARDOUR:LocationList

*C<sub>++</sub>*: std::list<ARDOUR::Location\*>

Constructor

ARDOUR.LocationList()

Methods

*Location*

back()

bool

empty()

*Location*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.44 ARDOUR:Locations

*C<sub>++</sub>*: ARDOUR::Locations

is-a: *PBD:StatefulDestructible*

A collection of session locations including unique dedicated locations (loop, punch, etc)

Methods

*Location*

auto\_loop\_location()

*Location*

auto\_punch\_location()

*LuaTable*(...)

find\_all\_between(long, long, *LocationList*&, *Flags*)

long

first\_mark\_after(long, bool)

*Location*

first\_mark\_at(long, long)

long

first\_mark\_before(long, bool)

*LocationList*

list()

*LuaTable*(...)

marks\_either\_side(long, long&, long&)

Look for the ‘marks’ (either locations which are marks, or start/end points of range markers) either side of a frame. Note that if frame is exactly on a ‘mark’, that mark will not be considered for returning as before/after.

**frame** Frame to look for.

**before** Filled in with the position of the last ‘mark’ before ‘frame’ (or max\_framepos if none exists)

**after** Filled in with the position of the next ‘mark’ after ‘frame’ (or max\_framepos if none exists)

void

remove(*Location*)

*Location*

session\_range\_location()

### 152.44.1 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object’s properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.45 ARDOUR.LuaAPI

Methods

...

build\_filename(–lua–)

Creates a filename from a series of elements using the correct separator for filenames.

No attempt is made to force the resulting filename to be an absolute path. If the first element is a relative path, the result will be a relative path.

...

`color_to_rgba(-lua-)`

A convenience function to expand RGBA parameters from an integer

convert a `Canvas::Color` (`uint32_t 0xRRGGBBAA`) into double RGBA values which can be passed as parameters to `Cairo::Context::set_source_rgba`

Example:

```
local r, g, b, a = ARDOUR.LuaAPI.color_to_rgba (0x88aa44ff)
cairo_ctx:set_source_rgba (ARDOUR.LuaAPI.color_to_rgba (0x11336699))
```

Returns 4 parameters: red, green, blue, alpha (in range 0..1)

*LuaTable*(float, ...)

`get_plugin_insert_param(PluginInsert, unsigned int, bool&)`

get a plugin control parameter value

**which** control port to query (starting at 0, including ports of type input and output)

**ok** boolean variable contains true or false after call returned. to be checked by caller before using value.

**proc** `Plugin-Insert`

Returns value

*LuaTable*(float, ...)

`get_processor_param(Processor, unsigned int, bool&)`

get a plugin control parameter value

**proc** `Plugin-Processor`

**which** control port to set (starting at 0, including ports of type input and output))

**ok** boolean variable contains true or false after call returned. to be checked by caller before using value.

Returns value

...

`hsla_to_rgba(-lua-)`

A convenience function for colorspace HSL to RGB conversion. All ranges are 0..1

Example:

```
local r, g, b, a = ARDOUR.LuaAPI.hsla_to_rgba (hue, saturation, luminosity, alpha)
```

Returns 4 parameters: red, green, blue, alpha (in range 0..1)

`long`

`monotonic_time()`

*Processor*

`new_luaproc(Session, std::string)`

create a new Lua Processor (Plugin)

**s** Session Handle

**p** Identifier or Name of the Processor

Returns Processor object (may be nil)

*NotePtr*

`new_noteptr(unsigned char,Beats,Beats, unsigned char, unsigned char)`

*Processor*

`new_plugin(Session, std::string,PluginType, std::string)`

create a new Plugin Instance

**s** Session Handle

**id** Plugin Name, ID or URI

**type** Plugin Type

Returns Processor or nil

*PluginInfo*

`new_plugin_info(std::string,PluginType)`

search a Plugin

**id** Plugin Name, ID or URI

**type** Plugin Type

Returns PluginInfo or nil if not found

*Processor*

`nil_proc()`

*NotePtrList*

`note_list(MidiModel)`

...

`plugin_automation(-lua-)`

A convenience function to get a Automation Lists and ParamaterDescriptor for a given plugin control.

This is equivalent to the following lua code

```
function (processor, param_id)
    local plugininsert = processor:to_insert ()
    local plugin = plugininsert:plugin(0)
    local _, t = plugin:get_parameter_descriptor(param_id, ARDOUR.ParameterDescriptor ())
    local ctrl = Evoral.Parameter (ARDOUR.AutomationType.PluginAutomation, 0, param_id)
    local ac = pi:automation_control (ctrl, false)
    local acl = ac:alist()
    return ac:alist(), ac:to_ctrl():list(), t[2]
end
```

Example usage: get the third input parameter of first plugin on the given route (Ardour starts counting at zero).

```
local al, cl, pd = ARDOUR.LuaAPI.plugin_automation (route:nth_plugin (0), 3)
```

Returns 3 parameters: AutomationList, ControlList, ParamaterDescriptor

bool

```
reset_processor_to_default(Processor)
reset a processor to its default values (only works for plugins )

This is a wrapper which looks up the Processor by plugin-insert.

proc Plugin-Insert
Returns true on success, false when the processor is not a plugin

...
sample_to_timecode(–lua–)
Generic conversion from audio sample count to timecode. (TimecodeType, sample-rate, sample-pos)

bool
set_plugin_insert_param(PluginInsert, unsigned int, float)
set a plugin control-input parameter value

This is a wrapper around set_processor_param which looks up the Processor by plugin-insert.

which control-input to set (starting at 0)

proc Plugin-Insert
value value to set
Returns true on success, false on error or out-of-bounds value

bool
set_processor_param(Processor, unsigned int, float)
set a plugin control-input parameter value

proc Plugin-Processor
which control-input to set (starting at 0)
value value to set
Returns true on success, false on error or out-of-bounds value

...
timecode_to_sample(–lua–)
Generic conversion from timecode to audio sample count. (TimecodeType, sample-rate, hh, mm, ss, ff)

void
usleep(unsigned long)
```

## 152.46 ARDOUR:LuaAPI:Vamp

*C#:* ARDOUR::LuaAPI::Vamp

Constructor

ARDOUR.LuaAPI.Vamp(std::string, float)

Methods

int  
analyze(*Readable*, unsigned int, *Lua-Function*)  
high-level abstraction to process a single channel of the given *Readable*.  
If the plugin is not yet initialized, initialize() is called.  
if is not nil, it is called with the immediate Vamp::Plugin::Features on every process call.  
**r** *readable*  
**channel** channel to process  
**fn** *lua* callback function  
Returns 0 on success  
bool  
initialize()  
initialize the plugin for use with analyze().  
This is equivalent to plugin():initialise (1, ssiz, bsiz) and prepares a plugin for analyze. (by preferred step and block sizes are used. if the plugin does not specify them or they're larger than 8K, both are set to 1024)  
Manual initialization is only required to set plugin-parameters which depend on prior initialization of the plugin.

```
vamp:reset ()  
vamp:initialize ()  
vamp:plugin():setParameter (0, 1.5, nil)  
vamp:analyze (r, 0)
```

*StringVector*  
list\_plugins()  
*Plugin*  
plugin()  
*FeatureSet*  
process(*FloatArrayVector*, *RealTime*)  
process given array of audio-samples.  
This is a lua-binding for vamp:plugin():process ()  
**d** audio-data, the vector must match the configured channel count and hold a complete buffer for every channel as set during plugin():initialise()  
**rt** timestamp matching the provided buffer.  
Returns features extracted from that data (if the plugin is causal)  
void  
reset()  
call plugin():reset() and clear intialization flag

## 152.47 ARDOUR:LuaOSC:Address

*C<sup>†</sup>*: ARDOUR::LuaOSC::Address

OSC transmitter

A Class to send OSC messages.

Constructor

ARDOUR.LuaOSC.Address(std::string)

Construct a new OSC transmitter object

**uri** the destination uri e.g. “osc.udp://localhost:7890”

Methods

...

send(-lua-)

Transmit an OSC message

Path (string) and type (string) must always be given. The number of following args must match the type.  
Supported types are:

‘i’: integer (lua number)

‘f’: float (lua number)

‘d’: double (lua number)

‘h’: 64bit integer (lua number)

‘s’: string (lua string)

‘c’: character (lua string)

‘T’: boolean (lua bool) – this is not implicitly True, a lua true/false must be given

‘F’: boolean (lua bool) – this is not implicitly False, a lua true/false must be given

**lua**: lua arguments: path, types, ...

Returns boolean true if successful, false on error.

## 152.48 ARDOUR:LuaProc

*C<sup>†</sup>*: boost::shared\_ptr< ARDOUR::LuaProc >, boost::weak\_ptr< ARDOUR::LuaProc >

is-a: *ARDOUR:Plugin*

A plugin is an external module (usually 3rd party provided) loaded into Ardour for the purpose of digital signal processing.

This class provides an abstraction for methods provided by all supported plugin standards such as presets, name, parameters etc.

Plugins are not used directly in Ardour but always wrapped by a PluginInsert.

Methods

bool

isnil()  
*DspShm*  
shmem()  
*LuaTableRef*  
table()

### 152.48.1 Inherited from ARDOUR:Plugin

Methods

*IOPortDescription*

describe\_io\_port(*Data Type*, bool, unsigned int)  
std::string  
get\_docs()  
*PluginInfo*  
get\_info()  
*LuaTable*(int, ...)  
get\_parameter\_descriptor(unsigned int, *ParameterDescriptor*&)  
std::string  
get\_parameter\_docs(unsigned int)  
char\*  
label()  
bool  
load\_preset(*PresetRecord*)  
Set parameters using a preset  
char\*  
maker()  
char\*  
name()  
*LuaTable*(unsigned int, ...)  
nth\_parameter(unsigned int, bool&)  
unsigned int  
parameter\_count()  
bool  
parameter\_is\_audio(unsigned int)  
bool  
parameter\_is\_control(unsigned int)  
bool

```
parameter_is_input(unsigned int)
bool
parameter_is_output(unsigned int)
std::string
parameter_label(unsigned int)
PresetRecord
preset_by_label(std::string)
PresetRecord
preset_by_uri(std::string)
std::string
unique_id()
Cast
LuaProc
to_luaproc()
```

## 152.48.2 Inherited from PBD:StatefulPtr

Methods

```
void
clear_changes()
Forget about any changes to this object's properties
ID
id()
OwnedPropertyList
properties()
```

## 152.49 ARDOUR:LuaTableRef

*C<sub>7</sub>*: ARDOUR::LuaTableRef

Methods

...

get(-lua-)

...

set(-lua-)

## 152.50 ARDOUR:Meter

*C<sub>7</sub>*: ARDOUR::Meter

Meter, or time signature (beats per bar, and which note type is a beat).

Constructor

ARDOUR.Meter(double, double)

Methods

double

divisions\_per\_bar()

double

frames\_per\_bar(*Tempo*, long)

double

frames\_per\_grid(*Tempo*, long)

double

note\_divisor()

## 152.51 ARDOUR:MeterSection

*C<sub>7</sub>*: ARDOUR::MeterSection

is-a: *ARDOUR:MetricSection*

A section of timeline with a certain Meter.

Methods

void

set\_beat(double)

void

set\_pulse(double)

Cast

*Meter*

to\_meter()

### 152.51.1 Inherited from ARDOUR:MetricSection

Methods

double

pulse()

## 152.52 ARDOUR:MetricSection

*C<sup>†</sup>*: ARDOUR::MetricSection

A section of timeline with a certain Tempo or Meter.

Methods

double

pulse()

void

set\_pulse(double)

## 152.53 ARDOUR:MidiBuffer

*C<sup>†</sup>*: ARDOUR::MidiBuffer

Buffer containing 8-bit unsigned char (MIDI) data.

Methods

void

copy(*MidiBuffer*)

bool

empty()

bool

push\_back(long, unsigned long, unsigned char\*)

bool

push\_event(*Event*)

void

resize(unsigned long)

Reallocate the buffer used internally to handle at least *size\_t* units of data.

The buffer is not silent after this operation. the *capacity* argument passed to the constructor must have been non-zero.

void

silence(long, long)

Clear (e.g. zero, or empty) buffer

unsigned long

size()

...

table(–lua–)

## 152.54 ARDOUR:MidiModel

*C $\ddagger$ :* boost::shared\_ptr< ARDOUR::MidiModel >, boost::weak\_ptr< ARDOUR::MidiModel >

is-a: *ARDOUR:AutomatableSequence*

This is a higher level (than MidiBuffer) model of MIDI data, with separate representations for notes (instead of just unassociated note on/off events) and controller data. Controller data is represented as part of the Automatable base (i.e. in a map of AutomationList, keyed by Parameter). Because of this MIDI controllers and automatable controllers/widgets/etc are easily interchangeable.

Methods

void

*apply\_command(Session, Command)*

bool

*isnil()*

*NoteDiffCommand*

*new\_note\_diff\_command(std::string)*

Start a new NoteDiff command.

This has no side-effects on the model or Session, the returned command can be held on to for as long as the caller wishes, or discarded without formality, until *apply\_command* is called and ownership is taken.

### 152.54.1 Inherited from ARDOUR:AutomatableSequence

Cast

*Sequence*

*to\_sequence()*

### 152.54.2 Inherited from ARDOUR:Automatable

Methods

*AutomationControl*

*automation\_control(Parameter, bool)*

Cast

*Slavable*

*to\_slavable()*

## 152.55 ARDOUR:MidiModel:DiffCommand

*C $\ddagger$ :* ARDOUR::MidiModel::DiffCommand

is-a: *PBD:Command*

Base class for Undo/Redo commands and changesets

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

### 152.55.1 Inherited from PBD:Command

Methods

std::string  
name()  
void  
set\_name(std::string)

### 152.55.2 Inherited from PBD:Stateful

Methods

void  
clear\_changes()  
Forget about any changes to this object's properties  
*ID*  
id()  
*OwnedPropertyList*  
properties()

## 152.56 ARDOUR:MidiModel:NoteDiffCommand

*C<sub>7</sub>*: ARDOUR::MidiModel::NoteDiffCommand

is-a: *ARDOUR:MidiModel:DiffCommand*

Base class for Undo/Redo commands and changesets

Methods

void  
add(*NotePtr*)  
void  
remove(*NotePtr*)

### 152.56.1 Inherited from PBD:Command

Methods

std::string  
name()  
void

set\_name(std::string)

### 152.56.2 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.57 ARDOUR:MidiPlaylist

*C++:* boost::shared\_ptr< ARDOUR::MidiPlaylist >, boost::weak\_ptr< ARDOUR::MidiPlaylist >

is-a: *ARDOUR:Playlist*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

bool

isnil()

void

set\_note\_mode(*NoteMode*)

### 152.57.1 Inherited from ARDOUR:Playlist

Methods

void

add\_region(*Region*, long, float, bool, int, double, bool)

Note: this calls set\_layer(..., DBL\_MAX) so it will reset the layering index of region

*Region*

combine(*RegionList*)

unsigned int

count\_regions\_at(long)

*Playlist*

cut(*AudioRangeList*, bool)

*DataType*

```
data_type()
void
duplicate(Region, long, long, float)
gap from the beginning of the region to the next beginning
void
duplicate_range(AudioRange&, float)
void
duplicate_until(Region, long, long, long)
gap from the beginning of the region to the next beginning
end the first frame that does not contain a duplicated frame
Region
find_next_region(long,RegionPoint, int)
long
find_next_region_boundary(long, int)
long
find_next_transient(long, int)
void
lower_region(Region)
void
lower_region_to_bottom(Region)
unsigned int
n_regions()
void
raise_region(Region)
void
raise_region_to_top(Region)
Region
region_by_id(ID)
RegionListPtr
region_list()
RegionListPtr
regions_at(long)
RegionListPtr
regions_touched(long, long)
start Range start.
end Range end.
```

Returns regions which have some part within this range.

*RegionListPtr*

regions\_with\_end\_within(*Range*)

*RegionListPtr*

regions\_with\_start\_within(*Range*)

void

remove\_region(*Region*)

void

split(long)

void

split\_region(*Region, MusicFrame*)

*Region*

top\_region\_at(long)

*Region*

top\_unmuted\_region\_at(long)

void

uncombine(*Region*)

Cast

*AudioPlaylist*

to\_audioplaylist()

*MidiPlaylist*

to\_midiplaylist()

## 152.57.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.58 ARDOUR:MidiPort

*C++:* boost::shared\_ptr< ARDOUR::MidiPort >, boost::weak\_ptr< ARDOUR::MidiPort >

is-a: *ARDOUR:Port*

Methods

*MidiBuffer*

get\_midi\_buffer(unsigned int)

bool

input\_active()

bool

isnil()

void

set\_input\_active(bool)

### 152.58.1 Inherited from ARDOUR:Port

Methods

int

connect(std::string)

bool

connected()

Returns true if this port is connected to anything

bool

connected\_to(std::string)

o Port name

Returns true if this port is connected to o, otherwise false.

int

disconnect(std::string)

int

disconnect\_all()

std::string

name()

Returns Port short name

std::string

pretty\_name(bool)

Returns Port human readable name

bool

receives\_input()

Returns true if this Port receives input, otherwise false

bool

sends\_output()

Returns true if this Port sends output, otherwise false

Cast

*AudioPort*

to\_audioport()

*MidiPort*

to\_midiport()

## 152.59 ARDOUR:MidiRegion

*C<sub>++</sub>*: boost::shared\_ptr< ARDOUR::MidiRegion >, boost::weak\_ptr< ARDOUR::MidiRegion >

is-a: *ARDOUR:Region*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

bool

do\_export(std::string)

Export the MIDI data of the MidiRegion to a new MIDI file (SMF).

bool

isnil()

double

length\_beats()

*MidiSource*

midi\_source(unsigned int)

*MidiModel*

model()

double

start\_beats()

### 152.59.1 Inherited from ARDOUR:Region

Methods

bool

at\_natural\_position()

bool  
automatic()  
bool  
can\_move()  
bool  
captured()  
void  
clear\_sync\_position()

*Control*

control(*Parameter*, bool)  
bool  
covers(long)  
void  
cut\_end(long, int)  
void  
cut\_front(long, int)

*Data Type*

data\_type()  
bool  
external()  
bool  
has\_transients()  
bool  
hidden()  
bool  
import()  
bool  
is\_compound()  
unsigned int  
layer()  
long  
length()  
bool  
locked()  
void  
lower()

```
void
lower_to_bottom()

StringVector
master_source_names()

SourceList
master_sources()
void
move_start(long, int)
void
move_to_natural_position()
bool
muted()
unsigned int
n_channels()
void
nudge_position(long)
bool
opaque()
long
position()
```

How the region parameters play together:

POSITION: first frame of the region along the timeline  
START: first frame of the region within its source(s)  
LENGTH: number of frames the region represents

```
bool
position_locked()
double
quarter_note()
void
raise()
void
raise_to_top()
void
set_hidden(bool)
void
set_initial_position(long)
```

A gui may need to create a region, then place it in an initial position determined by the user. When this takes place within one gui operation, we have to reset `_last_position` to prevent an implied move.

```
void
set_length(long, int)
void
set_locked(bool)
void
set_muted(bool)
void
set_opaque(bool)
void
set_position(long, int)
void
set_position_locked(bool)
void
set_start(long)
void
set_sync_position(long)
Set the region's sync point.
absolute_pos Session time.
void
set_video_locked(bool)
float
shift()
Source
source(unsigned int)
long
start()
float
stretch()
bool
sync_marked()
LuaTable(long, ...)
sync_offset(int&)
long
sync_position()
Returns Sync position in session time
Int64List
```

```
transients()
void
trim_end(long, int)
void
trim_front(long, int)
void
trim_to(long, long, int)
bool
video_locked()
bool
whole_file()
Cast
AudioRegion
to_audioregion()
MidiRegion
to_midiregion()
Readable
to_readable()
```

## 152.59.2 Inherited from ARDOUR:SessionObjectPtr

```
Methods
std::string
name()
Cast
Stateful
to_stateful()
StatefulDestructible
to_statefuldestructible()
```

## 152.60 ARDOUR:MidiSource

```
C#: boost::shared_ptr< ARDOUR::MidiSource >, boost::weak_ptr< ARDOUR::MidiSource >
is-a: ARDOUR:Source
Source for MIDI data
Methods
bool
```

empty()  
bool  
isnil()  
long  
length(long)  
*MidiModel*  
model()

### 152.60.1 Inherited from ARDOUR:Source

Methods  
std::string  
ancestor\_name()  
bool  
can\_be\_analysed()  
bool  
destructive()  
bool  
has\_been\_analysed()  
long  
natural\_position()  
long  
timeline\_position()  
long  
timestamp()  
int  
use\_count()  
bool  
used()  
bool  
writable()  
Cast  
*AudioSource*  
to\_audiosource()  
*FileSource*  
to\_filesource()  
*MidiSource*

to\_midisource()

### 152.60.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.61 ARDOUR:MidiTrack

*Cf:* boost::shared\_ptr< ARDOUR::MidiTrack >, boost::weak\_ptr< ARDOUR::MidiTrack >

is-a: *ARDOUR:Track*

A track is an route (bus) with a recordable diskstream and related objects relevant to tracking, playback and editing.

Specifically a track has regions and playlist objects.

Methods

bool

isnil()

### 152.61.1 Inherited from ARDOUR:Track

Methods

*Region*

bounce(*InterThreadInfo&*)

bounce track from session start to session end to new region

**itt** asynchronous progress report and cancel

Returns a new audio region (or nil in case of error)

*Region*

bounce\_range(long, long, *InterThreadInfo&*, *Processor*, bool)

Bounce the given range to a new audio region.

**start** start time (in samples)

**end** end time (in samples)

**itt** asynchronous progress report and cancel

**endpoint** the processor to tap the signal off (or nil for the top)

**include\_endpoint** include the given processor in the bounced audio.

Returns a new audio region (or nil in case of error)

bool

bounceable(*Processor*, bool)

Test if the track can be bounced with the given settings. If sends/inserts/returns are present in the signal path or the given track has no audio outputs bouncing is not possible.

**endpoint** the processor to tap the signal off (or nil for the top)

**include\_endpoint** include the given processor in the bounced audio.

Returns true if the track can be bounced, or false otherwise.

bool

can\_record()

*Playlist*

playlist()

bool

set\_name(std::string)

Cast

*AudioTrack*

to\_audio\_track()

*MidiTrack*

to\_midi\_track()

## 152.61.2 Inherited from ARDOUR:Route

Methods

bool

active()

int

add\_processor\_by\_index(*Processor*, int, *ProcessorStreams*, bool)

Add a processor to a route such that it ends up with a given index into the visible processors.

**index** Index to add the processor at, or -1 to add at the end of the list.

Returns 0 on success, non-0 on failure.

bool

add\_sidechain(*Processor*)

*Amp*

amp()

std::string

comment()  
bool  
customize\_plugin\_insert(*Processor*, unsigned int, *ChanCount*, *ChanCount*)  
enable custom plugin-insert configuration  
**proc** Processor to customize  
**count** number of plugin instances to use (if zero, reset to default)  
**outs** output port customization  
**sinks** input pins for variable-I/O plugins  
Returns true if successful  
*IO*  
input()  
*Delivery*  
main\_outs()  
the signal processor at end of the processing chain which produces output  
bool  
muted()  
*ChanCount*  
n\_inputs()  
*ChanCount*  
n\_outputs()  
*Processor*  
nth\_plugin(unsigned int)  
*Processor*  
nth\_processor(unsigned int)  
*Processor*  
nth\_send(unsigned int)  
*IO*  
output()  
*PannerShell*  
panner\_shell()  
*PeakMeter*  
peak\_meter()  
\*\*\*\*\* Pure interface begins  
here\*\*\*\*\*  
int  
remove\_processor(*Processor*, *ProcessorStreams*, bool)

remove plugin/processor

**proc** processor to remove

**err** error report (index where removal failed, channel-count why it failed) may be nil

**need\_process\_lock** if locking is required (set to true, unless called from RT context with lock)

Returns 0 on success

int

remove\_processors(*ProcessorList*,*ProcessorStreams*)

bool

remove\_sidechain(*Processor*)

int

reorder\_processors(*ProcessorList*,*ProcessorStreams*)

int

replace\_processor(*Processor*,*Processor*,*ProcessorStreams*)

replace plugin/processor with another

**old** processor to remove

**sub** processor to substitute the old one with

**err** error report (index where removal failed, channel-count why it failed) may be nil

Returns 0 on success

bool

reset\_plugin\_insert(*Processor*)

reset plugin-insert configuration to default, disable customizations.

This is equivalent to calling

```
customize_plugin_insert (proc, 0, unused)
```

**proc** Processor to reset

Returns true if successful

void

set\_active(bool, void\*)

void

set\_comment(std::string, void\*)

void

set\_meter\_point(*MeterPoint*, bool)

bool

set\_strict\_io(bool)

bool

soloed()

bool

strict\_io()

*Processor*

the\_instrument()

Return the first processor that accepts has at least one MIDI input and at least one audio output. In the vast majority of cases, this will be “the instrument”. This does not preclude other MIDI->audio processors later in the processing chain, but that would be a special case not covered by this utility function.

*Amp*

trim()

*Cast*

*Automatable*

to\_automatable()

*Slavable*

to\_slavable()

*Track*

to\_track()

### 152.61.3 Inherited from ARDOUR:Stripable

Methods

*AutomationControl*

comp\_enable\_control()

*AutomationControl*

comp\_makeup\_control()

*AutomationControl*

comp\_mode\_control()

std::string

comp\_mode\_name(unsigned int)

*ReadOnlyControl*

comp\_redux\_control()

*AutomationControl*

comp\_speed\_control()

std::string

comp\_speed\_name(unsigned int)

*AutomationControl*

comp\_threshold\_control()

unsigned int

eq\_band\_cnt()

std::string

eq\_band\_name(unsigned int)

*AutomationControl*

eq\_enable\_control()

*AutomationControl*

eq\_freq\_control(unsigned int)

*AutomationControl*

eq\_gain\_control(unsigned int)

*AutomationControl*

eq\_q\_control(unsigned int)

*AutomationControl*

eq\_shape\_control(unsigned int)

*AutomationControl*

filter\_enable\_controllable(bool)

*AutomationControl*

filter\_freq\_controllable(bool)

*AutomationControl*

filter\_slope\_controllable(bool)

*GainControl*

gain\_control()

bool

is\_auditioner()

bool

is\_hidden()

bool

is\_master()

bool

is\_monitor()

bool

is\_selected()

*AutomationControl*

master\_send\_enable\_control()

*MonitorProcessor*

monitor\_control()

*MuteControl*

mute\_control()

*AutomationControl*

```
pan_azimuth_control()  
AutomationControl  
pan_elevation_control()  
AutomationControl  
pan_frontback_control()  
AutomationControl  
pan_lfe_control()  
AutomationControl  
pan_width_control()  
PhaseControl  
phase_control()  
PresentationInfo  
presentation_info_ptr()  
AutomationControl  
rec_enable_control()  
AutomationControl  
rec_safe_control()  
AutomationControl  
send_enable_control(unsigned int)  
AutomationControl  
send_level_control(unsigned int)  
std::string  
send_name(unsigned int)  
void  
set_presentation_order(unsigned int)  
SoloControl  
solo_control()  
SoloIsolateControl  
solo_isolate_control()  
SoloSafeControl  
solo_safe_control()  
GainControl  
trim_control()  
Cast  
Route  
to_route()
```

*VCA*

`to_vca()`

#### 152.61.4 Inherited from ARDOUR:SessionObjectPtr

Methods

`std::string`

`name()`

Cast

*Stateful*

`to_stateful()`

*StatefulDestructible*

`to_statefuldestructible()`

## 152.62 ARDOUR:MidiTrackList

*C++:* `std::list<boost::shared_ptr<ARDOUR::MidiTrack> >`

Constructor

`ARDOUR.MidiTrackList()`

Methods

*LuaTable*

`add(LuaTable {MidiTrack})`

*MidiTrack*

`back()`

`bool`

`empty()`

*MidiTrack*

`front()`

*LuaIter*

`iter()`

`void`

`push_back(MidiTrack)`

`void`

`reverse()`

`unsigned long`

`size()`

*LuaTable*  
table()  
void  
unique()

## 152.63 ARDOUR:MonitorProcessor

*C++:* boost::shared\_ptr< ARDOUR::MonitorProcessor >, boost::weak\_ptr< ARDOUR::MonitorProcessor >

is-a: *ARDOUR:Processor*

A mixer strip element - plugin, send, meter, etc

Methods

*Controllable*

channel\_cut\_control(unsigned int)

*Controllable*

channel\_dim\_control(unsigned int)

*Controllable*

channel\_polarity\_control(unsigned int)

*Controllable*

channel\_solo\_control(unsigned int)

bool

cut(unsigned int)

bool

cut\_all()

*Controllable*

cut\_control()

bool

dim\_all()

*Controllable*

dim\_control()

float

dim\_level()

*Controllable*

dim\_level\_control()

bool

dimmed(unsigned int)

bool

```
inverted(unsigned int)
bool
isnil()
bool
monitor_active()
bool
mono()

Controllable
mono_control()
void
set_cut(unsigned int, bool)
void
set_cut_all(bool)
void
set_dim(unsigned int, bool)
void
set_dim_all(bool)
void
set_mono(bool)
void
set_polarity(unsigned int, bool)
void
set_solo(unsigned int, bool)

Controllable
solo_boost_control()
float
solo_boost_level()
bool
soloed(unsigned int)
```

### 152.63.1 Inherited from ARDOUR:Processor

Methods

```
void
activate()
bool
active()
```

```
void
deactivate()
std::string
display_name()
bool
display_to_user()
ChanCount
input_streams()
ChanCount
output_streams()
Cast
Amp
to_amp()
Automatable
to_automatable()
PluginInsert
to_insert()
IOProcessor
to_ioprocessor()
PeakMeter
to_meter()
MonitorProcessor
to_monitorprocessor()
PeakMeter
to_peakmeter()
PluginInsert
to_plugininsert()
SideChain
to_sidechain()
UnknownProcessor
to_unknownprocessor()
```

### 152.63.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.64 ARDOUR:MusicFrame

*C<sub>7</sub>*: ARDOUR::MusicFrame

Constructor

ARDOUR.MusicFrame(long, int)

Methods

void

set(long, int)

Data Members

int

division

long

frame

## 152.65 ARDOUR:MuteControl

*C<sub>7</sub>*: boost::shared\_ptr< ARDOUR::MuteControl >, boost::weak\_ptr< ARDOUR::MuteControl >

is-a: *ARDOUR:SlavableAutomationControl*

A PBD::Controllable with associated automation data (AutomationList)

Methods

bool

isnil()

bool

muted()

bool

muted\_by\_self()

### 152.65.1 Inherited from ARDOUR:SlavableAutomationControl

Methods

void  
add\_master(*AutomationControl*)  
void  
clear\_masters()  
int  
get\_boolean\_masters()  
double  
get\_masters\_value()  
void  
remove\_master(*AutomationControl*)  
bool  
slaved()  
bool  
slaved\_to(*AutomationControl*)

### 152.65.2 Inherited from ARDOUR:AutomationControl

Methods

*AutomationList*  
alist()  
*AutoState*  
automation\_state()  
double  
get\_value()  
Get the current effective ‘user’ value based on automation state  
void  
set\_automation\_state(*AutoState*)  
void  
set\_value(double, *GroupControlDisposition*)  
Get and Set ‘internal’ value  
All derived classes must implement this.  
Basic derived classes will ignore  
**group\_override**, but more sophisticated children, notably those that proxy the value setting logic via an object that is aware of group relationships between this control and others, will find it useful.

void  
start\_touch(double)  
void  
stop\_touch(double)  
bool  
writable()  
Cast  
*Control*  
to\_ctrl()  
*SlavableAutomationControl*  
to\_slavable()

### 152.65.3 Inherited from PBD:Controllable

Methods  
std::string  
name()

### 152.65.4 Inherited from PBD:StatefulPtr

Methods  
void  
clear\_changes()  
Forget about any changes to this object's properties  
*ID*  
id()  
*OwnedPropertyList*  
properties()

## 152.66 ARDOUR:NotePtrList

*C<sub>7</sub>*: std::list<boost::shared\_ptr<Evoral::Note<Evoral::Beats>>>>  
Constructor

ARDOUR.NotePtrList()  
Methods  
*LuaTable*  
add(LuaTable {Beats})

*NotePtr*  
back()  
bool  
empty()  
*NotePtr*  
front()  
*LuaIter*  
iter()  
void  
push\_back(*NotePtr*)  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()  
void  
unique()

## 152.67 ARDOUR:OwnedPropertyList

*C++:* PBD::OwnedPropertyList

is-a: *ARDOUR:PropertyList*

Persistent Property List

A variant of *PropertyList* that does not delete its property list in its destructor. Objects with their own Properties store them in an *OwnedPropertyList* to avoid having them deleted at the wrong time.

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.68 ARDOUR:PannerShell

*C++:* boost::shared\_ptr< ARDOUR::PannerShell >, boost::weak\_ptr< ARDOUR::PannerShell >

is-a: *ARDOUR:SessionObjectPtr*

Class to manage panning by instantiating and controlling an appropriate *Panner* object for a given in/out configuration.

Methods

bool

bypassed()  
bool  
isnil()  
void  
set\_bypassed(bool)

### 152.68.1 Inherited from ARDOUR:SessionObjectPtr

Methods  
std::string  
name()  
Cast  
*Stateful*  
to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.69 ARDOUR:ParameterDescriptor

*C#:* ARDOUR::ParameterDescriptor  
is-a: *Evoral:ParameterDescriptor*  
Descriptor of a parameter or control.  
Essentially a union of LADSPA, VST and LV2 info.  
Constructor

ARDOUR.ParameterDescriptor()  
Methods  
std::string  
midi\_note\_name(unsigned char, bool)  
Data Members  
std::string  
label

### 152.69.1 Inherited from Evoral:ParameterDescriptor

Constructor

Evoral.ParameterDescriptor()

Data Members

bool  
logarithmic  
True for log-scale parameters

float  
lower  
Minimum value (in Hz, for frequencies)

float  
normal  
Default value

bool  
toggled  
True iff parameter is boolean

float  
upper  
Maximum value (in Hz, for frequencies)

## 152.70 ARDOUR:PeakMeter

*C<sub>++</sub>*: boost::shared\_ptr< ARDOUR::PeakMeter >, boost::weak\_ptr< ARDOUR::PeakMeter >  
is-a: *ARDOUR:Processor*

Meters peaks on the input and stores them for access.

Methods

bool  
isnil()  
float  
meter\_level(unsigned int, *MeterType*)  
void  
reset\_max()  
void  
set\_type(*MeterType*)

### 152.70.1 Inherited from ARDOUR:Processor

Methods

void  
activate()

bool  
active()  
void  
deactivate()  
std::string  
display\_name()  
bool  
display\_to\_user()  
*ChanCount*  
input\_streams()  
*ChanCount*  
output\_streams()  
Cast  
*Amp*  
to\_amp()  
*Automatable*  
to\_automatable()  
*PluginInsert*  
to\_insert()  
*IOProcessor*  
to\_ioprocessor()  
*PeakMeter*  
to\_meter()  
*MonitorProcessor*  
to\_monitorprocessor()  
*PeakMeter*  
to\_peakmeter()  
*PluginInsert*  
to\_plugininsert()  
*SideChain*  
to\_sidechain()  
*UnknownProcessor*  
to\_unknownprocessor()

### 152.70.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.71 ARDOUR:PhaseControl

*C#:* boost::shared\_ptr< ARDOUR::PhaseControl >, boost::weak\_ptr< ARDOUR::PhaseControl >

is-a: *ARDOUR:AutomationControl*

A PBD::Controllable with associated automation data (AutomationList)

Methods

bool

inverted(unsigned int)

bool

isnil()

void

set\_phase\_invert(unsigned int, bool)

**c** Audio channel index.

**yn** true to invert phase, otherwise false.

### 152.71.1 Inherited from ARDOUR:AutomationControl

Methods

*AutomationList*

alist()

*AutoState*

automation\_state()

double

get\_value()

Get the current effective ‘user’ value based on automation state

void

set\_automation\_state(*AutoState*)

void  
set\_value(double, *GroupControlDisposition*)  
Get and Set ‘internal’ value  
All derived classes must implement this.  
Basic derived classes will ignore  
**group\_override**, but more sophisticated children, notably those that proxy the value setting logic via an object that is aware of group relationships between this control and others, will find it useful.  
void  
start\_touch(double)  
void  
stop\_touch(double)  
bool  
writable()  
Cast  
*Control*  
to\_ctrl()  
*SlavableAutomationControl*  
to\_slavable()

### 152.71.2 Inherited from PBD:Controllable

Methods  
std::string  
name()

### 152.71.3 Inherited from PBD:StatefulPtr

Methods  
void  
clear\_changes()  
Forget about any changes to this object’s properties  
*ID*  
id()  
*OwnedPropertyList*  
properties()

## 152.72 ARDOUR:Playlist

*C<sub>t</sub>*: boost::shared\_ptr< ARDOUR::Playlist >, boost::weak\_ptr< ARDOUR::Playlist >

is-a: *ARDOUR:SessionObjectPtr*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

void

add\_region(*Region*, long, float, bool, int, double, bool)

Note: this calls set\_layer (... , DBL\_MAX) so it will reset the layering index of region

*Region*

combine(*RegionList*)

unsigned int

count\_regions\_at(long)

*Playlist*

cut(*AudioRangeList*&, bool)

*Data Type*

data\_type()

void

duplicate(*Region*, long, long, float)

**gap** from the beginning of the region to the next beginning

void

duplicate\_range(*AudioRange*&, float)

void

duplicate\_until(*Region*, long, long, long)

**gap** from the beginning of the region to the next beginning

**end** the first frame that does not contain a duplicated frame

*Region*

find\_next\_region(long, *RegionPoint*, int)

long

find\_next\_region\_boundary(long, int)

long

find\_next\_transient(long, int)

bool

isnil()

void

lower\_region(*Region*)

```
void
lower_region_to_bottom(Region)
unsigned int
n_regions()
void
raise_region(Region)
void
raise_region_to_top(Region)
Region
region_by_id(ID)
RegionListPtr
region_list()
RegionListPtr
regions_at(long)
RegionListPtr
regions_touched(long, long)
start Range start.
end Range end.
Returns regions which have some part within this range.
RegionListPtr
regions_with_end_within(Range)
RegionListPtr
regions_with_start_within(Range)
void
remove_region(Region)
void
split(long)
void
split_region(Region,MusicFrame)
Region
top_region_at(long)
Region
top_unmuted_region_at(long)
void
uncombine(Region)
Cast
```

*AudioPlaylist*  
to\_audioplaylist()  
*MidiPlaylist*  
to\_midiplaylist()

### 152.72.1 Inherited from ARDOUR:SessionObjectPtr

Methods  
std::string  
name()  
Cast  
*Stateful*  
to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.73 ARDOUR:Plugin

*C<sub>t</sub>*: boost::shared\_ptr< ARDOUR::Plugin >, boost::weak\_ptr< ARDOUR::Plugin >  
is-a: *PBD:StatefulDestructiblePtr*

A plugin is an external module (usually 3rd party provided) loaded into Ardour for the purpose of digital signal processing.

This class provides an abstraction for methods provided by all supported plugin standards such as presets, name, parameters etc.

Plugins are not used directly in Ardour but always wrapped by a PluginInsert.

Methods

*IOPortDescription*  
describe\_io\_port(*Data Type*, bool, unsigned int)  
std::string  
get\_docs()  
*PluginInfo*  
get\_info()  
*LuaTable*(int, ...)  
get\_parameter\_descriptor(unsigned int, *ParameterDescriptor*&)  
std::string  
get\_parameter\_docs(unsigned int)  
bool  
isnil()

```
char*
label()
bool
load_preset(PresetRecord)
Set parameters using a preset
char*
maker()
char*
name()
LuaTable(unsigned int, ...)
nth_parameter(unsigned int, bool&)
unsigned int
parameter_count()
bool
parameter_is_audio(unsigned int)
bool
parameter_is_control(unsigned int)
bool
parameter_is_input(unsigned int)
bool
parameter_is_output(unsigned int)
std::string
parameter_label(unsigned int)
PresetRecord
preset_by_label(std::string)
PresetRecord
preset_by_uri(std::string)
std::string
unique_id()
Cast
LuaProc
to_luaproc()
```

### 152.73.1 Inherited from PBD:StatefulPtr

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.74 ARDOUR:Plugin:IOPortDescription

*C#:* ARDOUR::Plugin::IOPortDescription

Data Members

unsigned int

group\_channel

std::string

group\_name

bool

is\_sidechain

std::string

name

## 152.75 ARDOUR:PluginControl

*C#:* boost::shared\_ptr< ARDOUR::PluginInsert::PluginControl >, boost::weak\_ptr< ARDOUR::PluginInsert::PluginControl >

is-a: *ARDOUR:AutomationControl*

A control that manipulates a plugin parameter (control port).

Methods

bool

isnil()

### 152.75.1 Inherited from ARDOUR:AutomationControl

Methods

*AutomationList*

alist()

*AutoState*

automation\_state()

double

get\_value()

Get the current effective ‘user’ value based on automation state

void

set\_automation\_state(*AutoState*)

void

set\_value(double, *GroupControlDisposition*)

Get and Set ‘internal’ value

All derived classes must implement this.

Basic derived classes will ignore

**group\_override**, but more sophisticated children, notably those that proxy the value setting logic via an object that is aware of group relationships between this control and others, will find it useful.

void

start\_touch(double)

void

stop\_touch(double)

bool

writable()

Cast

*Control*

to\_ctrl()

*SlavableAutomationControl*

to\_slavable()

### 152.75.2 Inherited from PBD:Controllable

Methods

std::string

name()

### 152.75.3 Inherited from PBD:StatefulPtr

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.76 ARDOUR:PluginInfo

*C*#: boost::shared\_ptr< ARDOUR::PluginInfo >, boost::weak\_ptr< ARDOUR::PluginInfo >

Constructor

ARDOUR.PluginInfo()

Methods

*PresetVector*

get\_presets(bool)

bool

is\_instrument()

bool

isnil()

Data Members

std::string

category

std::string

creator

*ARDOUR:ChanCount*

n\_inputs

*ARDOUR:ChanCount*

n\_outputs

std::string

name

std::string

path

*ARDOUR.PluginType*

type

std::string

unique\_id

## 152.77 ARDOUR:PluginInsert

*C*#: boost::shared\_ptr< ARDOUR::PluginInsert >, boost::weak\_ptr< ARDOUR::PluginInsert >

is-a: *ARDOUR:Processor*

Plugin inserts: send data through a plugin

Methods

void

activate()

void

deactivate()

*ChanMapping*

input\_map(unsigned int)

bool

isnil()

*ChanCount*

natural\_input\_streams()

*ChanCount*

natural\_output\_streams()

*ChanMapping*

output\_map(unsigned int)

*Plugin*

plugin(unsigned int)

bool

reset\_parameters\_to\_default()

void

set\_input\_map(unsigned int, *ChanMapping*)

void

set\_output\_map(unsigned int, *ChanMapping*)

bool

strict\_io\_configured()

### 152.77.1 Inherited from ARDOUR:Processor

Methods

bool

active()

std::string

display\_name()

bool

display\_to\_user()

*ChanCount*

input\_streams()

*ChanCount*

output\_streams()

Cast

*Amp*

to\_amp()

*Automatable*

to\_automatable()

*PluginInsert*

to\_insert()

*IOProcessor*

to\_ioprocessor()

*PeakMeter*

to\_meter()

*MonitorProcessor*

to\_monitorprocessor()

*PeakMeter*

to\_peakmeter()

*PluginInsert*

to\_plugininsert()

*SideChain*

to\_sidechain()

*UnknownProcessor*

to\_unknownprocessor()

## 152.77.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.78 ARDOUR:Port

C#: boost::shared\_ptr< ARDOUR::Port >, boost::weak\_ptr< ARDOUR::Port >

Methods

int

connect(std::string)

bool

connected()

Returns true if this port is connected to anything

bool

connected\_to(std::string)

o Port name

Returns true if this port is connected to o, otherwise false.

int

disconnect(std::string)

int

disconnect\_all()

bool

isnil()

std::string

name()

Returns Port short name

std::string

pretty\_name(bool)

Returns Port human readable name

bool

`receives_input()`

Returns true if this Port receives input, otherwise false

`bool`

`sends_output()`

Returns true if this Port sends output, otherwise false

`Cast`

*AudioPort*

`to_audioport()`

*MidiPort*

`to_midiport()`

## 152.79 ARDOUR:PortEngine

*C<sup>++</sup>*: ARDOUR::PortEngine

PortEngine is an abstract base class that defines the functionality required by Ardour.

A Port is basically an endpoint for a datastream (which can either be continuous, like audio, or event-based, like MIDI). Ports have buffers associated with them into which data can be written (if they are output ports) and from which data can be read (if they input ports). Ports can be connected together so that data written to an output port can be read from an input port. These connections can be 1:1, 1:N OR N:1.

Ports may be associated with software only, or with hardware. Hardware related ports are often referred to as physical, and correspond to some relevant physical entity on a hardware device, such as an audio jack or a MIDI connector. Physical ports may be potentially asked to monitor their inputs, though some implementations may not support this.

Most physical ports will also be considered “terminal”, which means that data delivered there or read from there will go to or come from a system outside of the PortEngine implementation’s control (e.g. the analog domain for audio, or external MIDI devices for MIDI). Non-physical ports can also be considered “terminal”. For example, the output port of a software synthesizer is a terminal port, because the data contained in its buffer does not and cannot be considered to come from any other port - it is synthesized by its owner.

Ports also have latency associated with them. Each port has a playback latency and a capture latency:

**capture latency**: how long since the data read from the buffer of a port arrived at a terminal port. The data will have come from the “outside world” if the terminal port is also physical, or will have been synthesized by the entity that owns the terminal port.

**playback latency**: how long until the data written to the buffer of a port will reach a terminal port.

For more detailed questions about the PortEngine API, consult the JACK API documentation, on which this entire object is based.

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.80 ARDOUR:PortList

*C<sup>++</sup>*: std::list<boost::shared\_ptr<ARDOUR::Port>>

Constructor

ARDOUR::PortList()

Methods

*Port*

back()

bool

empty()

*Port*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.81 ARDOUR::PortManager

*C++:* ARDOUR::PortManager

Methods

int

connect(std::string, std::string)

bool

connected(std::string)

int

disconnect(std::string, std::string)

int

disconnect\_port(*Port*)

*LuaTable*(int, ...)

get\_backend\_ports(std::string, *DataType*, *PortFlags*, *StringVector*&)

*LuaTable*(int, ...)

get\_connections(std::string, *StringVector*&)

void

```
get_physical_inputs(DataType,StringVector&,MidiPortFlags,MidiPortFlags)
void
get_physical_outputs(DataType,StringVector&,MidiPortFlags,MidiPortFlags)
Port
get_port_by_name(std::string)
name Full or short name of port
Returns Corresponding Port or 0.
LuaTable(int, ...)
get_ports(DataType,PortList&)
std::string
get_pretty_name_by_name(std::string)
ChanCount
n_physical_inputs()
ChanCount
n_physical_outputs()
bool
physically_connected(std::string)
PortEngine
port_engine()
bool
port_is_physical(std::string)
```

## 152.82 ARDOUR:PortSet

*C++:* boost::shared\_ptr< ARDOUR::PortSet >, boost::weak\_ptr< ARDOUR::PortSet >

An ordered list of Ports, possibly of various types.

This allows access to all the ports as a list, ignoring type, or accessing the nth port of a given type. Note that port(n) and nth\_audio\_port(n) may NOT return the same port.

Each port is held twice; once in a per-type vector of vectors (*\_ports*) and once in a vector of all port (*\_all\_ports*). This is to speed up the fairly common case of iterating over all ports.

Methods

```
void
add(Port)
void
clear()
```

Remove all ports from the PortSet. Ports are not deregistered with the engine, it's the caller's responsibility to not leak here!

bool  
contains(*Port*)  
bool  
empty()  
bool  
isnil()  
unsigned long  
num\_ports(*Data Type*)  
*Port*  
port(*Data Type*, unsigned long)  
nth port of type *t*, or nth port if t = NIL  
**t** data type  
**index** port index  
bool  
remove(*Port*)

## 152.83 ARDOUR:PresentationInfo

*C++:* ARDOUR::PresentationInfo  
is-a: *PBD:Stateful*  
Base class for objects with saveable and undoable state  
Methods  
unsigned int  
color()  
*Flag*  
flags()  
unsigned int  
order()  
void  
set\_color(unsigned int)  
bool  
special(bool)

### 152.83.1 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.84 ARDOUR:PresetRecord

*C#:* ARDOUR::Plugin::PresetRecord

Constructor

ARDOUR.PresetRecord()

Data Members

std::string

label

std::string

uri

bool

user

bool

valid

## 152.85 ARDOUR:PresetVector

*C#:* std::vector<ARDOUR::Plugin::PresetRecord >

Constructor

ARDOUR.PresetVector()

ARDOUR.PresetVector()

Methods

*LuaTable*

add(LuaTable {*PresetRecord*})

*PresetRecord*

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(*PresetRecord*)

unsigned long

size()

*LuaTable*

table()

## 152.86 ARDOUR:Processor

*C*#: boost::shared\_ptr< ARDOUR::Processor >, boost::weak\_ptr< ARDOUR::Processor >

is-a: *ARDOUR:SessionObjectPtr*

A mixer strip element - plugin, send, meter, etc

Methods

void

activate()

bool

active()

void

deactivate()

std::string

display\_name()

bool

display\_to\_user()

*ChanCount*

input\_streams()

bool

isnil()

*ChanCount*

output\_streams()

Cast  
*Amp*  
to\_amp()  
*Automatable*  
to\_automatable()  
*PluginInsert*  
to\_insert()  
*IOProcessor*  
to\_ioprocessor()  
*PeakMeter*  
to\_meter()  
*MonitorProcessor*  
to\_monitorprocessor()  
*PeakMeter*  
to\_peakmeter()  
*PluginInsert*  
to\_plugininsert()  
*SideChain*  
to\_sidechain()  
*UnknownProcessor*  
to\_unknownprocessor()

### 152.86.1 Inherited from ARDOUR:SessionObjectPtr

Methods  
std::string  
name()  
Cast  
*Stateful*  
to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.87 ARDOUR:ProcessorList

C#: std::list<boost::shared\_ptr<ARDOUR::Processor>>

Constructor

ARDOUR.ProcessorList()

Methods

*LuaTable*

add(LuaTable {*Processor*})

*Processor*

back()

bool

empty()

*Processor*

front()

*LuaIter*

iter()

void

push\_back(*Processor*)

void

reverse()

unsigned long

size()

*LuaTable*

table()

void

unique()

## 152.88 ARDOUR:ProcessorVector

*C++*: std::vector<boost::shared\_ptr<ARDOUR::Processor>>

Constructor

ARDOUR.ProcessorVector()

ARDOUR.ProcessorVector()

Methods

*LuaTable*

add(LuaTable {*Processor*})

*Processor*

at(unsigned long)  
bool  
empty()  
*LuaIter*  
iter()  
void  
push\_back(*Processor*)  
unsigned long  
size()  
*LuaTable*  
table()

## 152.89 ARDOUR:Progress

*C<sub>r</sub>*: ARDOUR::Progress

A class to handle reporting of progress of something

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.90 ARDOUR:Properties:BoolProperty

*C<sub>r</sub>*: PBD::PropertyDescriptor<bool>

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.91 ARDOUR:Properties:FloatProperty

*C<sub>r</sub>*: PBD::PropertyDescriptor<float>

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.92 ARDOUR:Properties:FrameposProperty

*C<sub>r</sub>*: PBD::PropertyDescriptor<long>

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.93 ARDOUR:PropertyChange

*C $\ddagger$* : PBD::PropertyChange

A list of IDs of Properties that have changed in some situation or other

Methods

bool

containsBool(*BoolProperty*)

bool

containsFloat(*FloatProperty*)

bool

containsFramePos(*FrameposProperty*)

## 152.94 ARDOUR:PropertyList

*C $\ddagger$* : PBD::PropertyList

A list of properties, mapped using their ID

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.95 ARDOUR:RCConfiguration

*C $\ddagger$* : ARDOUR::RCConfiguration

is-a: *PBD:Configuration*

Base class for objects with saveable and undoable state

Methods

*AFLPosition*

get\_afl\_position()

bool

get\_all\_safe()

bool

get\_allow\_special\_bus\_removal()

bool

get\_ask\_replace\_instrument()

bool

get\_ask\_setup\_instrument()

float

get\_audio\_capture\_buffer\_seconds()

```
float
get_audio_playback_buffer_seconds()
std::string
get_auditioner_output_left()
std::string
get_auditioner_output_right()
bool
get_auto_analyse_audio()
bool
get_auto_connect_standard_busses()
AutoReturnTarget
get_auto_return_target_list()
bool
get_automation_follows_regions()
float
get_automation_interval_msecs()
double
get_automation_thinning_factor()
BufferingPreset
get_buffering_preset()
std::string
get_click_emphasis_sound()
float
get_click_gain()
bool
get_click_record_only()
std::string
get_click_sound()
bool
get_clicking()
bool
get_copy_demo_sessions()
bool
get_create_xrun_marker()
FadeShape
get_default_fade_shape()
```

```
std::string  
get_default_session_parent_dir()
```

*DenormalModel*

```
get_denormal_model()
```

```
bool
```

```
get_denormal_protection()
```

```
bool
```

```
get_disable_disarm_during_roll()
```

```
bool
```

```
get_discover_audio_units()
```

```
bool
```

```
get_discover_vst_on_start()
```

```
unsigned int
```

```
get_disk_choice_space_threshold()
```

```
std::string
```

```
get_donate_url()
```

*EditMode*

```
get_edit_mode()
```

```
bool
```

```
get_exclusive_solo()
```

```
float
```

```
get_export_preroll()
```

```
float
```

```
get_export_silence_threshold()
```

```
unsigned int
```

```
get_feedback_interval_ms()
```

```
bool
```

```
get_first_midi_bank_is_zero()
```

```
std::string
```

```
get_freesound_download_dir()
```

```
bool
```

```
get_hide_dummy_backend()
```

```
bool
```

```
get_hiding_groups_deactivates_groups()
```

```
int
```

```
get_history_depth()
```

```
int
get_initial_program_change()
AutoConnectOption
get_input_auto_connect()
int
get_inter_scene_gap_frames()
bool
get_latched_record_enable()
LayerModel
get_layer_model()
bool
get_link_send_and_route_panner()
std::string
get_linux_pingback_url()
ListenPosition
get_listen_position()
bool
get_locate_while_waiting_for_sync()
bool
get_loop_is_mode()
std::string
get_ltc_output_port()
float
get_ltc_output_volume()
bool
get_ltc_send_continuously()
std::string
get_ltc_source_port()
float
get_max_gain()
unsigned int
get_max_recent_sessions()
unsigned int
get_max_recent_templates()
float
get_meter_falloff()
```

*MeterType*

get\_meter\_type\_bus()

*MeterType*

get\_meter\_type\_master()

*MeterType*

get\_meter\_type\_track()

std::string

get\_midi\_audition\_synth\_uri()

bool

get\_midi\_feedback()

bool

get\_midi\_input\_follows\_selection()

float

get\_midi\_readahead()

float

get\_midi\_track\_buffer\_seconds()

unsigned int

get\_minimum\_disk\_read\_bytes()

unsigned int

get\_minimum\_disk\_write\_bytes()

bool

get mmc control()

int

get mmc receive device id()

int

get mmc send device id()

std::string

get\_monitor\_bus\_preferred\_bundle()

*MonitorModel*

get\_monitoring\_model()

int

get\_mtc\_qf\_speed\_tolerance()

bool

get\_mute\_affects\_control\_outs()

bool

get\_mute\_affects\_main\_outs()

```
bool  
get_mute_affects_post_fader()  
bool  
get_mute_affects_pre_fader()  
bool  
get_new_plugins_active()  
unsigned int  
get_osc_port()  
std::string  
get_osx_pingback_url()  
AutoConnectOption  
get_output_auto_connect()  
unsigned int  
get_periodic_safety_backup_interval()  
bool  
get_periodic_safety_backups()  
PFLPosition  
get_pfl_position()  
std::string  
get_plugin_path_lvxst()  
std::string  
get_plugin_path_vst()  
bool  
get_plugins_stop_with_transport()  
long  
get_postroll()  
long  
get_preroll()  
float  
get_preroll_seconds()  
int  
get_processor_usage()  
bool  
get_quieten_at_speed()  
long  
get_range_location_minimum()
```

```
std::string
get_reference_manual_url()
bool
get_region_boundaries_from_onscreen_tracks()
bool
get_region_boundaries_from_selected_tracks()
RegionSelectionAfterSplit
get_region_selection_after_split()
bool
get_replicate_missing_region_channels()
float
get_rf_speed()
bool
get_save_history()
int
get_saved_history_depth()
bool
get_seamless_loop()
bool
get_send_ltc()
bool
get_send_midi_clock()
bool
get_send_mmc()
bool
get_send_mtc()
bool
get_show_solo_mutes()
bool
get_show_video_export_info()
bool
get_show_video_server_dialog()
ShuttleBehaviour
get_shuttle_behaviour()
float
get_shuttle_max_speed()
```

```
float
get_shuttle_speed_factor()

float
get_shuttle_speed_threshold()

ShuttleUnits
get_shuttle_units()

bool
get_skip_playback()

bool
get_solo_control_is_listen_control()

float
get_solo_mute_gain()

bool
get_solo_mute_override()

bool
get_stop_at_session_end()

bool
get_stop_recording_on_xrun()

bool
get_strict_io()

SyncSource
get_sync_source()

bool
get_tape_machine_mode()

bool
get_timecode_source_2997()

bool
get_timecode_source_is_synced()

bool
get_timecode_sync_frame_rate()

bool
get_trace_midi_input()

bool
get_trace_midi_output()

TracksAutoNamingRule
get_tracks_auto_naming()
```

```
float
get_transient_sensitivity()

bool
get_try_autostart_engine()

std::string
get_tutorial_manual_url()
std::string
get_updates_url()
bool
get_use_click_emphasis()
bool
get_use_lxvst()
bool
get_use_macvst()
bool
get_use_monitor_bus()
bool
get_use_osc()
bool
get_use_overlap_equivency()
bool
get_use_plugin_own_gui()
bool
get_use_tranzport()
bool
get_use_windows_vst()
bool
get_verbose_plugin_scan()
bool
get_verify_remove_last_capture()
bool
get_video_advanced_setup()
std::string
get_video_server_docroot()
std::string
get_video_server_url()
```

```
int
get_vst_scan_timeout()
std::string
get_windows_pingback_url()
bool
set_afl_position(AFLPosition)
bool
set_all_safe(bool)
bool
set_allow_special_bus_removal(bool)
bool
set_ask_replace_instrument(bool)
bool
set_ask_setup_instrument(bool)
bool
set_audio_capture_buffer_seconds(float)
bool
set_audio_playback_buffer_seconds(float)
bool
set_auditioner_output_left(std::string)
bool
set_auditioner_output_right(std::string)
bool
set_auto_analyse_audio(bool)
bool
set_auto_connect_standard_busses(bool)
bool
set_auto_return_target_list(AutoReturnTarget)
bool
set_automation_follows_regions(bool)
bool
set_automation_interval_msecs(float)
bool
set_automation_thinning_factor(double)
bool
set_buffering_preset(BufferingPreset)
```

```
bool
set_click_emphasis_sound(std::string)
bool
set_click_gain(float)
bool
set_click_record_only(bool)
bool
set_click_sound(std::string)
bool
set_clicking(bool)
bool
set_copy_demo_sessions(bool)
bool
set_create_xrun_marker(bool)
bool
set_default_fade_shape(FadeShape)
bool
set_default_session_parent_dir(std::string)
bool
set_denormal_model(DenormalModel)
bool
set_denormal_protection(bool)
bool
set_disable_disarm_during_roll(bool)
bool
set_discover_audio_units(bool)
bool
set_discover_vst_on_start(bool)
bool
set_disk_choice_space_threshold(unsigned int)
bool
set_donate_url(std::string)
bool
set_edit_mode(EditMode)
bool
set_exclusive_solo(bool)
```

```
bool
set_export_preroll(float)
bool
set_export_silence_threshold(float)
bool
set_feedback_interval_ms(unsigned int)
bool
set_first_midi_bank_is_zero(bool)
bool
set_freesound_download_dir(std::string)
bool
set_hide_dummy_backend(bool)
bool
set_hiding_groups_deactivates_groups(bool)
bool
set_history_depth(int)
bool
set_initial_program_change(int)
bool
set_input_auto_connect(AutoConnectOption)
bool
set_inter_scene_gap_frames(int)
bool
set_latched_record_enable(bool)
bool
set_layer_model(LayerModel)
bool
set_link_send_and_route_panner(bool)
bool
set_linux_pingback_url(std::string)
bool
set_listen_position(ListenPosition)
bool
set_locate_while_waiting_for_sync(bool)
bool
set_loop_is_mode(bool)
```

```
bool
set_ltc_output_port(std::string)
bool
set_ltc_output_volume(float)
bool
set_ltc_send_continuously(bool)
bool
set_ltc_source_port(std::string)
bool
set_max_gain(float)
bool
set_max_recent_sessions(unsigned int)
bool
set_max_recent_templates(unsigned int)
bool
set_meter_falloff(float)
bool
set_meter_type_bus(MeterType)
bool
set_meter_type_master(MeterType)
bool
set_meter_type_track(MeterType)
bool
set_midi_audition_synth_uri(std::string)
bool
set_midi_feedback(bool)
bool
set_midi_input_follows_selection(bool)
bool
set_midi_readahead(float)
bool
set_midi_track_buffer_seconds(float)
bool
set_minimum_disk_read_bytes(unsigned int)
bool
set_minimum_disk_write_bytes(unsigned int)
```

```
bool
set_mmc_control(bool)
bool
set_mmc_receive_device_id(int)
bool
set_mmc_send_device_id(int)
bool
set_monitor_bus_preferred_bundle(std::string)
bool
set_monitoring_model(MonitorModel)
bool
set_mtc_qf_speed_tolerance(int)
bool
set_mute_affects_control_outs(bool)
bool
set_mute_affects_main_outs(bool)
bool
set_mute_affects_post_fader(bool)
bool
set_mute_affects_pre_fader(bool)
bool
set_new_plugins_active(bool)
bool
set_osc_port(unsigned int)
bool
set_osx_pingback_url(std::string)
bool
set_output_auto_connect(AutoConnectOption)
bool
set_periodic_safety_backup_interval(unsigned int)
bool
set_periodic_safety_backups(bool)
bool
set_pfl_position(PFLPosition)
bool
set_plugin_path_lvxst(std::string)
```

```
bool
set_plugin_path_vst(std::string)
bool
set_plugins_stop_with_transport(bool)
bool
set_postroll(long)
bool
set_preroll(long)
bool
set_preroll_seconds(float)
bool
set_processor_usage(int)
bool
set_quieten_at_speed(bool)
bool
set_range_location_minimum(long)
bool
set_reference_manual_url(std::string)
bool
set_region_boundaries_from_onscreen_tracks(bool)
bool
set_region_boundaries_from_selected_tracks(bool)
bool
set_region_selection_after_split(RegionSelectionAfterSplit)
bool
set_replicate_missing_region_channels(bool)
bool
set_rf_speed(float)
bool
set_save_history(bool)
bool
set_saved_history_depth(int)
bool
set_seamless_loop(bool)
bool
set_send_ltc(bool)
```

```
bool
set_send_midi_clock(bool)
bool
set_send_mmc(bool)
bool
set_send_mtc(bool)
bool
set_show_solo_mutes(bool)
bool
set_show_video_export_info(bool)
bool
set_show_video_server_dialog(bool)
bool
set_shuttle_behaviour(ShuttleBehaviour)
bool
set_shuttle_max_speed(float)
bool
set_shuttle_speed_factor(float)
bool
set_shuttle_speed_threshold(float)
bool
set_shuttle_units(ShuttleUnits)
bool
set_skip_playback(bool)
bool
set_solo_control_is_listen_control(bool)
bool
set_solo_mute_gain(float)
bool
set_solo_mute_override(bool)
bool
set_stop_at_session_end(bool)
bool
set_stop_recording_on_xrun(bool)
bool
set_strict_io(bool)
```

```
bool
set_sync_source(SyncSource)
bool
set_tape_machine_mode(bool)
bool
set_timecode_source_2997(bool)
bool
set_timecode_source_is_synced(bool)
bool
set_timecode_sync_frame_rate(bool)
bool
set_trace_midi_input(bool)
bool
set_trace_midi_output(bool)
bool
set_tracks_auto_naming(TracksAutoNamingRule)
bool
set_transient_sensitivity(float)
bool
set_try_autostart_engine(bool)
bool
set_tutorial_manual_url(std::string)
bool
set_updates_url(std::string)
bool
set_use_click_emphasis(bool)
bool
set_use_lxvst(bool)
bool
set_use_macvst(bool)
bool
set_use_monitor_bus(bool)
bool
set_use_osc(bool)
bool
set_use_overlap_equivalency(bool)
```

```
bool  
set_use_plugin_own_gui(bool)  
bool  
set_use_tranzport(bool)  
bool  
set_use_windows_vst(bool)  
bool  
set_verbose_plugin_scan(bool)  
bool  
set_verify_remove_last_capture(bool)  
bool  
set_video_advanced_setup(bool)  
bool  
set_video_server_docroot(std::string)  
bool  
set_video_server_url(std::string)  
bool  
set_vst_scan_timeout(int)  
bool  
set_windows_pingback_url(std::string)
```

#### Properties

*ARDOUR.AFLPosition*

```
afl_position  
bool  
all_safe  
bool  
allow_special_bus_removal  
bool  
ask_replace_instrument  
bool  
ask_setup_instrument  
float  
audio_capture_buffer_seconds  
float  
audio_playback_buffer_seconds  
std::string
```

```
auditioner_output_left
std::string
auditioner_output_right
bool
auto_analyse_audio
bool
auto_connect_standard_busses
ARDOUR.AutoReturnTarget
auto_return_target_list
bool
automation_follows_regions
float
automation_interval_msecs
double
automation_thinning_factor
ARDOUR.BufferingPreset
buffering_preset
std::string
click_emphasis_sound
float
click_gain
bool
click_record_only
std::string
click_sound
bool
clicking
bool
copy_demo_sessions
bool
create_xrun_marker
ARDOUR.FadeShape
default_fade_shape
std::string
default_session_parent_dir
ARDOUR.DenormalModel
```

denormal\_model  
bool  
denormal\_protection  
bool  
disable\_disarm\_during\_roll  
bool  
discover\_audio\_units  
bool  
discover\_vst\_on\_start  
unsigned int  
disk\_choice\_space\_threshold  
std::string  
donate\_url  
*ARDOUR.EditMode*  
edit\_mode  
bool  
exclusive\_solo  
float  
export\_preroll  
float  
export\_silence\_threshold  
unsigned int  
feedback\_interval\_ms  
bool  
first\_midi\_bank\_is\_zero  
std::string  
freesound\_download\_dir  
bool  
hide\_dummy\_backend  
bool  
hiding\_groups\_deactivates\_groups  
int  
history\_depth  
int  
initial\_program\_change  
*ARDOUR.AutoConnectOption*

```
input_auto_connect
int
inter_scene_gap_frames
bool
latched_record_enable
ARDOUR.LayerModel
layer_model
bool
link_send_and_route_panner
std::string
linux_pingback_url
ARDOUR.ListenPosition
listen_position
bool
locate_while_waiting_for_sync
bool
loop_is_mode
std::string
ltc_output_port
float
ltc_output_volume
bool
ltc_send_continuously
std::string
ltc_source_port
float
max_gain
unsigned int
max_recent_sessions
unsigned int
max_recent_templates
float
meter_falloff
ARDOUR.MeterType
meter_type_bus
ARDOUR.MeterType
```

```
meter_type_master
ARDOUR.MeterType
meter_type_track
std::string
midi_audition_synth_uri
bool
midi_feedback
bool
midi_input_follows_selection
float
midi_readahead
float
midi_track_buffer_seconds
unsigned int
minimum_disk_read_bytes
unsigned int
minimum_disk_write_bytes
bool
mmc_control
int
mmc_receive_device_id
int
mmc_send_device_id
std::string
monitor_bus_preferred_bundle
ARDOUR.MonitorModel
monitoring_model
int
mtc_qf_speed_tolerance
bool
mute_affects_control_outs
bool
mute_affects_main_outs
bool
mute_affects_post_fader
bool
```

```
mute_affects_pre_fader
bool
new_plugins_active
unsigned int
osc_port
std::string
osx_pingback_url
ARDOUR.AutoConnectOption
output_auto_connect
unsigned int
periodic_safety_backup_interval
bool
periodic_safety_backups
ARDOUR.PFLPosition
pfl_position
std::string
plugin_path_lvxst
std::string
plugin_path_vst
bool
plugins_stop_with_transport
long
postroll
long
preroll
float
preroll_seconds
int
processor_usage
bool
quieten_at_speed
long
range_location_minimum
std::string
reference_manual_url
bool
```

region\_boundaries\_from\_onscreen\_tracks  
bool  
region\_boundaries\_from\_selected\_tracks  
*ARDOUR.RegionSelectionAfterSplit*  
region\_selection\_after\_split  
bool  
replicate\_missing\_region\_channels  
float  
rf\_speed  
bool  
save\_history  
int  
saved\_history\_depth  
bool  
seamless\_loop  
bool  
send\_ltc  
bool  
send\_midi\_clock  
bool  
send\_mmc  
bool  
send\_mtc  
bool  
show\_solo\_mutes  
bool  
show\_video\_export\_info  
bool  
show\_video\_server\_dialog  
*ARDOUR.ShuttleBehaviour*  
shuttle\_behaviour  
float  
shuttle\_max\_speed  
float  
shuttle\_speed\_factor  
float

shuttle\_speed\_threshold  
*ARDOUR.ShuttleUnits*

shuttle\_units  
bool

skip\_playback  
bool

solo\_control\_is\_listen\_control  
float

solo\_mute\_gain  
bool

solo\_mute\_override  
bool

stop\_at\_session\_end  
bool

stop\_recording\_on\_xrun  
bool

strict\_io  
*ARDOUR.SyncSource*

sync\_source  
bool

tape\_machine\_mode  
bool

timecode\_source\_2997  
bool

timecode\_source\_is\_synced  
bool

timecode\_sync\_frame\_rate  
bool

trace\_midi\_input  
bool

trace\_midi\_output

*ARDOUR.TracksAutoNamingRule*

tracks\_auto\_naming  
float

transient\_sensitivity  
bool

```
try_autostart_engine
std::string
tutorial_manual_url
std::string
updates_url
bool
use_click_emphasis
bool
use_lxvst
bool
use_macvst
bool
use_monitor_bus
bool
use_osc
bool
use_overlap_equivency
bool
use_plugin_own_gui
bool
use_tranzport
bool
use_windows_vst
bool
verbose_plugin_scan
bool
verify_remove_last_capture
bool
video_advanced_setup
std::string
video_server_docroot
std::string
video_server_url
int
vst_scan_timeout
std::string
```

windows\_pingback\_url

### 152.95.1 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.96 ARDOUR:ReadOnlyControl

*Cf:* boost::shared\_ptr< ARDOUR::ReadOnlyControl >, boost::weak\_ptr< ARDOUR::ReadOnlyControl >

is-a: *PBD:StatefulDestructiblePtr*

Methods

*ParameterDescriptor*

desc()

std::string

describe\_parameter()

double

get\_parameter()

bool

isnil()

### 152.96.1 Inherited from PBD:StatefulPtr

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.97 ARDOUR:Readable

*C<sub>++</sub>*: boost::shared\_ptr< ARDOUR::Readable >, boost::weak\_ptr< ARDOUR::Readable >

Methods

bool

isnil()

unsigned int

n\_channels()

long

read(*FloatArray*, long, long, int)

long

readable\_length()

## 152.98 ARDOUR:Region

*C<sub>++</sub>*: boost::shared\_ptr< ARDOUR::Region >, boost::weak\_ptr< ARDOUR::Region >

is-a: *ARDOUR:SessionObjectPtr*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

bool

at\_natural\_position()

bool

automatic()

bool

can\_move()

bool

captured()

void

clear\_sync\_position()

*Control*

control(*Parameter*, bool)

bool

covers(long)

void

cut\_end(long, int)

void

cut\_front(long, int)

*DataType*

data\_type()

bool

external()

bool

has\_transients()

bool

hidden()

bool

import()

bool

is\_compound()

bool

isnil()

unsigned int

layer()

long

length()

bool

locked()

void

lower()

void

lower\_to\_bottom()

*StringVector*

master\_source\_names()

*SourceList*

master\_sources()

void

move\_start(long, int)

void

move\_to\_natural\_position()

bool

muted()

unsigned int

```
n_channels()  
void  
nudge_position(long)  
bool  
opaque()  
long  
position()
```

How the region parameters play together:

POSITION: first frame of the region along the timeline START: first frame of the region within its source(s)  
LENGTH: number of frames the region represents

```
bool  
position_locked()  
double  
quarter_note()  
void  
raise()  
void  
raise_to_top()  
void  
set_hidden(bool)  
void  
set_initial_position(long)
```

A gui may need to create a region, then place it in an initial position determined by the user. When this takes place within one gui operation, we have to reset `_last_position` to prevent an implied move.

```
void  
set_length(long, int)  
void  
set_locked(bool)  
void  
set_muted(bool)  
void  
set_opaque(bool)  
void  
set_position(long, int)  
void  
set_position_locked(bool)  
void
```

```
set_start(long)
void
set_sync_position(long)
Set the region's sync point.
absolute_pos Session time.
void
set_video_locked(bool)
float
shift()
Source
source(unsigned int)
long
start()
float
stretch()
bool
sync_marked()
LuaTable(long, ...)
sync_offset(int&)
long
sync_position()
>Returns Sync position in session time
Int64List
transients()
void
trim_end(long, int)
void
trim_front(long, int)
void
trim_to(long, long, int)
bool
video_locked()
bool
whole_file()
Cast
AudioRegion
```

to\_audioregion()

*MidiRegion*

to\_midiregion()

*Readable*

to\_readable()

### 152.98.1 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.99 ARDOUR:RegionFactory

C#: ARDOUR::RegionFactory

Methods

*Region*

clone\_region(*Region*, bool, bool)

*Region*

region\_by\_id(*ID*)

*RegionMap*

regions()

## 152.100 ARDOUR:RegionList

C#: std::list<boost::shared\_ptr<ARDOUR::Region>>

Constructor

ARDOUR.RegionList()

Methods

*Region*

back()

bool  
empty()  
*Region*  
front()  
*LuaIter*  
iter()  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()

## 152.101 ARDOUR:RegionListPtr

*C#:* boost::shared\_ptr<std::list<boost::shared\_ptr<ARDOUR::Region>>>  
Constructor

ARDOUR.RegionListPtr()

Methods

*LuaTable*  
add(LuaTable {*Region*})  
bool  
empty()  
*LuaIter*  
iter()  
void  
push\_back(*Region*)  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()  
void  
unique()

## 152.102 ARDOUR:RegionMap

*C<sub>7</sub>*: std::map<PBD::ID, boost::shared\_ptr<ARDOUR::Region>>

Constructor

ARDOUR.RegionMap()

Methods

*LuaTable*

add(LuaTable {*Region*})

...

at(*-lua-*)

void

clear()

unsigned long

count(*ID*)

bool

empty()

*LuaIter*

iter()

unsigned long

size()

*LuaTable*

table()

## 152.103 ARDOUR:RegionVector

*C<sub>7</sub>*: std::vector<boost::shared\_ptr<ARDOUR::Region>>

Constructor

ARDOUR.RegionVector()

ARDOUR.RegionVector()

Methods

*LuaTable*

add(LuaTable {*Region*})

*Region*

at(unsigned long)

---

```

bool
empty()
LuaIter
iter()
void
push_back(Region)
unsigned long
size()
LuaTable
table()

```

## 152.104 ARDOUR:Route

*C<sub>r</sub>*: boost::shared\_ptr< ARDOUR::Route >, boost::weak\_ptr< ARDOUR::Route >  
 is-a: *ARDOUR:Stripable*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

```

bool
active()
int
add_processor_by_index(Processor, int, ProcessorStreams, bool)

```

Add a processor to a route such that it ends up with a given index into the visible processors.

**index** Index to add the processor at, or -1 to add at the end of the list.

Returns 0 on success, non-0 on failure.

```

bool
add_sidechain(Processor)

```

*Amp*

amp()

std::string

comment()

bool

customize\_plugin\_insert(*Processor*, unsigned int, *ChanCount*, *ChanCount*)

enable custom plugin-insert configuration

**proc** Processor to customize

**count** number of plugin instances to use (if zero, reset to default)

**outs** output port customization

**sinks** input pins for variable-I/O plugins

Returns true if successful

*IO*

`input()`

`bool`

`isnil()`

*Delivery*

`main_outs()`

the signal processor at end of the processing chain which produces output

`bool`

`muted()`

*ChanCount*

`n_inputs()`

*ChanCount*

`n_outputs()`

*Processor*

`nth_plugin(unsigned int)`

*Processor*

`nth_processor(unsigned int)`

*Processor*

`nth_send(unsigned int)`

*IO*

`output()`

*PannerShell*

`panner_shell()`

*PeakMeter*

`peak_meter()`

\*\*\*\*\* Pure interface begins  
here\*\*\*\*\*  
int

`remove_processor(Processor,ProcessorStreams, bool)`

remove plugin/processor

**proc** processor to remove

**err** error report (index where removal failed, channel-count why it failed) may be nil

**need\_process\_lock** if locking is required (set to true, unless called from RT context with lock)

Returns 0 on success

```
int
remove_processors(ProcessorList,ProcessorStreams)
bool
remove_sidechain(Processor)
int
reorder_processors(ProcessorList,ProcessorStreams)
int
replace_processor(Processor,Processor,ProcessorStreams)
replace plugin/processor with another
old processor to remove
sub processor to substitute the old one with
err error report (index where removal failed, channel-count why it failed) may be nil
Returns 0 on success
```

bool

reset\_plugin\_insert(*Processor*)

reset plugin-insert configuration to default, disable customizations.

This is equivalent to calling

`customize_plugin_insert (proc, 0, unused)`

**proc** Processor to reset

Returns true if successful

void

set\_active(bool, void\*)

void

set\_comment(std::string, void\*)

void

set\_meter\_point(*MeterPoint*, bool)

bool

set\_name(std::string)

bool

set\_strict\_io(bool)

bool

soloed()

bool

strict\_io()

*Processor*

`the_instrument()`

Return the first processor that accepts has at least one MIDI input and at least one audio output. In the vast majority of cases, this will be “the instrument”. This does not preclude other MIDI->audio processors later in the processing chain, but that would be a special case not covered by this utility function.

*Amp*

`trim()`

*Cast*

*Automatable*

`to_automatable()`

*Slavable*

`to_slavable()`

*Track*

`to_track()`

### 152.104.1 Inherited from ARDOUR:Stripable

Methods

*AutomationControl*

`comp_enable_control()`

*AutomationControl*

`comp_makeup_control()`

*AutomationControl*

`comp_mode_control()`

`std::string`

`comp_mode_name(unsigned int)`

*ReadOnlyControl*

`comp_redux_control()`

*AutomationControl*

`comp_speed_control()`

`std::string`

`comp_speed_name(unsigned int)`

*AutomationControl*

`comp_threshold_control()`

`unsigned int`

`eq_band_cnt()`

`std::string`

`eq_band_name(unsigned int)`

*AutomationControl*

```
eq_enable_control()  
AutomationControl  
eq_freq_control(unsigned int)  
AutomationControl  
eq_gain_control(unsigned int)  
AutomationControl  
eq_q_control(unsigned int)  
AutomationControl  
eq_shape_control(unsigned int)  
AutomationControl  
filter_enable_controllable(bool)  
AutomationControl  
filter_freq_controllable(bool)  
AutomationControl  
filter_slope_controllable(bool)  
GainControl  
gain_control()  
bool  
is_auditioner()  
bool  
is_hidden()  
bool  
is_master()  
bool  
is_monitor()  
bool  
is_selected()  
AutomationControl  
master_send_enable_control()  
MonitorProcessor  
monitor_control()  
MuteControl  
mute_control()  
AutomationControl  
pan_azimuth_control()  
AutomationControl
```

pan\_elevation\_control()  
*AutomationControl*

pan\_frontback\_control()  
*AutomationControl*

pan\_lfe\_control()  
*AutomationControl*

pan\_width\_control()  
*PhaseControl*

phase\_control()  
*PresentationInfo*

presentation\_info\_ptr()  
*AutomationControl*

rec\_enable\_control()  
*AutomationControl*

rec\_safe\_control()  
*AutomationControl*

send\_enable\_control(unsigned int)  
*AutomationControl*

send\_level\_control(unsigned int)  
std::string

send\_name(unsigned int)

void

set\_presentation\_order(unsigned int)  
*SoloControl*

solo\_control()  
*SoloIsolateControl*

solo\_isolate\_control()  
*SoloSafeControl*

solo\_safe\_control()  
*GainControl*

trim\_control()  
*Cast*

Route

to\_route()  
*VCA*

to\_vca()

## 152.104.2 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.105 ARDOUR:Route:ProcessorStreams

*C#:* ARDOUR::Route::ProcessorStreams

A record of the stream configuration at some point in the processor list. Used to return where and why an processor list configuration request failed.

Constructor

ARDOUR.Route.ProcessorStreams()

## 152.106 ARDOUR:RouteGroup

*C#:* ARDOUR::RouteGroup

is-a: ARDOUR:SessionObject

A group identifier for routes.

RouteGroups permit to define properties which are shared among all Routes that use the given identifier.

A route can at most be in one group.

Methods

int

add(*Route*)

Add a route to a group. Adding a route which is already in the group is allowed; nothing will happen.

**r** Route to add.

void

clear()

void

destroy\_subgroup()

bool

```
empty()
int
group_master_number()
bool
has_subgroup()
bool
is_active()
bool
is_color()
bool
is_gain()
bool
is_hidden()
bool
is_monitoring()
bool
is_mute()
bool
is_recenable()
bool
is_relative()
bool
is_route_active()
bool
is_select()
bool
is_solo()
void
make_subgroup(bool,Placement)
int
remove(Route)
unsigned int
rgba()
RouteListPtr
route_list()
void
```

```
set_active(bool, void*)
void
set_color(bool)
void
set_gain(bool)
void
set_hidden(bool, void*)
void
set_monitoring(bool)
void
set_mute(bool)
void
set_recenable(bool)
void
set_relative(bool, void*)
void
set_rgba(unsigned int)
set route-group color and notify UI about change
void
set_route_active(bool)
void
set_select(bool)
void
set_solo(bool)
unsigned long
size()
```

### 152.106.1 Inherited from ARDOUR:SessionObject

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

## 152.107 ARDOUR:RouteGroupList

*C<sub>++</sub>*: std::list<ARDOUR::RouteGroup\*>

Constructor

ARDOUR.RouteGroupList()

Methods

*RouteGroup*

back()

bool

empty()

*RouteGroup*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.108 ARDOUR:RouteList

*C<sub>++</sub>*: std::list<boost::shared\_ptr<ARDOUR::Route>>

Constructor

ARDOUR.RouteList()

Methods

*Route*

back()

bool

empty()

*Route*

front()

*LuaIter*

iter()  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()

## 152.109 ARDOUR:RouteListPtr

*C#:* boost::shared\_ptr<std::list<boost::shared\_ptr<ARDOUR::Route>>>  
Constructor

ARDOUR.RouteListPtr()  
Methods  
*LuaTable*  
add(*LuaTable* {*Route*})  
bool  
empty()  
*LuaIter*  
iter()  
void  
push\_back(*Route*)  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()  
void  
unique()

## 152.110 ARDOUR:Session

*C#:* ARDOUR::Session  
Ardour Session

Methods

void

abort\_reversible\_command()

abort an open undo command This must only be called after begin\_reversible\_command ()

bool

actively\_recording()

void

add\_command(*Command*)

void

add\_internal\_sends(*Route, Placement, RouteListPtr*)

int

add\_master\_bus(*ChanCount*)

void

add\_monitor\_section()

*StatefulDiffCommand*

add\_stateful\_diff\_command(*StatefulDestructiblePtr*)

create an StatefulDiffCommand from the given object and add it to the stack.

This function must only be called after begin\_reversible\_command. Failing to do so may lead to a crash.

**sfd** the object to diff

Returns the allocated StatefulDiffCommand (already added via add\_command)

void

begin\_reversible\_command(std::string)

begin collecting undo information

This call must always be followed by either begin\_reversible\_command() or commit\_reversible\_command()

**cmd\_name** human readable name for the undo operation

void

cancel\_all\_solo()

*SessionConfiguration*

cfg()

void

clear\_all\_solo\_state(*RouteListPtr*)

void

commit\_reversible\_command(*Command*)

finalize an undo command and commit pending transactions

This must only be called after begin\_reversible\_command ()

**cmd** (additional) command to add

*Controllable*controllable\_by\_id(*ID*)

long

current\_end\_frame()

long

current\_start\_frame()

void

disable\_record(bool, bool)

bool

end\_is\_free()

*AudioEngine*

engine()

bool

export\_track\_state(*RouteListPtr*, std::string)

long

frame\_rate()

“actual” sample rate of session, set by current audioengine rate, pullup/down etc.

unsigned int

get\_block\_size()

bool

get\_play\_loop()

*Route*

get\_remote\_nth\_route(unsigned int)

*Stripable*get\_remote\_nth\_stripable(unsigned int, *Flag*)*RouteListPtr*

get\_routes()

*BufferSet*get\_scratch\_buffers(*ChanCount*, bool)*BufferSet*get\_silent\_buffers(*ChanCount*)*StripableList*

get\_stripables()

*RouteListPtr*

get\_tracks()

void

```
goto_end()
void
goto_start(bool)
long
last_transport_start()
bool
listening()

Locations
locations()

Route
master_out()
void
maybe_enable_record(bool)

Route
monitor_out()
std::string
name()

RouteList
new_audio_route(int, int, RouteGroup, unsigned int, std::string, Flag, unsigned int)

AudioTrackList
new_audio_track(int, int, RouteGroup, unsigned int, std::string, unsigned int, TrackMode)
RouteList
new_midi_route(RouteGroup, unsigned int, std::string, bool, PluginInfo, PresetRecord, Flag, unsigned int)

MidiTrackList
new_midi_track(ChanCount, ChanCount, bool, PluginInfo, PresetRecord, RouteGroup, unsigned int,
std::string, unsigned int, TrackMode)
RouteList
new_route_from_template(unsigned int, unsigned int, std::string, std::string, PlaylistDisposition)
RouteGroup
new_route_group(std::string)
long
nominal_frame_rate()
“native” sample rate of session, regardless of current audioengine rate, pullup/down etc
std::string
path()

Processor
```

```
processor_by_id(ID)
RecordState
record_status()
void
remove_monitor_section()
void
remove_route_group(RouteGroup)
void
request_locate(long, bool)
void
request_play_loop(bool, bool)
void
request_stop(bool, bool)
void
request_transport_speed(double, bool)
Route
route_by_id(ID)
Route
route_by_name(std::string)
Route
route_by_selected_count(unsigned int)
RouteGroupList
route_groups()
...
sample_to_timecode_lua(~lua~)
double
samples_per_timecode_frame()
int
save_state(std::string, bool, bool, bool)
save session
snapshot_name name of the session (use an empty string for the current name)
pending save a ‘recovery’, not full state (default: false)
switch_to_snapshot switch to given snapshot after saving (default: false)
template_only save a session template (default: false)
```

Returns zero on success

void  
scripts\_changed()  
void  
set\_control(*AutomationControl*, double, *GroupControlDisposition*)  
void  
set\_controls(*ControlListPtr*, double, *GroupControlDisposition*)  
void  
set\_dirty()  
void  
set\_end\_is\_free(bool)  
void  
set\_exclusive\_input\_active(*RouteListPtr*, bool, bool)  
std::string  
snap\_name()  
bool  
solo\_isolated()  
bool  
soloing()  
*Source*  
source\_by\_id(*ID*)  
*TempoMap*  
tempo\_map()  
bool  
timecode\_drop\_frames()  
long  
timecode\_frames\_per\_hour()  
double  
timecode\_frames\_per\_second()  
...  
timecode\_to\_sample\_lua(~lua~)  
*Track*  
track\_by\_diskstream\_id(*ID*)  
long  
transport\_frame()  
bool

```
transport_rolling()
double
transport_speed()
StringList
unknown_processors()
VCAManager
vca_manager()
long
worst_input_latency()
long
worst_output_latency()
long
worst_playback_latency()
long
worst_track_latency()
```

## 152.111 ARDOUR:SessionConfiguration

*C<sub>r</sub>*: ARDOUR::SessionConfiguration

is-a: *PBD:Configuration*

Base class for objects with saveable and undoable state

Methods

```
std::string
get_audio_search_path()
bool
get_auto_input()
bool
get_auto_play()
bool
get_auto_return()
bool
get_count_in()
unsigned int
get_destructive_xfade_msecs()
bool
get_external_sync()
```

```
bool  
get_glue_new_markers_to_bars_and_beats()  
bool  
get_glue_new_regions_to_bars_and_beats()  
InsertMergePolicy  
get_insert_merge_policy()  
bool  
get_jack_time_master()  
bool  
get_layered_record_mode()  
unsigned int  
get_meterbridge_label_height()  
bool  
get_midi_copy_is_fork()  
std::string  
get_midi_search_path()  
long  
get_minitimeline_span()  
SampleFormat  
get_native_file_data_format()  
HeaderFormat  
get_native_file_header_format()  
bool  
get_punch_in()  
bool  
get_punch_out()  
std::string  
get_raid_path()  
bool  
get_realtime_export()  
MonitorChoice  
get_session_monitoring()  
bool  
get_show_busses_on_meterbridge()  
bool  
get_show_group_tabs()
```

```
bool
get_show_master_on_meterbridge()

bool
get_show_midi_on_meterbridge()

bool
get_show_monitor_on_meterbridge()

bool
get_show_mute_on_meterbridge()

bool
get_show_name_on_meterbridge()

bool
get_show_rec_on_meterbridge()

bool
get_show_region_fades()

bool
get_show_solo_on_meterbridge()

bool
get_show_summary()
std::string

get_slave_timecode_offset()
unsigned int

get_subframes_per_frame()
std::string

get_take_name()
TimecodeFormat

get_timecode_format()
std::string

get_timecode_generator_offset()
long

get_timecode_offset()
bool

get_timecode_offset_negative()
bool

get_track_name_number()
bool

get_track_name_take()
```

```
bool
get_use_monitor_fades()

bool
get_use_region_fades()

bool
get_use_transport_fades()

bool
get_use_video_file_fps()

bool
get_use_video_sync()

float
get_video_pullup()

bool
get_videotimeline_pullup()

double
get_wave_amplitude_zoom()

unsigned short
get_wave_zoom_factor()

bool
set_audio_search_path(std::string)

bool
set_auto_input(bool)

bool
set_auto_play(bool)

bool
set_auto_return(bool)

bool
set_count_in(bool)

bool
set_destructive_xfade_msecs(unsigned int)

bool
set_external_sync(bool)

bool
set_glue_new_markers_to_bars_and_beats(bool)

bool
set_glue_new_regions_to_bars_and_beats(bool)
```

```
bool
set_insert_merge_policy(InsertMergePolicy)
bool
set_jack_time_master(bool)
bool
set_layered_record_mode(bool)
bool
set_meterbridge_label_height(unsigned int)
bool
set_midi_copy_is_fork(bool)
bool
set_midi_search_path(std::string)
bool
set_minitimeline_span(long)
bool
set_native_file_data_format(SampleFormat)
bool
set_native_file_header_format(HeaderFormat)
bool
set_punch_in(bool)
bool
set_punch_out(bool)
bool
set_raid_path(std::string)
bool
set_realtime_export(bool)
bool
set_session_monitoring(MonitorChoice)
bool
set_show_busses_on_meterbridge(bool)
bool
set_show_group_tabs(bool)
bool
set_show_master_on_meterbridge(bool)
bool
set_show_midi_on_meterbridge(bool)
```

```
bool
set_show_monitor_on_meterbridge(bool)
bool
set_show_mute_on_meterbridge(bool)
bool
set_show_name_on_meterbridge(bool)
bool
set_show_rec_on_meterbridge(bool)
bool
set_show_region_fades(bool)
bool
set_show_solo_on_meterbridge(bool)
bool
set_show_summary(bool)
bool
set_slave_timecode_offset(std::string)
bool
set_subframes_per_frame(unsigned int)
bool
set_take_name(std::string)
bool
set_timecode_format(TimecodeFormat)
bool
set_timecode_generator_offset(std::string)
bool
set_timecode_offset(long)
bool
set_timecode_offset_negative(bool)
bool
set_track_name_number(bool)
bool
set_track_name_take(bool)
bool
set_use_monitor_fades(bool)
bool
set_use_region_fades(bool)
```

```
bool
set_use_transport_fades(bool)
bool
set_use_video_file_fps(bool)
bool
set_use_video_sync(bool)
bool
set_video_pullup(float)
bool
set_videotimeline_pullup(bool)
bool
set_wave_amplitude_zoom(double)
bool
set_wave_zoom_factor(unsigned short)

Properties

std::string
audio_search_path
bool
auto_input
bool
auto_play
bool
auto_return
bool
count_in
unsigned int
destructive_xfade_msecs
bool
external_sync
bool
glue_new_markers_to_bars_and_beats
bool
glue_new_regions_to_bars_and_beats

ARDOUR.InsertMergePolicy
insert_merge_policy
bool
```

```
jack_time_master
bool
layered_record_mode
unsigned int
meterbridge_label_height
bool
midi_copy_is_fork
std::string
midi_search_path
long
minitimeline_span
ARDOUR.SampleFormat
native_file_data_format
ARDOUR.HeaderFormat
native_file_header_format
bool
punch_in
bool
punch_out
std::string
raid_path
bool
realtime_export
ARDOUR.MonitorChoice
session_monitoring
bool
show_busses_on_meterbridge
bool
show_group_tabs
bool
show_master_on_meterbridge
bool
show_midi_on_meterbridge
bool
show_monitor_on_meterbridge
bool
```

```
show_mute_on_meterbridge
bool
show_name_on_meterbridge
bool
show_rec_on_meterbridge
bool
show_region_fades
bool
show_solo_on_meterbridge
bool
show_summary
std::string
slave_timecode_offset
unsigned int
subframes_per_frame
std::string
take_name
Timecode.TimecodeFormat
timecode_format
std::string
timecode_generator_offset
long
timecode_offset
bool
timecode_offset_negative
bool
track_name_number
bool
track_name_take
bool
use_monitor_fades
bool
use_region_fades
bool
use_transport_fades
bool
```

```
use_video_file_fps
bool
use_video_sync
float
video_pullup
bool
videotimeline_pullup
double
wave_amplitude_zoom
unsigned short
wave_zoom_factor
```

### 152.111.1 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.112 ARDOUR:SessionObject

*C*#: ARDOUR::SessionObject

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

## 152.113 ARDOUR:SessionObjectPtr

*C++:* boost::shared\_ptr< ARDOUR::SessionObject >, boost::weak\_ptr< ARDOUR::SessionObject >

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

bool

isnil()

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.114 ARDOUR:SideChain

*C++:* boost::shared\_ptr< ARDOUR::SideChain >, boost::weak\_ptr< ARDOUR::SideChain >

is-a: *ARDOUR:IOProcessor*

A mixer strip element (Processor) with 1 or 2 IO elements.

Methods

bool

isnil()

### 152.114.1 Inherited from ARDOUR:IOProcessor

Methods

*IO*

input()

*ChanCount*

natural\_input\_streams()

*ChanCount*

natural\_output\_streams()

*IO*

output()

## 152.114.2 Inherited from ARDOUR:Processor

Methods

void

activate()

bool

active()

void

deactivate()

std::string

display\_name()

bool

display\_to\_user()

*ChanCount*

input\_streams()

*ChanCount*

output\_streams()

Cast

*Amp*

to\_amp()

*Automatable*

to\_automatable()

*PluginInsert*

to\_insert()

*IOProcessor*

to\_ioprocessor()

*PeakMeter*

to\_meter()

*MonitorProcessor*

to\_monitorprocessor()

*PeakMeter*

to\_peakmeter()

*PluginInsert*

to\_plugininsert()

*SideChain*

to\_sidechain()

*UnknownProcessor*

to\_unknownprocessor()

### 152.114.3 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.115 ARDOUR:Slavable

*Cf:* boost::shared\_ptr< ARDOUR::Slavable >, boost::weak\_ptr< ARDOUR::Slavable >

Methods

void

assign( *VCA* )

bool

isnil()

void

unassign( *VCA* )

## 152.116 ARDOUR:SlavableAutomationControl

*Cf:* boost::shared\_ptr< ARDOUR::SlavableAutomationControl >, boost::weak\_ptr< ARDOUR::SlavableAutomationControl >

is-a: *ARDOUR:AutomationControl*

A PBD::Controllable with associated automation data (AutomationList)

Methods

void

add\_master(*AutomationControl*)

void

clear\_masters()

int

get\_boolean\_masters()

double

```
get_masters_value()
bool
isnil()
void
remove_master(AutomationControl)
bool
slaved()
bool
slaved_to(AutomationControl)
```

### 152.116.1 Inherited from ARDOUR:AutomationControl

Methods

*AutomationList*

alist()

*AutoState*

automation\_state()

double

get\_value()

Get the current effective ‘user’ value based on automation state

void

set\_automation\_state(*AutoState*)

void

set\_value(double, *GroupControlDisposition*)

Get and Set ‘internal’ value

All derived classes must implement this.

Basic derived classes will ignore

**group\_override**, but more sophisticated children, notably those that proxy the value setting logic via an object that is aware of group relationships between this control and others, will find it useful.

void

start\_touch(double)

void

stop\_touch(double)

bool

writable()

Cast

*Control*

to\_ctrl()  
*SlavableAutomationControl*  
to\_slavable()

### 152.116.2 Inherited from PBD:Controllable

Methods  
std::string  
name()

### 152.116.3 Inherited from PBD:StatefulPtr

Methods  
void  
clear\_changes()  
Forget about any changes to this object's properties  
*ID*  
id()  
*OwnedPropertyList*  
properties()

## 152.117 ARDOUR:SoloControl

*C#:* boost::shared\_ptr< ARDOUR::SoloControl >, boost::weak\_ptr< ARDOUR::SoloControl >  
is-a: *ARDOUR:SlavableAutomationControl*

A PBD::Controllable with associated automation data (AutomationList)

Methods  
bool  
can\_solo()  
bool  
isnil()  
bool  
self\_soloed()  
bool  
soloed()

### 152.117.1 Inherited from ARDOUR:SlvableAutomationControl

Methods

void  
add\_master(*AutomationControl*)  
void  
clear\_masters()  
int  
get\_boolean\_masters()  
double  
get\_masters\_value()  
void  
remove\_master(*AutomationControl*)  
bool  
slaved()  
bool  
slaved\_to(*AutomationControl*)

### 152.117.2 Inherited from ARDOUR:AutomationControl

Methods

*AutomationList*  
alist()  
*AutoState*  
automation\_state()  
double  
get\_value()  
Get the current effective ‘user’ value based on automation state  
void  
set\_automation\_state(*AutoState*)  
void  
set\_value(double, *GroupControlDisposition*)  
Get and Set ‘internal’ value  
All derived classes must implement this.  
Basic derived classes will ignore  
**group\_override**, but more sophisticated children, notably those that proxy the value setting logic via an object that is aware of group relationships between this control and others, will find it useful.

void  
start\_touch(double)  
void  
stop\_touch(double)  
bool  
writable()  
Cast  
*Control*  
to\_ctrl()  
*SlavableAutomationControl*  
to\_slavable()

### 152.117.3 Inherited from PBD:Controllable

Methods  
std::string  
name()

### 152.117.4 Inherited from PBD:StatefulPtr

Methods  
void  
clear\_changes()  
Forget about any changes to this object's properties  
*ID*  
id()  
*OwnedPropertyList*  
properties()

## 152.118 ARDOUR:SoloIsolateControl

*C++:* boost::shared\_ptr< ARDOUR::SoloIsolateControl >, boost::weak\_ptr< ARDOUR::SoloIsolateControl >

is-a: *ARDOUR:SlavableAutomationControl*

A PBD::Controllable with associated automation data (AutomationList)

Methods

bool  
isnil()

```
bool  
self_solo_isolated()  
bool  
solo_isolated()
```

### 152.118.1 Inherited from ARDOUR:SlavableAutomationControl

Methods

```
void  
add_master(AutomationControl)  
void  
clear_masters()  
int  
get_boolean_masters()  
double  
get_masters_value()  
void  
remove_master(AutomationControl)  
bool  
slaved()  
bool  
slaved_to(AutomationControl)
```

### 152.118.2 Inherited from ARDOUR:AutomationControl

Methods

```
AutomationList  
alist()  
AutoState  
automation_state()  
double  
get_value()  
Get the current effective ‘user’ value based on automation state  
void  
set_automation_state(AutoState)  
void  
set_value(double, GroupControlDisposition)  
Get and Set ‘internal’ value
```

All derived classes must implement this.

Basic derived classes will ignore

**group\_override**, but more sophisticated children, notably those that proxy the value setting logic via an object that is aware of group relationships between this control and others, will find it useful.

void

start\_touch(double)

void

stop\_touch(double)

bool

writable()

Cast

*Control*

to\_ctrl()

*SlavableAutomationControl*

to\_slavable()

### 152.118.3 Inherited from PBD:Controllable

Methods

std::string

name()

### 152.118.4 Inherited from PBD:StatefulPtr

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.119 ARDOUR:SoloSafeControl

*C $\ddagger$* : boost::shared\_ptr< ARDOUR::SoloSafeControl >, boost::weak\_ptr< ARDOUR::SoloSafeControl >

is-a: *ARDOUR:SlavableAutomationControl*

A PBD::Controllable with associated automation data (AutomationList)

Methods

bool

isnil()

bool

solo\_safe()

### 152.119.1 Inherited from ARDOUR:SlavableAutomationControl

Methods

void

add\_master(*AutomationControl*)

void

clear\_masters()

int

get\_boolean\_masters()

double

get\_masters\_value()

void

remove\_master(*AutomationControl*)

bool

slaved()

bool

slaved\_to(*AutomationControl*)

### 152.119.2 Inherited from ARDOUR:AutomationControl

Methods

*AutomationList*

alist()

*AutoState*

automation\_state()

double

get\_value()

Get the current effective ‘user’ value based on automation state

void

set\_automation\_state(*AutoState*)

void

set\_value(double, *GroupControlDisposition*)

Get and Set ‘internal’ value

All derived classes must implement this.

Basic derived classes will ignore

**group\_override**, but more sophisticated children, notably those that proxy the value setting logic via an object that is aware of group relationships between this control and others, will find it useful.

void

start\_touch(double)

void

stop\_touch(double)

bool

writable()

Cast

*Control*

to\_ctrl()

*SlavableAutomationControl*

to\_slavable()

### 152.119.3 Inherited from PBD:Controllable

Methods

std::string

name()

### 152.119.4 Inherited from PBD:StatefulPtr

Methods

void

clear\_changes()

Forget about any changes to this object’s properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.120 ARDOUR:Source

*Cf:* boost::shared\_ptr< ARDOUR::Source >, boost::weak\_ptr< ARDOUR::Source >

is-a: *ARDOUR:SessionObjectPtr*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

std::string

ancestor\_name()

bool

can\_be\_analysed()

bool

destructive()

bool

empty()

bool

has\_been\_analysed()

bool

isnil()

long

length(long)

long

natural\_position()

long

timeline\_position()

long

timestamp()

int

use\_count()

bool

used()

bool

writable()

Cast

*AudioSource*

to\_audiosource()

*FileSource*

to\_filesource()

*MidiSource*

to\_midisource()

### 152.120.1 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()

*StatefulDestructible*

to\_statefuldestructible()

## 152.121 ARDOUR:SourceList

*C*#: std::vector<boost::shared\_ptr<ARDOUR::Source>>

Constructor

ARDOUR.SourceList()

ARDOUR.SourceList()

Methods

*LuaTable*

add(LuaTable {*Source*})

*Source*

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(*Source*)

unsigned long

size()

*LuaTable*

table()

## 152.122 ARDOUR:Stripable

*C<sub>++</sub>:* boost::shared\_ptr< ARDOUR::Stripable >, boost::weak\_ptr< ARDOUR::Stripable >

is-a: *ARDOUR:SessionObjectPtr*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods

*AutomationControl*

comp\_enable\_control()

*AutomationControl*

comp\_makeup\_control()

*AutomationControl*

comp\_mode\_control()

std::string

comp\_mode\_name(unsigned int)

*ReadOnlyControl*

comp\_redux\_control()

*AutomationControl*

comp\_speed\_control()

std::string

comp\_speed\_name(unsigned int)

*AutomationControl*

comp\_threshold\_control()

unsigned int

eq\_band\_cnt()

std::string

eq\_band\_name(unsigned int)

*AutomationControl*

eq\_enable\_control()

*AutomationControl*

eq\_freq\_control(unsigned int)

*AutomationControl*

eq\_gain\_control(unsigned int)

*AutomationControl*

eq\_q\_control(unsigned int)

*AutomationControl*

eq\_shape\_control(unsigned int)

*AutomationControl*

filter\_enable\_controllable(bool)

*AutomationControl*

filter\_freq\_controllable(bool)

*AutomationControl*

filter\_slope\_controllable(bool)

*GainControl*

gain\_control()

bool

is\_auditioner()

bool

is\_hidden()

bool

is\_master()

bool

is\_monitor()

bool

is\_selected()

bool

isnil()

*AutomationControl*

master\_send\_enable\_control()

*MonitorProcessor*

monitor\_control()

*MuteControl*

mute\_control()

*AutomationControl*

pan\_azimuth\_control()

*AutomationControl*

pan\_elevation\_control()

*AutomationControl*

pan\_frontback\_control()

*AutomationControl*

pan\_lfe\_control()

*AutomationControl*

pan\_width\_control()

*PhaseControl*  
phase\_control()  
*PresentationInfo*  
presentation\_info\_ptr()  
*AutomationControl*  
rec\_enable\_control()  
*AutomationControl*  
rec\_safe\_control()  
*AutomationControl*  
send\_enable\_control(unsigned int)  
*AutomationControl*  
send\_level\_control(unsigned int)  
std::string  
send\_name(unsigned int)  
void  
set\_presentation\_order(unsigned int)  
*SoloControl*  
solo\_control()  
*SoloIsolateControl*  
solo\_isolate\_control()  
*SoloSafeControl*  
solo\_safe\_control()  
*GainControl*  
trim\_control()  
Cast  
*Route*  
to\_route()  
*VCA*  
to\_vca()

### 152.122.1 Inherited from ARDOUR:SessionObjectPtr

Methods

std::string

name()

Cast

*Stateful*

to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.123 ARDOUR:StripableList

*C#:* std::list<boost::shared\_ptr<ARDOUR::Stripable>>  
Constructor

ARDOUR.StripableList()

Methods

*Stripable*

back()

bool

empty()

*Stripable*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.124 ARDOUR:Tempo

*C#:* ARDOUR::Tempo  
Tempo, the speed at which musical time progresses (BPM).  
Constructor

ARDOUR.Tempo(double, double, double)

Methods

double

frames\_per\_note\_type(long)

audio samples per note type. if you want an instantaneous value for this, use TempoMap::frames\_per\_quarter\_note\_at() instead.

**sr** samplerate

double

frames\_per\_quarter\_note(long)

audio samples per quarter note. if you want an instantaneous value for this, use TempoMap::frames\_per\_quarter\_note\_at() instead.

**sr** samplerate

double

note\_type()

double

note\_types\_per\_minute()

double

quarter\_notes\_per\_minute()

## 152.125 ARDOUR:TempoMap

*C<sup>++</sup>*: ARDOUR::TempoMap

Tempo Map - mapping of timecode to musical time. convert audio-samples, sample-rate to Bar/Beat/Tick, Meter/Tempo

Methods

*MeterSection*

add\_meter(*Meter*,*BBT\_TIME*, long,*PositionLockStyle*)

*TempoSection*

add\_tempo(*Tempo*, double, long,*PositionLockStyle*)

*BBT\_TIME*

bbt\_at\_frame(long)

Returns the BBT time corresponding to the supplied frame position.

**frame** the position in audio samples.

Returns the BBT time at the frame position .

double

exact\_beat\_at\_frame(long, int)

double

exact\_qn\_at\_frame(long, int)

long

framepos\_plus\_qn(long, *Beats*)

Add some (fractional) Beats to a session frame position, and return the result in frames. pos can be -ve, if required.

*Beats*`framewalk_to_qn(long, long)`

Count the number of beats that are equivalent to distance when going forward, starting at pos.

*MeterSection*`meter_section_at_beat(double)`*MeterSection*`meter_section_at_frame(long)`*TempoSection*`tempo_section_at_frame(long)`*TempoSection*`tempo_section_at_frame(long)`

## 152.126 ARDOUR:TempoSection

*C<sub>r</sub>t*: ARDOUR::TempoSection

is-a: *ARDOUR:MetricSection*

A section of timeline with a certain Tempo.

Methods

double

c()

### 152.126.1 Inherited from ARDOUR:MetricSection

Methods

double

pulse()

void

set\_pulse(double)

## 152.127 ARDOUR:Track

*C<sub>r</sub>t*: boost::shared\_ptr< ARDOUR::Track >, boost::weak\_ptr< ARDOUR::Track >

is-a: *ARDOUR:Route*

A track is an route (bus) with a recordable diskstream and related objects relevant to tracking, playback and editing.

Specifically a track has regions and playlist objects.

Methods

*Region*

bounce(*InterThreadInfo&*)

bounce track from session start to session end to new region

**itt** asynchronous progress report and cancel

Returns a new audio region (or nil in case of error)

*Region*

bounce\_range(long, long, *InterThreadInfo&*, *Processor*, bool)

Bounce the given range to a new audio region.

**start** start time (in samples)

**end** end time (in samples)

**itt** asynchronous progress report and cancel

**endpoint** the processor to tap the signal off (or nil for the top)

**include\_endpoint** include the given processor in the bounced audio.

Returns a new audio region (or nil in case of error)

bool

bounceable(*Processor*, bool)

Test if the track can be bounced with the given settings. If sends/inserts/returns are present in the signal path or the given track has no audio outputs bouncing is not possible.

**endpoint** the processor to tap the signal off (or nil for the top)

**include\_endpoint** include the given processor in the bounced audio.

Returns true if the track can be bounced, or false otherwise.

bool

can\_record()

bool

isnil()

*Playlist*

playlist()

bool

set\_name(std::string)

Cast

*AudioTrack*

to\_audio\_track()

*MidiTrack*

to\_midi\_track()

## 152.127.1 Inherited from ARDOUR:Route

Methods

bool

active()

int

`add_processor_by_index(Processor, int, ProcessorStreams, bool)`

Add a processor to a route such that it ends up with a given index into the visible processors.

**index** Index to add the processor at, or -1 to add at the end of the list.

Returns 0 on success, non-0 on failure.

bool

`add_sidechain(Processor)`

*Amp*

`amp()`

std::string

`comment()`

bool

`customize_plugin_insert(Processor, unsigned int, ChanCount, ChanCount)`

enable custom plugin-insert configuration

**proc** Processor to customize

**count** number of plugin instances to use (if zero, reset to default)

**outs** output port customization

**sinks** input pins for variable-I/O plugins

Returns true if successful

*IO*

`input()`

*Delivery*

`main_outs()`

the signal processor at end of the processing chain which produces output

bool

`muted()`

*ChanCount*

`n_inputs()`

*ChanCount*

`n_outputs()`

*Processor*

`nth_plugin(unsigned int)`

*Processor*

nth\_processor(unsigned int)

*Processor*

nth\_send(unsigned int)

*IO*

output()

*PannerShell*

panner\_shell()

*PeakMeter*

peak\_meter()

\*\*\*\*\* Pure interface begins  
here\*\*\*\*\*

int

remove\_processor(*Processor*,*ProcessorStreams*, bool)

remove plugin/processor

**proc** processor to remove

**err** error report (index where removal failed, channel-count why it failed) may be nil

**need\_process\_lock** if locking is required (set to true, unless called from RT context with lock)

Returns 0 on success

int

remove\_processors(*ProcessorList*,*ProcessorStreams*)

bool

remove\_sidechain(*Processor*)

int

reorder\_processors(*ProcessorList*,*ProcessorStreams*)

int

replace\_processor(*Processor*,*Processor*,*ProcessorStreams*)

replace plugin/processor with another

**old** processor to remove

**sub** processor to substitute the old one with

**err** error report (index where removal failed, channel-count why it failed) may be nil

Returns 0 on success

bool

reset\_plugin\_insert(*Processor*)

reset plugin-insert configuration to default, disable customizations.

This is equivalent to calling

```
customize_plugin_insert (proc, 0, unused)
```

**proc** Processor to reset

Returns true if successful

void

set\_active(bool, void\*)

void

set\_comment(std::string, void\*)

void

set\_meter\_point(*MeterPoint*, bool)

bool

set\_strict\_io(bool)

bool

soloed()

bool

strict\_io()

*Processor*

the\_instrument()

Return the first processor that accepts has at least one MIDI input and at least one audio output. In the vast majority of cases, this will be “the instrument”. This does not preclude other MIDI->audio processors later in the processing chain, but that would be a special case not covered by this utility function.

*Amp*

trim()

Cast

*Automatable*

to\_automatable()

*Slavable*

to\_slavable()

*Track*

to\_track()

## 152.127.2 Inherited from ARDOUR:Stripable

Methods

*AutomationControl*

comp\_enable\_control()

*AutomationControl*

comp\_makeup\_control()

*AutomationControl*

comp\_mode\_control()

std::string

comp\_mode\_name(unsigned int)

*ReadOnlyControl*

comp\_redux\_control()

*AutomationControl*

comp\_speed\_control()

std::string

comp\_speed\_name(unsigned int)

*AutomationControl*

comp\_threshold\_control()

unsigned int

eq\_band\_cnt()

std::string

eq\_band\_name(unsigned int)

*AutomationControl*

eq\_enable\_control()

*AutomationControl*

eq\_freq\_control(unsigned int)

*AutomationControl*

eq\_gain\_control(unsigned int)

*AutomationControl*

eq\_q\_control(unsigned int)

*AutomationControl*

eq\_shape\_control(unsigned int)

*AutomationControl*

filter\_enable\_controllable(bool)

*AutomationControl*

filter\_freq\_controllable(bool)

*AutomationControl*

filter\_slope\_controllable(bool)

*GainControl*

gain\_control()

bool

is\_auditioner()

```
bool  
is_hidden()  
bool  
is_master()  
bool  
is_monitor()  
bool  
is_selected()  
AutomationControl  
master_send_enable_control()  
MonitorProcessor  
monitor_control()  
MuteControl  
mute_control()  
AutomationControl  
pan_azimuth_control()  
AutomationControl  
pan_elevation_control()  
AutomationControl  
pan_frontback_control()  
AutomationControl  
pan_lfe_control()  
AutomationControl  
pan_width_control()  
PhaseControl  
phase_control()  
PresentationInfo  
presentation_info_ptr()  
AutomationControl  
rec_enable_control()  
AutomationControl  
rec_safe_control()  
AutomationControl  
send_enable_control(unsigned int)  
AutomationControl  
send_level_control(unsigned int)
```

```
std::string  
send_name(unsigned int)  
void  
set_presentation_order(unsigned int)  
SoloControl  
solo_control()  
SoloIsolateControl  
solo_isolate_control()  
SoloSafeControl  
solo_safe_control()  
GainControl  
trim_control()  
Cast  
Route  
to_route()  
VCA  
to_vca()
```

### 152.127.3 Inherited from ARDOUR:SessionObjectPtr

Methods

```
std::string  
name()  
Cast  
Stateful  
to_stateful()  
StatefulDestructible  
to_statefuldestructible()
```

## 152.128 ARDOUR:UnknownProcessor

*C#:* boost::shared\_ptr< ARDOUR::UnknownProcessor >, boost::weak\_ptr< ARDOUR::UnknownProcessor >  
is-a: *ARDOUR:Processor*

A stub Processor that can be used in place of a ‘real’ one that cannot be created for some reason; usually because it requires a plugin which is not present. UnknownProcessors are special-cased in a few places, notably in route configuration and signal processing, so that on encountering them configuration or processing stops.

When a Processor is missing from a Route, the following processors cannot be configured, as the missing Processor's output port configuration is unknown.

The main utility of the UnknownProcessor is that it allows state to be preserved, so that, for example, loading and re-saving a session on a machine without a particular plugin will not corrupt the session.

Methods

bool

isnil()

### 152.128.1 Inherited from ARDOUR:Processor

Methods

void

activate()

bool

active()

void

deactivate()

std::string

display\_name()

bool

display\_to\_user()

*ChanCount*

input\_streams()

*ChanCount*

output\_streams()

Cast

*Amp*

to\_amp()

*Automatable*

to\_automatable()

*PluginInsert*

to\_insert()

*IOProcessor*

to\_ioprocessor()

*PeakMeter*

to\_meter()

*MonitorProcessor*

to\_monitorprocessor()

*PeakMeter*  
to\_peakmeter()  
*PluginInsert*  
to\_plugininsert()  
*SideChain*  
to\_sidechain()  
*UnknownProcessor*  
to\_unknownprocessor()

## 152.128.2 Inherited from ARDOUR:SessionObjectPtr

Methods  
std::string  
name()  
Cast  
*Stateful*  
to\_stateful()  
*StatefulDestructible*  
to\_statefuldestructible()

## 152.129 ARDOUR:VCA

*C<sub>t</sub>*: boost::shared\_ptr< ARDOUR::VCA >, boost::weak\_ptr< ARDOUR::VCA >  
is-a: *ARDOUR:Stripable*

A named object associated with a Session. Objects derived from this class are expected to be destroyed before the session calls drop\_references().

Methods  
std::string  
full\_name()  
*GainControl*  
gain\_control()  
bool  
isnil()  
*MuteControl*  
mute\_control()  
int  
number()

*SoloControl*

solo\_control()

### 152.129.1 Inherited from ARDOUR:Stripable

Methods

*AutomationControl*

comp\_enable\_control()

*AutomationControl*

comp\_makeup\_control()

*AutomationControl*

comp\_mode\_control()

std::string

comp\_mode\_name(unsigned int)

*ReadOnlyControl*

comp\_redux\_control()

*AutomationControl*

comp\_speed\_control()

std::string

comp\_speed\_name(unsigned int)

*AutomationControl*

comp\_threshold\_control()

unsigned int

eq\_band\_cnt()

std::string

eq\_band\_name(unsigned int)

*AutomationControl*

eq\_enable\_control()

*AutomationControl*

eq\_freq\_control(unsigned int)

*AutomationControl*

eq\_gain\_control(unsigned int)

*AutomationControl*

eq\_q\_control(unsigned int)

*AutomationControl*

eq\_shape\_control(unsigned int)

*AutomationControl*

filter\_enable\_controllable(bool)

*AutomationControl*

filter\_freq\_controllable(bool)

*AutomationControl*

filter\_slope\_controllable(bool)

bool

is\_auditioner()

bool

is\_hidden()

bool

is\_master()

bool

is\_monitor()

bool

is\_selected()

*AutomationControl*

master\_send\_enable\_control()

*MonitorProcessor*

monitor\_control()

*AutomationControl*

pan\_azimuth\_control()

*AutomationControl*

pan\_elevation\_control()

*AutomationControl*

pan\_frontback\_control()

*AutomationControl*

pan\_lfe\_control()

*AutomationControl*

pan\_width\_control()

*PhaseControl*

phase\_control()

*PresentationInfo*

presentation\_info\_ptr()

*AutomationControl*

rec\_enable\_control()

*AutomationControl*

```
rec_safe_control()  
AutomationControl  
send_enable_control(unsigned int)  
AutomationControl  
send_level_control(unsigned int)  
std::string  
send_name(unsigned int)  
void  
set_presentation_order(unsigned int)  
SoloIsolateControl  
solo_isolate_control()  
SoloSafeControl  
solo_safe_control()  
GainControl  
trim_control()  
Cast  
Route  
to_route()  
VCA  
to_vca()
```

## 152.129.2 Inherited from ARDOUR:SessionObjectPtr

```
Methods  
std::string  
name()  
Cast  
Stateful  
to_stateful()  
StatefulDestructible  
to_statefuldestructible()
```

## 152.130 ARDOUR:VCAList

*C++:* std::list<boost::shared\_ptr<ARDOUR::VCA>>  
Constructor

ARDOUR.VCAList()

Methods

*VCA*

back()

bool

empty()

*VCA*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.131 ARDOUR:VCAManager

*C<sub>++</sub>*: ARDOUR::VCAManageris-a: *PBD:StatefulDestructible*

Base class for objects with saveable and undoable state with destruction notification

Methods

int

create\_vca(unsigned int, std::string)

unsigned long

n\_vcás()

void

remove\_vca(*VCA*)*VCA*

vca\_by\_name(std::string)

*VCA*

vca\_by\_number(int)

*VCAList*

vcás()

### 152.131.1 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.132 ARDOUR:Weak AudioSourceList

*C++:* std::list<boost::weak\_ptr<ARDOUR:: AudioSource>>

Constructor

ARDOUR.Weak AudioSourceList()

Methods

back()

bool

empty()

*AudioSource*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.133 ARDOUR:Weak RouteList

*C++:* std::list<boost::weak\_ptr<ARDOUR::Route>>

Constructor

ARDOUR.WeakRouteList()

Methods

*Route*

back()

bool

empty()

*Route*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.134 ARDOUR:WeakSourceList

*C<sub>++</sub>*: std::list<boost::weak\_ptr<ARDOUR::Source> >

Constructor

ARDOUR.WeakSourceList()

Methods

*Source*

back()

bool

empty()

*Source*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()  
*LuaTable*  
table()

## 152.135 ArdourUI

Methods  
std::string  
http\_get(std::string)  
*ProcessorVector*  
processor\_selection()  
unsigned int  
translate\_order(*InsertAt*)

## 152.136 ArdourUI:ArdourMarker

*C<sub>7</sub>*: ArdourMarker  
Location Marker  
Editor ruler representation of a location marker or range on the timeline.  
Methods  
std::string  
name()  
long  
position()  
*Type*  
type()

## 152.137 ArdourUI:ArdourMarkerList

*C<sub>7</sub>*: std::list<ArdourMarker\* >  
Constructor

ArdourUI.ArdourMarkerList()  
Methods  
*LuaTable*  
add(ArdourMarker\*)  
*ArdourMarker*

back()  
bool  
empty()  
*ArdourMarker*  
front()  
*LuaIter*  
iter()  
void  
push\_back(*ArdourMarker*)  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()  
void  
unique()

## 152.138 ArdourUI:AxisView

*C++:* AxisView

AxisView defines the abstract base class for horizontal and vertical presentations of Stripables.

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.139 ArdourUI:Editor

*C++:* PublicEditor

This class contains just the public interface of the Editor class, in order to decouple it from the private implementation, so that callers of PublicEditor need not be recompiled if private methods or member variables change.

Methods  
void  
access\_action(std::string, std::string)  
void  
add\_location\_from\_playhead\_cursor()  
*TrackViewList*

```
axis_views_from_routes(RouteListPtr)
void
center_screen(long)
void
clear_playlist(Playlist)
void
clear_playlists(TimeAxisView)
void
consider_auditioning(Region)
Possibly start the audition of a region. If
r is 0, or not an AudioRegion any current audition is cancelled. If we are currently auditioning
r will start.
r Region to consider.
r, the audition will be cancelled. Otherwise an audition of
void
copy_playlists(TimeAxisView)
MouseMode
current_mouse_mode()
Returns The current mouse mode (gain, object, range, timefx etc.) (defined in editing_syms.h)
long
current_page_samples()
void
deselect_all()
LuaTable(...)
do_embed(StringVector,ImportDisposition,ImportMode, long&,PluginInfo)
LuaTable(...)
do_import(StringVector,ImportDisposition,ImportMode,SrcQuality,MidiTrackNameSource,MidiTempoMapDisposition, long&,PluginInfo)
Import existing media
bool
dragging_playhead()
Returns true if the playhead is currently being dragged, otherwise false
MouseMode
effective_mouse_mode()
void
export_audio()
```

Open main export dialog  
void  
export\_range()  
Open export dialog with current range pre-selected  
void  
export\_selection()  
Open export dialog with current selection pre-selected  
*LuaTable(Location, ...)*  
find\_location\_from\_marker(*ArdourMarker*, bool&)  
*ArdourMarker*  
find\_marker\_from\_location\_id(*ID*, bool)  
void  
fit\_selection()  
bool  
follow\_playhead()  
Returns true if the editor is following the playhead  
long  
get\_current\_zoom()  
*Selection*  
get\_cut\_buffer()  
unsigned int  
get\_grid\_beat\_divisions(long)  
*LuaTable(Beats, ...)*  
get\_grid\_type\_as\_beats(bool&, long)  
*LuaTable(long, ...)*  
get\_nudge\_distance(long, long&)  
long  
get\_paste\_offset(long, unsigned int, long)  
*LuaTable(...)*  
get\_pointer\_position(double&, double&)  
*Selection*  
get\_selection()  
*LuaTable(bool, ...)*  
get\_selection\_extents(long&, long&)  
bool  
get\_smart\_mode()

*StripableTimeAxisView*

get\_stripable\_time\_axis\_by\_id(*ID*)

*TrackViewList*

get\_track\_views()

int

get\_videotl\_bar\_height()

double

get\_y\_origin()

*ZoomFocus*

get\_zoom\_focus()

void

goto\_nth\_marker(int)

void

hide\_track\_in\_display(*TimeAxisView*, bool)

long

leftmost\_sample()

void

maximise\_editing\_space()

void

maybe\_locate\_with\_edit\_preroll(long)

void

mouse\_add\_new\_marker(long, bool)

void

new\_playlists(*TimeAxisView*)

void

new\_region\_from\_selection()

void

override\_visible\_track\_count()

long

pixel\_to\_sample(double)

void

play\_selection()

void

play\_with\_preroll()

void

redo(unsigned int)

Redo some transactions.

**n** Number of transaction to redo.

*RegionView*

regionview\_from\_region(*Region*)

void

remove\_last\_capture()

void

remove\_location\_at\_playhead\_cursor()

void

remove\_tracks()

void

reset\_x\_origin(long)

void

reset\_y\_origin(double)

void

reset\_zoom(long)

void

restore\_editing\_space()

*RouteTimeAxisView*

rtav\_from\_route(*Route*)

double

sample\_to\_pixel(long)

bool

scroll\_down\_one\_track(bool)

void

scroll\_tracks\_down\_line()

void

scroll\_tracks\_up\_line()

bool

scroll\_up\_one\_track(bool)

void

select\_all\_tracks()

void

separate\_region\_from\_selection()

void

set\_follow\_playhead(bool, bool)

Set whether the editor should follow the playhead.

**yn** true to follow playhead, otherwise false.

**catch\_up** true to reset the editor view to show the playhead (if yn == true), otherwise false.

void

set\_loop\_range(long, long, std::string)

void

set\_mouse\_mode(*MouseMode*, bool)

Set the mouse mode (gain, object, range, timefx etc.)

**m** Mouse mode (defined in editing\_syms.h)

**force** Perform the effects of the change even if no change is required (ie even if the current mouse mode is equal to

void

set\_punch\_range(long, long, std::string)

void

set\_selection(*SelectionList*, *Operation*)

void

set\_show\_measures(bool)

void

set\_snap\_mode(*SnapMode*)

Set the snap mode.

**m** Snap mode (defined in editing\_syms.h)

void

set\_snap\_threshold(double)

Set the snap threshold.

**t** Snap threshold in ‘units’.

void

set\_stationary\_playhead(bool)

void

set\_toggleaction(std::string, std::string, bool)

void

set\_video\_timeline\_height(int)

void

set\_visible\_track\_count(int)

void

set\_zoom\_focus(*ZoomFocus*)

bool

```
show_measures()
void
show_track_in_display(TimeAxisView, bool)
SnapMode
snap_mode()
SnapType
snap_type()
bool
stationary_playhead()
void
stem_export()
Open stem export dialog
void
temporal_zoom_step(bool)
void
toggle_meter_updating()
void
toggle_ruler_video(bool)
void
toggle_xjadeo_proc(int)
void
undo(unsigned int)
Undo some transactions.
n Number of transactions to undo.
double
visible_canvas_height()
```

## 152.140 ArdourUI:MarkerSelection

*C<sub>7</sub>*: MarkerSelection

is-a: *ArdourUI:ArdourMarkerList*

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

### 152.140.1 Inherited from ArdourUI:ArdourMarkerList

Constructor

ArdourUI.ArdourMarkerList()

Methods

*LuaTable*

add(ArdourMarker\*)

*ArdourMarker*

back()

bool

empty()

*ArdourMarker*

front()

*LuaIter*

iter()

void

push\_back(*ArdourMarker*)

void

reverse()

unsigned long

size()

*LuaTable*

table()

void

unique()

### 152.141 ArdourUI:RegionSelection

*C<sub>7</sub>*: RegionSelection

Class to represent list of selected regions.

Methods

long

end\_frame()

unsigned long

n\_midi\_regions()

*RegionList*

regionlist()  
long  
start()

## 152.142 ArdourUI:RegionView

*C<sup>#</sup>*: RegionView  
is-a: *ArdourUI:TimeAxisViewItem*

Base class for items that may appear upon a TimeAxisView.

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.143 ArdourUI:RouteTimeAxisView

*C<sup>#</sup>*: RouteTimeAxisView  
is-a: *ArdourUI:RouteUI*

Base class for objects with auto-disconnection. trackable must be inherited when objects shall automatically invalidate slots referring to them on destruction. A slot built from a member function of a trackable derived type installs a callback that is invoked when the trackable object is destroyed or overwritten.

add\_destroy\_notify\_callback() and remove\_destroy\_notify\_callback() can be used to manually install and remove callbacks when notification of the object dying is needed.

notify\_callbacks() invokes and removes all previously installed callbacks and can therefore be used to disconnect from all signals.

Note that there is no virtual destructor. Don't use **trackable\*** as pointer type for managing your data or the destructors of your derived types won't be called when deleting your objects.

**signal**

Cast  
*StripableTimeAxisView*  
to\_stripabletimeaxisview()  
*TimeAxisView*  
to\_timeaxisview()

## 152.144 ArdourUI:RouteUI

*C<sup>#</sup>*: RouteUI  
is-a: *ArdourUI:Selectable*

Base class for objects with auto-disconnection. trackable must be inherited when objects shall automatically invalidate slots referring to them on destruction. A slot built from a member function of a trackable derived type installs a callback that is invoked when the trackable object is destroyed or overwritten.

`add_destroy_notify_callback()` and `remove_destroy_notify_callback()` can be used to manually install and remove callbacks when notification of the object dying is needed.

`notify_callbacks()` invokes and removes all previously installed callbacks and can therefore be used to disconnect from all signals.

Note that there is no virtual destructor. Don't use `trackable*` as pointer type for managing your data or the destructors of your derived types won't be called when deleting your objects.

### signal

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.145 ArdourUI:Selectable

*C<sup>#</sup>*: Selectable

Base class for objects with auto-disconnection. `trackable` must be inherited when objects shall automatically invalidate slots referring to them on destruction. A slot built from a member function of a `trackable` derived type installs a callback that is invoked when the `trackable` object is destroyed or overwritten.

`add_destroy_notify_callback()` and `remove_destroy_notify_callback()` can be used to manually install and remove callbacks when notification of the object dying is needed.

`notify_callbacks()` invokes and removes all previously installed callbacks and can therefore be used to disconnect from all signals.

Note that there is no virtual destructor. Don't use `trackable*` as pointer type for managing your data or the destructors of your derived types won't be called when deleting your objects.

### signal

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.146 ArdourUI:Selection

*C<sup>#</sup>*: Selection

The Selection class holds lists of selected items (tracks, regions, etc. etc.).

Methods

void

`clear()`

Clear everything from the Selection

void

`clear_all()`

bool

`empty(bool)`

check if all selections are empty

**internal\_selection** also check object internals (e.g midi notes, automation points), when false only check objects.

Returns true if nothing is selected.

Data Members

*ArdourUI:MarkerSelection*

markers

*ArdourUI:RegionSelection*

regions

*ArdourUI:TimeSelection*

time

*ArdourUI:TrackSelection*

tracks

## 152.147 ArdourUI:SelectionList

*C<sub>7</sub>:* std::list<Selectable\* >

Constructor

ArdourUI.SelectionList()

Methods

*Selectable*

back()

bool

empty()

*Selectable*

front()

*LuaIter*

iter()

void

push\_back(*Selectable*)

void

reverse()

unsigned long

size()

*LuaTable*

table()

void

unique()

## 152.148 ArdourUI:StripableTimeAxisView

*C<sub>7</sub>*: StripableTimeAxisView

is-a: *ArdourUI:TimeAxisView*

Abstract base class for time-axis views (horizontal editor ‘strips’)

This class provides the basic LHS controls and display methods. This should be extended to create functional time-axis based views.

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.149 ArdourUI:TimeAxisView

*C<sub>7</sub>*: TimeAxisView

is-a: *ArdourUI:AxisView*

Abstract base class for time-axis views (horizontal editor ‘strips’)

This class provides the basic LHS controls and display methods. This should be extended to create functional time-axis based views.

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.150 ArdourUI:TimeAxisViewItem

*C<sub>7</sub>*: TimeAxisViewItem

is-a: *ArdourUI:Selectable*

Base class for items that may appear upon a TimeAxisView.

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

## 152.151 ArdourUI:TimeSelection

*C<sub>7</sub>*: TimeSelection

is-a: *ARDOUR:AudioRangeList*

Methods

long

end\_frame()

long

length()

long

start()

### 152.151.1 Inherited from ARDOUR:AudioRangeList

Constructor

ARDOUR.AudioRangeList()

Methods

*AudioRange*

back()

bool

empty()

*AudioRange*

front()

*LuaIter*

iter()

void

reverse()

unsigned long

size()

*LuaTable*

table()

## 152.152 ArdourUI:TrackSelection

*C $\ddagger$* : TrackSelection

is-a: *ArdourUI:TrackViewList*

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

### 152.152.1 Inherited from ArdourUI:TrackViewList

Methods

bool

contains(*TimeAxisView*)

*RouteList*

routelist()

## 152.152.2 Inherited from ArdourUI:TrackViewStdList

Constructor

ArdourUI.TrackViewStdList()

Methods

*TimeAxisView*

back()

bool

empty()

*TimeAxisView*

front()

*LuaIter*

iter()

void

push\_back(*TimeAxisView*)

void

reverse()

unsigned long

size()

*LuaTable*

table()

void

unique()

## 152.153 ArdourUI:TrackViewList

*C<sub>7</sub>*: TrackViewList

is-a: *ArdourUI:TrackViewStdList*

Methods

bool

contains(*TimeAxisView*)

*RouteList*

routelist()

### 152.153.1 Inherited from ArdourUI:TrackViewStdList

Constructor

ArdourUI.TrackViewStdList()

Methods

*TimeAxisView*

back()

bool

empty()

*TimeAxisView*

front()

*LuaIter*

iter()

void

push\_back(*TimeAxisView*)

void

reverse()

unsigned long

size()

*LuaTable*

table()

void

unique()

### 152.154 ArdourUI:TrackViewStdList

*C<sub>7</sub>*: std::list<TimeAxisView\*>

Constructor

ArdourUI.TrackViewStdList()

Methods

*TimeAxisView*

back()

bool

empty()

*TimeAxisView*

front()  
*LuaIter*  
iter()  
void  
push\_back(*TimeAxisView*)  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()  
void  
unique()

## 152.155 C:ByteArray

*C‡:* unsigned char\*

Methods

*LuaMetaTable*

array()  
*LuaTable*  
get\_table()  
unsigned char\*  
offset(unsigned int)  
*void*  
set\_table(LuaTable {unsigned char})

## 152.156 C:DoubleVector

*C‡:* std::vector<double >

Constructor

C.DoubleVector()

C.DoubleVector()  
Methods

*LuaTable*  
add(LuaTable {double})  
*double*  
at(unsigned long)  
*bool*  
empty()  
*LuaIter*  
iter()  
*void*  
push\_back(double)  
unsigned long  
size()  
*LuaTable*  
table()

## 152.157 C:FloatArray

*C‡: float\**  
Methods  
*LuaMetaTable*  
array()  
*LuaTable*  
get\_table()  
*FloatArray*  
offset(unsigned int)  
*void*  
set\_table(LuaTable {float})

## 152.158 C:FloatArrayVector

*C‡: std::vector<float\* >*  
Constructor  
  
C.FloatArrayVector()  
  
C.FloatArrayVector()

Methods

*LuaTable*

add(LuaTable {*FloatArray*})

*FloatArray*

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(*FloatArray*)

unsigned long

size()

*LuaTable*

table()

## 152.159 C:FloatVector

*C*#: std::vector<float >

Constructor

C.FloatVector()

C.FloatVector()

Methods

*LuaTable*

add(LuaTable {float})

float

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(float)

unsigned long

size()  
*LuaTable*  
table()

## 152.160 C:Int64List

*C<sub>r</sub>*: std::list<long >  
Constructor

C.Int64List()

Methods

*LuaTable*  
add(LuaTable {long})  
long  
back()  
bool  
empty()  
long  
front()  
*LuaIter*  
iter()  
void  
push\_back(long)  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()  
void  
unique()

## 152.161 C:IntArray

*C<sub>r</sub>*: int\*  
Methods

*LuaMetaTable*  
array()  
*LuaTable*  
get\_table()  
*IntArray*  
offset(unsigned int)  
*void*  
set\_table(LuaTable {int})

## 152.162 C:StringList

*C* #: std::list<std::string >  
Constructor

C.StringList()  
Methods  
*LuaTable*  
add(LuaTable {std::string})  
std::string  
back()  
bool  
empty()  
std::string  
front()  
*LuaIter*  
iter()  
void  
push\_back(std::string)  
void  
reverse()  
unsigned long  
size()  
*LuaTable*  
table()  
void  
unique()

## 152.163 C:StringVector

*C<sub>7</sub>*: std::vector<std::string>

Constructor

C.StringVector()

C.StringVector()

Methods

*LuaTable*

add(LuaTable {std::string})

std::string

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(std::string)

unsigned long

size()

*LuaTable*

table()

## 152.164 Cairo:Context

*C<sub>7</sub>*: Cairo::Context

Context is the main class used to draw in cairomm. It contains the current state of the rendering device, including coordinates of yet to be drawn shapes.

In the simplest case, create a Context with its target Surface, set its drawing options (line width, color, etc), create shapes with methods like move\_to() and line\_to(), and then draw the shapes to the Surface using methods such as stroke() or fill().

Context is a reference-counted object that should be used via Cairo::RefPtr.

Methods

void

arc(double, double, double, double, double)

Adds a circular arc of the given radius to the current path. The arc is centered at  $(xc, yc)$ , begins at  $angle1$  and proceeds in the direction of increasing angles to end at  $angle2$ . If  $angle2$  is less than  $angle1$  it will be progressively increased by  $2*M\_PI$  until it is greater than  $angle1$ .

If there is a current point, an initial line segment will be added to the path to connect the current point to the beginning of the arc. If this initial line is undesired, it can be avoided by calling `begin_new_sub_path()` before calling `arc()`.

Angles are measured in radians. An angle of 0 is in the direction of the positive X axis (in user-space). An angle of  $M\_PI/2.0$  radians (90 degrees) is in the direction of the positive Y axis (in user-space). Angles increase in the direction from the positive X axis toward the positive Y axis. So with the default transformation matrix, angles increase in a clockwise direction.

( To convert from degrees to radians, use  $\text{degrees} * (M\_PI / 180.0)$ . )

This function gives the arc in the direction of increasing angles; see `arc_negative()` to get the arc in the direction of decreasing angles.

The arc is circular in user-space. To achieve an elliptical arc, you can scale the current transformation matrix by different amounts in the X and Y directions. For example, to draw an ellipse in the box given by `x, y, width, height`:

```
context->save();
context->translate(x, y);
context->scale(width / 2.0, height / 2.0);
context->arc(0.0, 0.0, 1.0, 0.0, 2 * M_PI);
context->restore();
```

**xc** X position of the center of the arc

**yc** Y position of the center of the arc

**radius** the radius of the arc

**angle1** the start angle, in radians

**angle2** the end angle, in radians

**void**

`arc_negative(double, double, double, double)`

Adds a circular arc of the given `radius` to the current path. The arc is centered at  $(xc, yc)$ , begins at  $angle1$  and proceeds in the direction of decreasing angles to end at  $angle2$ . If  $angle2$  is greater than  $angle1$  it will be progressively decreased by  $2*M\_PI$  until it is greater than  $angle1$ .

See `arc()` for more details. This function differs only in the direction of the arc between the two angles.

**xc** X position of the center of the arc

**yc** Y position of the center of the arc

**radius** the radius of the arc

**angle1** the start angle, in radians

**angle2** the end angle, in radians

**void**

`begin_new_path()`

Clears the current path. After this call there will be no current point.

**void**

`begin_new_sub_path()`

Begin a new subpath. Note that the existing path is not affected. After this call there will be no current point.

In many cases, this call is not needed since new subpaths are frequently started with `move_to()`.

A call to `begin_new_sub_path()` is particularly useful when beginning a new subpath with one of the `arc()` calls. This makes things easier as it is no longer necessary to manually compute the arc's initial coordinates for a call to `move_to()`.

1.2

`void`

`clip()`

Establishes a new clip region by intersecting the current clip region with the current Path as it would be filled by `fill()` and according to the current fill rule.

After `clip()`, the current path will be cleared from the cairo Context.

The current clip region affects all drawing operations by effectively masking out any changes to the surface that are outside the current clip region.

Calling `clip()` can only make the clip region smaller, never larger. But the current clip is part of the graphics state, so a temporary restriction of the clip region can be achieved by calling `clip()` within a `save()/restore()` pair. The only other means of increasing the size of the clip region is `reset_clip()`.

`set_fill_rule()`

`void`

`clip_preserve()`

Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by `fill()` and according to the current fill rule.

Unlike `clip()`, `clip_preserve` preserves the path within the cairo Context.

`clip()`

`set_fill_rule()`

`void`

`close_path()`

Adds a line segment to the path from the current point to the beginning of the current subpath, (the most recent point passed to `move_to()`), and closes this subpath. After this call the current point will be at the joined endpoint of the sub-path.

The behavior of `close_path()` is distinct from simply calling `line_to()` with the equivalent coordinate in the case of stroking. When a closed subpath is stroked, there are no caps on the ends of the subpath. Instead, there is a line join connecting the final and initial segments of the subpath.

If there is no current point before the call to `close_path()`, this function will have no effect.

`void`

`curve_to(double, double, double, double, double)`

Adds a cubic Bezier spline to the path from the current point to position ( $x_3, y_3$ ) in user-space coordinates, using ( $x_1, y_1$ ) and ( $x_2, y_2$ ) as the control points. After this call the current point will be ( $x_3, y_3$ ).

If there is no current point before the call to `curve_to()` this function will behave as if preceded by a call to `move_to( $x_1, y_1$ )`.

**x1** the X coordinate of the first control point  
**y1** the Y coordinate of the first control point  
**x2** the X coordinate of the second control point  
**y2** the Y coordinate of the second control point  
**x3** the X coordinate of the end of the curve  
**y3** the Y coordinate of the end of the curve  
void  
fill()

A drawing operator that fills the current path according to the current fill rule, (each sub-path is implicitly closed before being filled). After fill(), the current path will be cleared from the cairo context.

set\_fill\_rule()  
fill\_preserve()  
void  
fill\_preserve()

A drawing operator that fills the current path according to the current fill rule, (each sub-path is implicitly closed before being filled). Unlike fill(), fill\_preserve() preserves the path within the cairo Context.

set\_fill\_rule()  
fill().  
void  
line\_to(double, double)

Adds a line to the path from the current point to position (x, y) in user-space coordinates. After this call the current point will be (x, y).

If there is no current point before the call to line\_to() this function will behave as move\_to(x, y).

**x** the X coordinate of the end of the new line  
**y** the Y coordinate of the end of the new line  
void  
move\_to(double, double)

If the current subpath is not empty, begin a new subpath. After this call the current point will be (x, y).

**x** the X coordinate of the new position  
**y** the Y coordinate of the new position  
void

paint()

A drawing operator that paints the current source everywhere within the current clip region.

void

paint\_with\_alpha(double)

A drawing operator that paints the current source everywhere within the current clip region using a mask of constant alpha value alpha. The effect is similar to paint(), but the drawing is faded out using the alpha value.

**alpha** an alpha value, between 0 (transparent) and 1 (opaque)

void

rectangle(double, double, double, double)

Adds a closed-subpath rectangle of the given size to the current path at position (x, y) in user-space coordinates.

This function is logically equivalent to:

```
context->move_to(x, y);
context->rel_line_to(width, 0);
context->rel_line_to(0, height);
context->rel_line_to(-width, 0);
context->close_path();
```

**x** the X coordinate of the top left corner of the rectangle

**y** the Y coordinate to the top left corner of the rectangle

**width** the width of the rectangle

**height** the height of the rectangle

void

rel\_curve\_to(double, double, double, double, double)

Relative-coordinate version of curve\_to(). All offsets are relative to the current point. Adds a cubic Bezier spline to the path from the current point to a point offset from the current point by (dx3, dy3), using points offset by (dx1, dy1) and (dx2, dy2) as the control points. After this call the current point will be offset by (dx3, dy3).

Given a current point of (x, y),

```
rel_curve_to(dx1, dy1, dx2, dy2, dx3, dy3)
```

is logically equivalent to

```
curve_to(x + dx1, y + dy1, x + dx2, y + dy2, x + dx3, y + dy3).
```

It is an error to call this function with no current point. Doing so will cause this to shutdown with a status of CAIRO\_STATUS\_NO\_CURRENT\_POINT. Cairomm will then throw an exception.

**dx1** the X offset to the first control point

**dy1** the Y offset to the first control point

**dx2** the X offset to the second control point

**dy2** the Y offset to the second control point

**dx3** the X offset to the end of the curve

**dy3** the Y offset to the end of the curve

void

rel\_line\_to(double, double)

Relative-coordinate version of line\_to(). Adds a line to the path from the current point to a point that is offset from the current point by (dx, dy) in user space. After this call the current point will be offset by (dx, dy).

Given a current point of (x, y),

```
rel_line_to(dx, dy)
```

is logically equivalent to

```
line_to(x + dx, y + dy).
```

It is an error to call this function with no current point. Doing so will cause this to shutdown with a status of CAIRO\_STATUS\_NO\_CURRENT\_POINT. Cairomm will then throw an exception.

**dx** the X offset to the end of the new line

**dy** the Y offset to the end of the new line

void

**rel\_move\_to(double, double)**

If the current subpath is not empty, begin a new subpath. After this call the current point will offset by (x, y).

Given a current point of (x, y),

```
rel_move_to(dx, dy)
```

is logically equivalent to

```
move_to(x + dx, y + dy)
```

It is an error to call this function with no current point. Doing so will cause this to shutdown with a status of CAIRO\_STATUS\_NO\_CURRENT\_POINT. Cairomm will then throw an exception.

**dx** the X offset

**dy** the Y offset

void

**reset\_clip()**

Reset the current clip region to its original, unrestricted state. That is, set the clip region to an infinitely large shape containing the target surface. Equivalently, if infinity is too hard to grasp, one can imagine the clip region being reset to the exact bounds of the target surface.

Note that code meant to be reusable should not call `reset_clip()` as it will cause results unexpected by higher-level code which calls `clip()`. Consider using `save()` and `restore()` around `clip()` as a more robust means of temporarily restricting the clip region.

void

**restore()**

Restores cr to the state saved by a preceding call to `save()` and removes that state from the stack of saved states.

**save()**

void

**rotate(double)**

Modifies the current transformation matrix (CTM) by rotating the user-space axes by angle radians. The rotation of the axes takes places after any existing transformation of user space. The rotation direction for positive angles is from the positive X axis toward the positive Y axis.

**angle** angle (in radians) by which the user-space axes will be rotated

void

save()

Makes a copy of the current state of the Context and saves it on an internal stack of saved states. When restore() is called, it will be restored to the saved state. Multiple calls to save() and restore() can be nested; each call to restore() restores the state from the matching paired save().

It isn't necessary to clear all saved states before a cairo\_t is freed. Any saved states will be freed when the Context is destroyed.

restore()

void

scale(double, double)

Modifies the current transformation matrix (CTM) by scaling the X and Y user-space axes by sx and sy respectively. The scaling of the axes takes place after any existing transformation of user space.

**sx** scale factor for the X dimension

**sy** scale factor for the Y dimension

void

set\_dash(*DoubleVector*, double)

Sets the dash pattern to be used by stroke(). A dash pattern is specified by dashes, an array of positive values. Each value provides the user-space length of alternate “on” and “off” portions of the stroke. The offset specifies an offset into the pattern at which the stroke begins.

Each “on” segment will have caps applied as if the segment were a separate sub-path. In particular, it is valid to use an “on” length of 0.0 with Cairo::LINE\_CAP\_ROUND or Cairo::LINE\_CAP\_SQUARE in order to distributed dots or squares along a path.

Note: The length values are in user-space units as evaluated at the time of stroking. This is not necessarily the same as the user space at the time of set\_dash().

If dashes is empty dashing is disabled. If the size of dashes is 1, a symmetric pattern is assumed with alternating on and off portions of the size specified by the single value in dashes.

It is invalid for any value in dashes to be negative, or for all values to be 0. If this is the case, an exception will be thrown

**dashes** an array specifying alternate lengths of on and off portions

**offset** an offset into the dash pattern at which the stroke should start

void

set\_font\_size(double)

Sets the current font matrix to a scale by a factor of *size*, replacing any font matrix previously set with set\_font\_size() or set\_font\_matrix(). This results in a font size of *size* user space units. (More precisely, this matrix will result in the font's em-square being a by *size* square in user space.)

If text is drawn without a call to set\_font\_size(), (nor set\_font\_matrix() nor set\_scaled\_font()), the default font size is 10.0.

**size** the new font size, in user space units)

void

`set_line_cap(LineCap)`

Sets the current line cap style within the cairo Context. See LineCap for details about how the available line cap styles are drawn.

As with the other stroke parameters, the current line cap style is examined by `stroke()`, `stroke_extents()`, and `stroke_to_path()`, but does not have any effect during path construction.

The default line cap style is Cairo::LINE\_CAP\_BUTT.

**line\_cap** a line cap style, as a LineCap

void

`set_line_join(LineJoin)`

Sets the current line join style within the cairo Context. See LineJoin for details about how the available line join styles are drawn.

As with the other stroke parameters, the current line join style is examined by `stroke()`, `stroke_extents()`, and `stroke_to_path()`, but does not have any effect during path construction.

The default line join style is Cairo::LINE\_JOIN\_MITER.

**line\_join** a line joint style, as a LineJoin

void

`set_line_width(double)`

Sets the current line width within the cairo Context. The line width specifies the diameter of a pen that is circular in user-space, (though device-space pen may be an ellipse in general due to scaling/shear/rotation of the CTM).

Note: When the description above refers to user space and CTM it refers to the user space and CTM in effect at the time of the stroking operation, not the user space and CTM in effect at the time of the call to `set_line_width()`. The simplest usage makes both of these spaces identical. That is, if there is no change to the CTM between a call to `set_line_width()` and the stroking operation, then one can just pass user-space values to `set_line_width()` and ignore this note.

As with the other stroke parameters, the current line cap style is examined by `stroke()`, `stroke_extents()`, and `stroke_to_path()`, but does not have any effect during path construction.

The default line width value is 2.0.

**width** a line width, as a user-space value

void

`set_operator(Operator)`

Sets the compositing operator to be used for all drawing operations. See Operator for details on the semantics of each available compositing operator.

**op** a compositing operator, specified as a Operator

void

`set_source_rgb(double, double, double)`

Sets the source pattern within the Context to an opaque color. This opaque color will then be used for any subsequent drawing operation until a new source pattern is set.

The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

```
set_source_rgba()  
set_source()  
red red component of color  
green green component of color  
blue blue component of color  
void  
set_source_rgba(double, double, double, double)
```

Sets the source pattern within the Context to a translucent color. This color will then be used for any subsequent drawing operation until a new source pattern is set.

The color and alpha components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

```
set_source_rgb()  
set_source()  
red red component of color  
green green component of color  
blue blue component of color  
alpha alpha component of color  
void  
show_text(std::string)
```

A drawing operator that generates the shape from a string of UTF-8 characters, rendered according to the current font\_face, font\_size (font\_matrix), and font\_options.

This function first computes a set of glyphs for the string of text. The first glyph is placed so that its origin is at the current point. The origin of each subsequent glyph is offset from that of the previous glyph by the advance values of the previous glyph.

After this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for easy display of a single logical string with multiple calls to show\_text().

Note: The show\_text() function call is part of what the cairo designers call the “toy” text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See show\_glyphs() for the “real” text display API in cairo.

**utf8** a string containing text encoded in UTF-8

void

stroke()

A drawing operator that strokes the current Path according to the current line width, line join, line cap, and dash settings. After stroke(), the current Path will be cleared from the cairo Context.

```
set_line_width()  
set_line_join()  
set_line_cap()
```

```
set_dash()
stroke_preserve().
```

Note: Degenerate segments and sub-paths are treated specially and provide a useful result. These can result in two different situations:

1. Zero-length “on” segments set in set\_dash(). If the cap style is Cairo::LINE\_CAP\_ROUND or Cairo::LINE\_CAP\_SQUARE then these segments will be drawn as circular dots or squares respectively. In the case of Cairo::LINE\_CAP\_SQUARE, the orientation of the squares is determined by the direction of the underlying path.
2. A sub-path created by move\_to() followed by either a close\_path() or one or more calls to line\_to() to the same coordinate as the move\_to(). If the cap style is Cairo::LINE\_CAP\_ROUND then these sub-paths will be drawn as circular dots. Note that in the case of Cairo::LINE\_CAP\_SQUARE a degenerate sub-path will not be drawn at all, (since the correct orientation is indeterminate).

In no case will a cap style of Cairo::LINE\_CAP\_BUTT cause anything to be drawn in the case of either degenerate segments or sub-paths.

```
void
stroke_preserve()
```

A drawing operator that strokes the current Path according to the current line width, line join, line cap, and dash settings. Unlike stroke(), stroke\_preserve() preserves the Path within the cairo Context.

```
set_line_width()
set_line_join()
set_line_cap()
set_dash()
stroke_preserve().
```

```
void
translate(double, double)
```

Modifies the current transformation matrix (CTM) by translating the user-space origin by (tx, ty). This offset is interpreted as a user-space coordinate according to the CTM in place before the new call to translate. In other words, the translation of the user-space origin takes place after any existing transformation.

**tx** amount to translate in the X direction

**ty** amount to translate in the Y direction

void

unset\_dash()

This function disables a dash pattern that was set with set\_dash()

## 152.165 Cairo:ImageSurface

*C*#: LuaCairo::ImageSurface

wrap RefPtr< Cairo::ImageSurface >

Image surfaces provide the ability to render to memory buffers either allocated by cairo or by the calling code. The supported image formats are those defined in Cairo::Format.

Constructor

Cairo.ImageSurface(*Format*, int, int)

Methods

*Context*

context()

Returns a context object to perform operations on the surface

int

get\_height()

Gets the height of the ImageSurface in pixels

int

get\_stride()

Returns the stride of the image surface in bytes (or 0 if surface is not an image surface). The stride is the distance in bytes from the beginning of one row of the image data to the beginning of the next row.

int

get\_width()

Gets the width of the ImageSurface in pixels

void

set\_as\_source(*Context*, int, int)

## 152.166 Cairo:PangoLayout

*C*#: LuuCairo::PangoLayout

Constructor

Cairo.PangoLayout(*Context*, std::string)

Methods

*EllipsizeMode*

get\_ellipsize()

Gets the type of ellipsization being performed for *layout*. See set\_ellipsize()

Use is\_ellipsized() to query whether any paragraphs were actually ellipsized.

Returns The current ellipsization mode for *layout*.

...

get\_pixel\_size(-lua-)

Determines the logical width and height of a Pango::Layout in device units.

std::string

get\_text()

Gets the text in the layout. The returned text should not be freed or modified.

Returns The text in the *layout*.

*WrapMode*

`get_wrap()`

Gets the wrap mode for the layout.

Use `is_wrapped()` to query whether any paragraphs were actually wrapped.

Returns Active wrap mode.

`bool`

`is_ellipsized()`

Queries whether the layout had to ellipsize any paragraphs.

This returns `true` if the ellipsization mode for *layout* is not Pango::ELLIPSIZE\_NONE, a positive width is set on *layout*, and there are paragraphs exceeding that width that have to be ellipsized.

Returns `true` if any paragraphs had to be ellipsized, `false` otherwise.

`bool`

`is_wrapped()`

Queries whether the layout had to wrap any paragraphs.

This returns `true` if a positive width is set on *layout*, ellipsization mode of *layout* is set to Pango::ELLIPSIZE\_NONE, and there are paragraphs exceeding the layout width that have to be wrapped.

Returns `true` if any paragraphs had to be wrapped, `false` otherwise.

`void`

`layout_cairo_path(Context)`

`void`

`set_ellipsize(EllipsizeMode)`

Sets the type of ellipsization being performed for *layout*. Depending on the ellipsization mode *ellipsize* text is removed from the start, middle, or end of text so they fit within the width and height of layout set with `set_width()` and `set_height()`.

If the layout contains characters such as newlines that force it to be layed out in multiple paragraphs, then whether each paragraph is ellipsized separately or the entire layout is ellipsized as a whole depends on the set height of the layout. See `set_height()` for details.

**ellipsize** The new ellipsization mode for *layout*.

`void`

`set_markup(std::string)`

Sets the layout text and attribute list from marked-up text (see markup format). Replaces the current text and attribute list.

**markup** Some marked-up text.

`void`

`set_text(std::string)`

Set the text of the layout.

**text** The text for the layout.

void

set\_width(int)

Sets the width to which the lines of the Pango::Layout should wrap or ellipsized. The default value is -1: no width set.

**width** The desired width in Pango units, or -1 to indicate that no wrapping or ellipsization should be performed.

void

set\_wrap(*WrapMode*)

Sets the wrap mode; the wrap mode only has effect if a width is set on the layout with set\_width(). To turn off wrapping, set the width to -1.

**wrap** The wrap mode.

void

show\_in\_cairo\_context(*Context*)

## 152.167 Evoral::Beats

*C<sub>++</sub>*: Evoral::Beats

Musical time in beats.

Constructor

Evoral.Beats(double)

Create from a real number of beats.

Methods

double

to\_double()

## 152.168 Evoral::Control

*C<sub>++</sub>*: boost::shared\_ptr< Evoral::Control >, boost::weak\_ptr< Evoral::Control >

Base class representing some kind of (automatable) control; a fader's gain, for example, or a compressor plugin's threshold.

The class knows the Evoral::Parameter that it is controlling, and has a list of values for automation.

Methods

bool

isnil()

*ControlList*

list()

## 152.169 Evoral::ControlEvent

*C<sub>7</sub>*: Evoral::ControlEvent

A single event (time-stamped value) for a control

Data Members

double

value

double

when

## 152.170 Evoral::ControlList

*C<sub>7</sub>*: boost::shared\_ptr< Evoral::ControlList >, boost::weak\_ptr< Evoral::ControlList >

A list (sequence) of time-stamped values for a control

Methods

void

add(double, double, bool, bool)

add automation events

**when** absolute time in samples

**value** parameter value

**with\_guards** if true, add guard-points

**with\_initial** if true, add an initial point if the list is empty

void

clear(double, double)

remove all automation events between the given time range

**start** start of range (inclusive) in audio samples

**end** end of range (inclusive) in audio samples

void

clear\_list()

double

eval(double)

query value at given time (takes a read-lock, not safe while writing automation)

**where** absolute time in samples

Returns parameter value

*EventList*

events()

bool

in\_write\_pass()

*InterpolationStyle*

interpolation()

query interpolation style of the automation data

Returns Interpolation Style

bool

isnil()

*LuaTable(double, ...)*

rt\_safe\_eval(double, bool&)

realtime safe version of eval, may fail if read-lock cannot be taken

**where** absolute time in samples

**ok** boolean reference if returned value is valid

Returns parameter value

bool

set\_interpolation(*InterpolationStyle*)

set the interpolation style of the automation data.

This will fail when asking for Logarithmic scale and min,max crosses 0 or Exponential scale with min != 0.

**is** interpolation style

Returns true if style change was successful

void

thin(double)

Thin the number of events in this list.

The thinning factor corresponds to the area of a triangle computed between three points in the list (time-difference \* value-difference). If the area is large, it indicates significant non-linearity between the points.

Time is measured in samples, value is usually normalized to 0..1.

During automation recording we thin the recorded points using this value. If a point is sufficiently co-linear with its neighbours (as defined by the area of the triangle formed by three of them), we will not include it in the list. The larger the value, the more points are excluded, so this effectively measures the amount of thinning to be done.

**thinning\_factor** area-size (default: 20)

void

truncate\_end(double)

truncate the event list after the given time

**last\_coordinate** last event to include

void

truncate\_start(double)

truncate the event list to the given time

**overall\_length** overall length

## 152.171 Evoral:ControlSet

*C<sub>r</sub>*: boost::shared\_ptr< Evoral::ControlSet >, boost::weak\_ptr< Evoral::ControlSet >

Methods

bool

isnil()

## 152.172 Evoral:Event

*C<sub>r</sub>*: Evoral::Event<long>

Methods

unsigned char\*

buffer()

unsigned char

channel()

void

clear()

void

set\_buffer(unsigned int, unsigned char\*, bool)

void

set\_channel(unsigned char)

void

set\_type(unsigned char)

unsigned int

size()

long

time()

unsigned char

type()

## 152.173 Evoral:NotePtr

*C<sub>r</sub>*: boost::shared\_ptr< Evoral::Note<Evoral::Beats> >, boost::weak\_ptr< Evoral::Note<Evoral::Beats> >

Methods

unsigned char

channel()

bool

isnil()

*Beats*

length()

unsigned char

note()

unsigned char

off\_velocity()

*Beats*

time()

unsigned char

velocity()

## 152.174 Evoral:Parameter

*C<sup>#</sup>:* Evoral::Parameter

ID of a [play|record|automate]able parameter.

A parameter is defined by (type, id, channel). Type is an integer which can be used in any way by the application (e.g. cast to a custom enum, map to/from a URI, etc). ID is type specific (e.g. MIDI controller #).

This class defines a < operator which is a strict weak ordering, so Parameter may be stored in a std::set, used as a std::map key, etc.

Constructor

Evoral.Parameter(unsigned int, unsigned char, unsigned int)

Methods

unsigned char

channel()

unsigned int

id()

unsigned int

type()

## 152.175 Evoral:ParameterDescriptor

*C<sub>7</sub>*: Evoral::ParameterDescriptor

Description of the value range of a parameter or control.

Constructor

Evoral.ParameterDescriptor()

Data Members

bool

logarithmic

True for log-scale parameters

float

lower

Minimum value (in Hz, for frequencies)

float

normal

Default value

bool

toggled

True iff parameter is boolean

float

upper

Maximum value (in Hz, for frequencies)

## 152.176 Evoral:Range

*C<sub>7</sub>*: Evoral::Range<long>

Constructor

Evoral.Range(long, long)

Data Members

long

from

long

to

## 152.177 Evoral:Sequence

*C<sub>7</sub>*: boost::shared\_ptr< Evoral::Sequence<Evoral::Beats> >, boost::weak\_ptr< Evoral::Sequence<Evoral::Beats> >

is-a: *Evoral:ControlSet*

Methods

bool

isnil()

## 152.178 LuaDialog:Dialog

*C<sub>7</sub>*: LuaDialog::Dialog

Constructor

LuaDialog.Dialog(std::string, Lua-Function)

Methods

...

run(-lua-)

## 152.179 LuaDialog:Message

*C<sub>7</sub>*: LuaDialog::Message

Constructor

LuaDialog.Message(std::string, std::string, *MessageType*, *ButtonType*)

Methods

int

run()

## 152.180 LuaSignal:Set

*C<sub>7</sub>*: std::bitset<48ul>

Constructor

LuaSignal.Set()

Methods

*LuaTable*

```
add(48ul)
bool
any()
unsigned long
count()
bool
none()
Set
reset()
Set
set(unsigned long, bool)
unsigned long
size()
LuaTable
table()
bool
test(unsigned long)
```

## 152.181 PBD

Methods

```
bool
open_uri(std::string)
bool
open_uri(std::string)
```

## 152.182 PBD:Command

*C++:* Command  
is-a: *PBD:StatefulDestructible*  
Base class for Undo/Redo commands and changesets

Methods

```
std::string
name()
void
set_name(std::string)
```

### 152.182.1 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.183 PBD:Configuration

*C++:* PBD::Configuration

is-a: *PBD:Stateful*

Base class for objects with saveable and undoable state

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

### 152.183.1 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.184 PBD:Controllable

*C++:* boost::shared\_ptr< PBD::Controllable >, boost::weak\_ptr< PBD::Controllable >

is-a: *PBD:StatefulDestructiblePtr*

This is a pure virtual class to represent a scalar control.

Note that it contains no storage/state for the controllable thing that it represents. Derived classes must provide set\_value()/get\_value() methods, which will involve (somehow) an actual location to store the value.

In essence, this is an interface, not a class.

Without overriding upper() and lower(), a derived class will function as a control whose value can range between 0 and 1.0.

Methods

double

get\_value()

bool

isnil()

std::string

name()

### 152.184.1 Inherited from PBD:StatefulPtr

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.185 PBD:ID

*C<sub>7</sub>*: PBD::ID

a unique ID to identify objects numerically

Constructor

PBD.ID(std::string)

Methods

std::string

to\_s()

## 152.186 PBD:IdVector

*C<sub>7</sub>*: std::vector<PBD::ID >

Constructor

PBD.IdVector()

PBD.IdVector()

Methods

*LuaTable*

add(LuaTable {*ID*})

*ID*

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(*ID*)

unsigned long

size()

*LuaTable*

table()

## 152.187 PBD:RingBuffer8

*C#*: PBD::RingBufferNPT<unsigned char>

Constructor

PBD.RingBuffer8(unsigned long)

Methods

void

increment\_read\_ptr(unsigned long)

void

increment\_write\_ptr(unsigned long)

unsigned long

read(unsigned char\*, unsigned long)

unsigned long

read\_space()

void

reset()

unsigned long

```
write(unsigned char*, unsigned long)
unsigned long
write_one(unsigned char)
unsigned long
write_space()
```

## 152.188 PBD:RingBufferF

*C<sub>7</sub>*: PBD::RingBufferNPT<float>

Constructor

PBD.RingBufferF(unsigned long)

Methods

```
void
increment_read_ptr(unsigned long)
void
increment_write_ptr(unsigned long)
unsigned long
read(FloatArray, unsigned long)
unsigned long
read_space()
void
reset()
unsigned long
write(FloatArray, unsigned long)
unsigned long
write_one(float)
unsigned long
write_space()
```

## 152.189 PBD:RingBufferI

*C<sub>7</sub>*: PBD::RingBufferNPT<int>

Constructor

PBD.RingBufferI(unsigned long)

Methods

void

increment\_read\_ptr(unsigned long)

void

increment\_write\_ptr(unsigned long)

unsigned long

read(*IntArray*, unsigned long)

unsigned long

read\_space()

void

reset()

unsigned long

write(*IntArray*, unsigned long)

unsigned long

write\_one(int)

unsigned long

write\_space()

## 152.190 PBD:Stateful

*C*#: PBD::Stateful

Base class for objects with saveable and undoable state

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.191 PBD:StatefulDestructible

*C*#: PBD::StatefulDestructible

is-a: *PBD:Stateful*

Base class for objects with saveable and undoable state with destruction notification

This class object is only used indirectly as return-value and function-parameter. It provides no methods by itself.

### 152.191.1 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.192 PBD:StatefulDestructiblePtr

*C<sub>7</sub>*: boost::shared\_ptr< PBD::StatefulDestructible >, boost::weak\_ptr< PBD::StatefulDestructible >

is-a: *PBD:StatefulPtr*

Base class for objects with saveable and undoable state with destruction notification

Methods

bool

isnil()

### 152.192.1 Inherited from PBD:StatefulPtr

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.193 PBD:StatefulDiffCommand

*C<sub>7</sub>*: PBD::StatefulDiffCommand

is-a: *PBD:Command*

A Command which stores its action as the differences between the before and after state of a Stateful object.

Methods

bool

empty()

void

undo()

### 152.193.1 Inherited from PBD:Command

Methods

std::string

name()

void

set\_name(std::string)

### 152.193.2 Inherited from PBD:Stateful

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

*OwnedPropertyList*

properties()

## 152.194 PBD:StatefulPtr

*C++:* boost::shared\_ptr< PBD::Stateful >, boost::weak\_ptr< PBD::Stateful >

Base class for objects with saveable and undoable state

Methods

void

clear\_changes()

Forget about any changes to this object's properties

*ID*

id()

bool

isnil()

*OwnedPropertyList*

properties()

## 152.195 PBD:XMLNode

*C<sub>7</sub>*: XMLNode

Methods

std::string

name()

## 152.196 Timecode:BBT\_TIME

*C<sub>7</sub>*: Timecode::BBT\_Time

Bar, Beat, Tick Time (i.e. Tempo-Based Time)

Constructor

Timecode.BBT\_TIME(unsigned int, unsigned int, unsigned int)

Data Members

unsigned int

bars

unsigned int

beats

unsigned int

ticks

## 152.197 Timecode:Time

*C<sub>7</sub>*: Timecode::Time

Constructor

Timecode.Time(double)

Data Members

bool

drop

Whether this Time uses dropframe Timecode

unsigned int

frames

Timecode frames (not audio samples)

unsigned int

hours

unsigned int

minutes

bool

negative

double

rate

Frame rate of this Time

unsigned int

seconds

unsigned int

subframes

Typically unused

## 152.198 Vamp:Plugin

*C<sub>r</sub>*: Vamp::Plugin

is-a: *Vamp:PluginBase*

Vamp::Plugin is a base class for plugin instance classes that provide feature extraction from audio or related data.

In most cases, the input will be audio and the output will be a stream of derived data at a lower sampling resolution than the input.

Note that this class inherits several abstract methods from PluginBase. These must be implemented by the subclass.

### PLUGIN LIFECYCLE

Feature extraction plugins are managed differently from real-time plugins (such as VST effects). The main difference is that the parameters for a feature extraction plugin are configured before the plugin is used, and do not change during use.

1. Host constructs the plugin, passing it the input sample rate. The plugin may do basic initialisation, but should not do anything computationally expensive at this point. You must make sure your plugin is cheap to construct, otherwise you'll seriously affect the startup performance of almost all hosts. If you have serious initialisation to do, the proper place is in initialise() (step 5).

2. Host may query the plugin's available outputs.
3. Host queries programs and parameter descriptors, and may set some or all of them. Parameters that are not explicitly set should take their default values as specified in the parameter descriptor. When a program is set, the parameter values may change and the host will re-query them to check.

4. Host queries the preferred step size, block size and number of channels. These may all vary depending on the parameter values. (Note however that you cannot make the number of distinct outputs dependent on parameter values.)
5. Plugin is properly initialised with a call to initialise. This fixes the step size, block size, and number of channels, as well as all of the parameter and program settings. If the values passed in to initialise do not match the plugin's advertised preferred values from step 4, the plugin may refuse to initialise and return false (although if possible it should accept the new values). Any computationally expensive setup code should take place here.
6. Host finally checks the number of values, resolution, extents etc per output (which may vary depending on the number of channels, step size and block size as well as the parameter values).
7. Host will repeatedly call the process method to pass in blocks of input data. This method may return features extracted from that data (if the plugin is causal).
8. Host will call getRemainingFeatures exactly once, after all the input data has been processed. This may return any non-causal or leftover features.
9. At any point after initialise was called, the host may optionally call the reset method and restart processing. (This does not mean it can change the parameters, which are fixed from initialise until destruction.)

A plugin does not need to handle the case where setParameter or selectProgram is called after initialise has been called. It's the host's responsibility not to do that. Similarly, the plugin may safely assume that initialise is called no more than once.

## Methods

### *InputDomain*

`getInputDomain()`

Get the plugin's required input domain.

If this is TimeDomain, the samples provided to the process() function (below) will be in the time domain, as for a traditional audio processing plugin.

If this is FrequencyDomain, the host will carry out a windowed FFT of size equal to the negotiated block size on the data before passing the frequency bin data in to process(). The input data for the FFT will be rotated so as to place the origin in the centre of the block. The plugin does not get to choose the window type – the host will either let the user do so, or will use a Hanning window.

`unsigned long`

`getMaxChannelCount()`

Get the maximum supported number of input channels.

`unsigned long`

`getMinChannelCount()`

Get the minimum supported number of input channels.

### *OutputList*

`getOutputDescriptors()`

Get the outputs of this plugin. An output's index in this list is used as its numeric index when looking it up in the FeatureSet returned from the process() call.

`unsigned long`

`getPreferredBlockSize()`

Get the preferred block size (window size – the number of sample frames passed in each block to the process() function). This should be called before initialise().

A plugin that can handle any block size may return 0. The final block size will be set in the initialise() call.

unsigned long

getPreferredStepSize()

Get the preferred step size (window increment – the distance in sample frames between the start frames of consecutive blocks passed to the process() function) for the plugin. This should be called before initialise().

A plugin may return 0 if it has no particular interest in the step size. In this case, the host should make the step size equal to the block size if the plugin is accepting input in the time domain. If the plugin is accepting input in the frequency domain, the host may use any step size. The final step size will be set in the initialise() call.

*FeatureSet*

getRemainingFeatures()

After all blocks have been processed, calculate and return any remaining features derived from the complete input.

std::string

getType()

Used to distinguish between Vamp::Plugin and other potential sibling subclasses of PluginBase. Do not reimplement this function in your subclass.

bool

initialise(unsigned long, unsigned long, unsigned long)

Initialise a plugin to prepare it for use with the given number of input channels, step size (window increment, in sample frames) and block size (window size, in sample frames).

The input sample rate should have been already specified at construction time.

Return true for successful initialisation, false if the number of input channels, step size and/or block size cannot be supported.

void

reset()

Reset the plugin after use, to prepare it for another clean run. Not called for the first initialisation (i.e. initialise must also do a reset).

## 152.198.1 Inherited from Vamp:PluginBase

Methods

std::string

getCopyright()

Get the copyright statement or licensing summary for the plugin. This can be an informative text, without the same presentation constraints as mentioned for getMaker above.

std::string

getCurrentProgram()

Get the current program.

```
std::string  
getDescription()
```

Get a human-readable description for the plugin, typically a line of text that may optionally be displayed in addition to the plugin’s “name”. May be empty if the name has said it all already.

Example: “Detect and count zero crossing points”

```
std::string  
getIdentifier()
```

Get the computer-readable name of the plugin. This should be reasonably short and contain no whitespace or punctuation characters. It may only contain the characters [a-zA-Z0-9\_-]. This is the authoritative way for a program to identify a plugin within a given library.

This text may be visible to the user, but it should not be the main text used to identify a plugin to the user (that will be the name, below).

Example: “zero\_crossings”

```
std::string  
getMaker()
```

Get the name of the author or vendor of the plugin in human-readable form. This should be a short identifying text, as it may be used to label plugins from the same source in a menu or similar.

```
std::string  
getName()
```

Get a human-readable name or title of the plugin. This should be brief and self-contained, as it may be used to identify the plugin to the user in isolation (i.e. without also showing the plugin’s “identifier”).

Example: “Zero Crossings”

```
float  
getParameter(std::string)
```

Get the value of a named parameter. The argument is the identifier field from that parameter’s descriptor.

*ParameterList*

```
getParameterDescriptors()
```

Get the controllable parameters of this plugin.

```
int  
getPluginVersion()
```

Get the version number of the plugin.

*StringVector*

```
getPrograms()
```

Get the program settings available in this plugin. A program is a named shorthand for a set of parameter values; changing the program may cause the plugin to alter the values of its published parameters (and/or non-public internal processing parameters). The host should re-read the plugin’s parameter values after setting a new program.

The programs must have unique names.

```
void
```

selectProgram(std::string)

Select a program. (If the given program name is not one of the available programs, do nothing.)

void

setParameter(std::string, float)

Set a named parameter. The first argument is the identifier field from that parameter's descriptor.

## 152.199 Vamp:Plugin:Feature

*C#:* Vamp::Plugin::Feature

Data Members

*Vamp:RealTime*

duration

Duration of the output feature. This is mandatory if the output has VariableSampleRate or FixedSampleRate and hasDuration is true, and unused otherwise.

bool

hasDuration

True if an output feature has a specified duration. This is optional if the output has VariableSampleRate or FixedSampleRate, and unused if the output has OneSamplePerStep.

bool

hasTimestamp

True if an output feature has its own timestamp. This is mandatory if the output has VariableSampleRate, optional if the output has FixedSampleRate, and unused if the output has OneSamplePerStep.

std::string

label

Label for the sample of this feature.

*Vamp:RealTime*

timestamp

Timestamp of the output feature. This is mandatory if the output has VariableSampleRate or if the output has FixedSampleRate and hasTimestamp is true, and unused otherwise.

*C:FloatVector*

values

Results for a single sample of this feature. If the output hasFixedBinCount, there must be the same number of values as the output's binCount count.

## 152.200 Vamp:Plugin:FeatureList

*C#:* std::vector<Vamp::Plugin::Feature >

Constructor

Vamp.Plugin.FeatureList()

Vamp.Plugin.FeatureList()

Methods

*LuaTable*

add(*LuaTable* {*Feature*})

*Feature*

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(*Feature*)

unsigned long

size()

*LuaTable*

table()

## 152.201 Vamp:Plugin:FeatureSet

*C<sub>f</sub>*: std::map<int, std::vector<Vamp::Plugin::Feature>>>

Constructor

Vamp.Plugin.FeatureSet()

Methods

*LuaTable*

add(*LuaTable* {*Feature*})

...

at(-lua-)

void

clear()

unsigned long

count(int)

bool

empty()  
*LuaIter*  
iter()  
unsigned long  
size()  
*LuaTable*  
table()

## 152.202 Vamp::Plugin::OutputDescriptor

*C#:* Vamp::Plugin::OutputDescriptor

Data Members

unsigned long

binCount

The number of values per result of the output. Undefined if hasFixedBinCount is false. If this is zero, the output is point data (i.e. only the time of each output is of interest, the value list will be empty).

*C:StringVector*

binNames

The (human-readable) names of each of the bins, if appropriate. This is always optional.

std::string

description

A human-readable short text describing the output. May be empty if the name has said it all already.  
Example: “The number of zero crossing points per processing block”

bool

hasDuration

True if the returned results for this output are known to have a duration field.

bool

hasFixedBinCount

True if the output has the same number of values per sample for every output sample. Outputs for which this is false are unlikely to be very useful in a general-purpose host.

bool

hasKnownExtents

True if the results in each output bin fall within a fixed numeric range (minimum and maximum values). Undefined if binCount is zero.

std::string

identifier

The name of the output, in computer-usuable form. Should be reasonably short and without whitespace or punctuation, using the characters [a-zA-Z0-9\_-] only. Example: “zero\_crossing\_count”

bool

isQuantized

True if the output values are quantized to a particular resolution. Undefined if binCount is zero.

float

maxValue

Maximum value of the results in the output. Undefined if hasKnownExtents is false or binCount is zero.

float

minValue

Minimum value of the results in the output. Undefined if hasKnownExtents is false or binCount is zero.

float

quantizeStep

Quantization resolution of the output values (e.g. 1.0 if they are all integers). Undefined if isQuantized is false or binCount is zero.

float

sampleRate

Sample rate of the output results, as samples per second. Undefined if sampleType is OneSamplePerStep.

If sampleType is VariableSampleRate and this value is non-zero, then it may be used to calculate a resolution for the output (i.e. the “duration” of each sample, in time, will be 1/sampleRate seconds). It’s recommended to set this to zero if that behaviour is not desired.

*Vamp.Plugin.OutputDescriptor.SampleType*

sampleType

Positioning in time of the output results.

std::string

unit

The unit of the output, in human-readable form.

## 152.203 Vamp:Plugin:OutputList

*C<sub>7</sub>*: std::vector<Vamp::Plugin::OutputDescriptor >

Constructor

Vamp.Plugin.OutputList()

Vamp.Plugin.OutputList()

Methods

*LuaTable*

add(LuaTable { *OutputDescriptor* })

*OutputDescriptor*  
at(unsigned long)  
bool  
empty()  
*LuaIter*  
iter()  
void  
push\_back(*OutputDescriptor*)  
unsigned long  
size()  
*LuaTable*  
table()

## 152.204 Vamp:PluginBase

*C*#: Vamp::PluginBase

A base class for plugins with optional configurable parameters, programs, etc. The Vamp::Plugin is derived from this, and individual Vamp plugins should derive from that.

This class does not provide the necessary interfaces to instantiate or run a plugin. It only specifies an interface for retrieving those controls that the host may wish to show to the user for editing. It could meaningfully be subclassed by real-time plugins or other sorts of plugin as well as Vamp plugins.

Methods

std::string

getCopyright()

Get the copyright statement or licensing summary for the plugin. This can be an informative text, without the same presentation constraints as mentioned for getMaker above.

std::string

getCurrentProgram()

Get the current program.

std::string

getDescription()

Get a human-readable description for the plugin, typically a line of text that may optionally be displayed in addition to the plugin's "name". May be empty if the name has said it all already.

Example: "Detect and count zero crossing points"

std::string

getIdentifer()

Get the computerusable name of the plugin. This should be reasonably short and contain no whitespace or punctuation characters. It may only contain the characters [a-zA-Z0-9\_-]. This is the authoritative way for a program to identify a plugin within a given library.

This text may be visible to the user, but it should not be the main text used to identify a plugin to the user (that will be the name, below).

Example: “zero\_crossings”

std::string

getMaker()

Get the name of the author or vendor of the plugin in human-readable form. This should be a short identifying text, as it may be used to label plugins from the same source in a menu or similar.

std::string

getName()

Get a human-readable name or title of the plugin. This should be brief and self-contained, as it may be used to identify the plugin to the user in isolation (i.e. without also showing the plugin’s “identifier”).

Example: “Zero Crossings”

float

getParameter(std::string)

Get the value of a named parameter. The argument is the identifier field from that parameter’s descriptor.

*ParameterList*

getParameterDescriptors()

Get the controllable parameters of this plugin.

int

getPluginVersion()

Get the version number of the plugin.

*StringVector*

getPrograms()

Get the program settings available in this plugin. A program is a named shorthand for a set of parameter values; changing the program may cause the plugin to alter the values of its published parameters (and/or non-public internal processing parameters). The host should re-read the plugin’s parameter values after setting a new program.

The programs must have unique names.

std::string

getType()

Get the type of plugin. This is to be implemented by the immediate subclass, not by actual plugins. Do not attempt to implement this in plugin code.

void

selectProgram(std::string)

Select a program. (If the given program name is not one of the available programs, do nothing.)

void

setParameter(std::string, float)

Set a named parameter. The first argument is the identifier field from that parameter’s descriptor.

## 152.205 Vamp::PluginBase::ParameterDescriptor

*C<sub>7</sub>*: Vamp::PluginBase::ParameterDescriptor

Data Members

float

defaultValue

The default value of the parameter. The plugin should ensure that parameters have this value on initialisation (i.e. the host is not required to explicitly set parameters if it wants to use their default values).

std::string

description

A human-readable short text describing the parameter. May be empty if the name has said it all already.

std::string

identifier

The name of the parameter, in computer-readable form. Should be reasonably short, and may only contain the characters [a-zA-Z0-9\_-].

bool

isQuantized

True if the parameter values are quantized to a particular resolution.

float

maxValue

The maximum value of the parameter.

float

minValue

The minimum value of the parameter.

std::string

name

The human-readable name of the parameter.

float

quantizeStep

Quantization resolution of the parameter values (e.g. 1.0 if they are all integers). Undefined if isQuantized is false.

std::string

unit

The unit of the parameter, in human-readable form.

*C:StringVector*

valueNames

Names for the quantized values. If isQuantized is true, this may either be empty or contain one string for each of the quantize steps from minValue up to maxValue inclusive. Undefined if isQuantized is false.

If these names are provided, they should be shown to the user in preference to the values themselves. The user may never see the actual numeric values unless they are also encoded in the names.

## 152.206 Vamp:PluginBase:ParameterList

*C++:* std::vector<Vamp::PluginBase::ParameterDescriptor >

Constructor

Vamp.PluginBase.ParameterList()

Vamp.PluginBase.ParameterList()

Methods

*LuaTable*

add(*LuaTable* {*ParameterDescriptor*})

*ParameterDescriptor*

at(unsigned long)

bool

empty()

*LuaIter*

iter()

void

push\_back(*ParameterDescriptor*)

unsigned long

size()

*LuaTable*

table()

## 152.207 Vamp:RealTime

*C++:* Vamp::RealTime

RealTime represents time values to nanosecond precision with accurate arithmetic and frame-rate conversion functions.

Constructor

Vamp.RealTime(int, int)

Methods

*RealTime*

```
frame2RealTime(long, unsigned int)
int
msec()
long
realTime2Frame(RealTime, unsigned int)
std::string
toString()

Return a human-readable debug-type string to full precision (probably not a format to show to a user directly)

int
usec()

Data Members

int
nsec
int
sec
```

## 152.208 os

```
Methods

int
execute(std::string)
LuaTable
forkexec()
```

## 152.209 Enum/Constants

### 152.209.1 PBD.Controllable.GroupControlDisposition

- PBD.GroupControlDisposition.InverseGroup
- PBD.GroupControlDisposition.NoGroup
- PBD.GroupControlDisposition.UseGroup

### 152.209.2 Timecode.TimecodeFormat

- Timecode.TimecodeFormat.TC23976
- Timecode.TimecodeFormat.TC24
- Timecode.TimecodeFormat.TC24976

- Timecode.TimecodeFormat.TC25
- Timecode.TimecodeFormat.TC2997
- Timecode.TimecodeFormat.TC2997DF
- Timecode.TimecodeFormat.TC2997000
- Timecode.TimecodeFormat.TC2997000DF
- Timecode.TimecodeFormat.TC30
- Timecode.TimecodeFormat.TC5994
- Timecode.TimecodeFormat.TC60

### 152.209.3 **Evoral.ControlList.InterpolationStyle**

- Evoral.InterpolationStyle.Discrete
- Evoral.InterpolationStyle.Linear
- Evoral.InterpolationStyle.Curved

### 152.209.4 **Vamp.Plugin.InputDomain**

- Vamp.Plugin.InputDomain.TimeDomain
- Vamp.Plugin.InputDomain.FrequencyDomain

### 152.209.5 **Vamp.Plugin.OutputDescriptor.SampleType**

- Vamp.Plugin.OutputDescriptor.SampleType.OneSamplePerStep
- Vamp.Plugin.OutputDescriptor.SampleType.FixedSampleRate
- Vamp.Plugin.OutputDescriptor.SampleType.VariableSampleRate

### 152.209.6 **ARDOUR.ChanMapping**

- ARDOUR.ChanMapping.Invalid

### 152.209.7 **PBD.PropertyDescriptor<long>\***

- ARDOUR.Properties.Start
- ARDOUR.Properties.Length
- ARDOUR.Properties.Position

### 152.209.8 **ARDOUR.PresentationInfo**

- ARDOUR.PresentationInfo.max\_order

## 152.209.9 ARDOUR.PluginType

- ARDOUR.PluginType.AudioUnit
- ARDOUR.PluginType.LADSPA
- ARDOUR.PluginType.LV2
- ARDOUR.PluginType.Windows\_VST
- ARDOUR.PluginType.LXVST
- ARDOUR.PluginType.Lua

## 152.209.10 ARDOUR.PresentationInfo.Flag

- ARDOUR.PresentationInfo.Flag.AudioTrack
- ARDOUR.PresentationInfo.Flag.MidiTrack
- ARDOUR.PresentationInfo.Flag.AudioBus
- ARDOUR.PresentationInfo.Flag.MidiBus
- ARDOUR.PresentationInfo.Flag.VCA
- ARDOUR.PresentationInfo.Flag.MasterOut
- ARDOUR.PresentationInfo.Flag.MonitorOut
- ARDOUR.PresentationInfo.Flag.Auditioner
- ARDOUR.PresentationInfo.Flag.Hidden
- ARDOUR.PresentationInfo.Flag.GroupOrderSet
- ARDOUR.PresentationInfo.Flag.StatusMask

## 152.209.11 ARDOUR.AutoScale

- ARDOUR.AutoScale.Off
- ARDOUR.AutoScale.Write
- ARDOUR.AutoScale.Touch
- ARDOUR.AutoScale.Play

## 152.209.12 ARDOUR.AutomationType

- ARDOUR.AutomationType.GainAutomation
- ARDOUR.AutomationType.PluginAutomation
- ARDOUR.AutomationType.SoloAutomation
- ARDOUR.AutomationType.SoloIsolateAutomation
- ARDOUR.AutomationType.SoloSafeAutomation
- ARDOUR.AutomationType.MuteAutomation
- ARDOUR.AutomationType.RecEnableAutomation

- ARDOUR.AutomationType.RecSafeAutomation
- ARDOUR.AutomationType.TrimAutomation
- ARDOUR.AutomationType.PhaseAutomation
- ARDOUR.AutomationType.MidiCCAutomation
- ARDOUR.AutomationType.MidiPgmChangeAutomation
- ARDOUR.AutomationType.MidiPitchBenderAutomation
- ARDOUR.AutomationType.MidiChannelPressureAutomation
- ARDOUR.AutomationType.MidiNotePressureAutomation
- ARDOUR.AutomationType.MidiSystemExclusiveAutomation

### 152.209.13 ARDOUR.SrcQuality

- ARDOUR.SrcQuality.SrcBest

### 152.209.14 ARDOUR.MeterType

- ARDOUR.MeterType.MeterMaxSignal
- ARDOUR.MeterType.MeterMaxPeak
- ARDOUR.MeterType.MeterPeak
- ARDOUR.MeterType.MeterKrms
- ARDOUR.MeterType.MeterK20
- ARDOUR.MeterType.MeterK14
- ARDOUR.MeterType.MeterIEC1DIN
- ARDOUR.MeterType.MeterIEC1NOR
- ARDOUR.MeterType.MeterIEC2BBC
- ARDOUR.MeterType.MeterIEC2EBU
- ARDOUR.MeterType.MeterVU
- ARDOUR.MeterType.MeterK12
- ARDOUR.MeterType.MeterPeak0dB
- ARDOUR.MeterType.MeterMCP

### 152.209.15 ARDOUR.MeterPoint

- ARDOUR.MeterPoint.MeterInput
- ARDOUR.MeterPoint.MeterPreFader
- ARDOUR.MeterPoint.MeterPostFader
- ARDOUR.MeterPoint.MeterOutput
- ARDOUR.MeterPoint.MeterCustom

## 152.209.16 ARDOUR.Placement

- ARDOUR.Placement.PreFader
- ARDOUR.Placement.PostFader

## 152.209.17 ARDOUR.MonitorChoice

- ARDOUR.MonitorChoice.MonitorAuto
- ARDOUR.MonitorChoice.MonitorInput
- ARDOUR.MonitorChoice.MonitorDisk
- ARDOUR.MonitorChoice.MonitorCue

## 152.209.18 ARDOUR.NoteMode

- ARDOUR.NoteMode.Sustained
- ARDOUR.NoteMode.Percussive

## 152.209.19 ARDOUR.PortFlags

- ARDOUR.PortFlags.IsInput
- ARDOUR.PortFlags.IsOutput
- ARDOUR.PortFlags.IsPhysical
- ARDOUR.PortFlags.CanMonitor
- ARDOUR.PortFlags.IsTerminal

## 152.209.20 ARDOUR.MidiPortFlags

- ARDOUR.MidiPortFlags.MidiPortMusic
- ARDOUR.MidiPortFlags.MidiPortControl
- ARDOUR.MidiPortFlags.MidiPortSelection
- ARDOUR.MidiPortFlags.MidiPortVirtual

## 152.209.21 ARDOUR.PlaylistDisposition

- ARDOUR.PlaylistDisposition.CopyPlaylist
- ARDOUR.PlaylistDisposition.NewPlaylist
- ARDOUR.PlaylistDisposition.SharePlaylist

## 152.209.22 ARDOUR.MidiTrackNameSource

- ARDOUR.MidiTrackNameSource.SMFTrackNumber
- ARDOUR.MidiTrackNameSource.SMFTrackName
- ARDOUR.MidiTrackNameSource.SMFInstrumentName

## 152.209.23 ARDOUR.MidiTempoMapDisposition

- ARDOUR.MidiTempoMapDisposition.SMFTempoIgnore
- ARDOUR.MidiTempoMapDisposition.SMFTempoUse

## 152.209.24 ARDOUR.RegionPoint

- ARDOUR.RegionPoint.Start
- ARDOUR.RegionPoint.End
- ARDOUR.RegionPoint.SyncPoint

## 152.209.25 ARDOUR.PositionLockStyle

- ARDOUR.TempoSection.PositionLockStyle.AudioTime
- ARDOUR.TempoSection.PositionLockStyle.MusicTime

## 152.209.26 ARDOUR.TempoSection.Type

- ARDOUR.TempoSection.Type.Ramp
- ARDOUR.TempoSection.Type.Constant

## 152.209.27 ARDOUR.TrackMode

- ARDOUR.TrackMode.Normal
- ARDOUR.TrackMode.NonLayered
- ARDOUR.TrackMode.Destructive

## 152.209.28 ARDOUR.SampleFormat

- ARDOUR.SampleFormat.Float
- ARDOUR.SampleFormat.Int24
- ARDOUR.SampleFormat.Int16

## 152.209.29 ARDOUR.HeaderFormat

- ARDOUR.HeaderFormat.BWF
- ARDOUR.HeaderFormat.WAVE
- ARDOUR.HeaderFormat.WAVE64
- ARDOUR.HeaderFormat.CAF
- ARDOUR.HeaderFormat.AIFF
- ARDOUR.HeaderFormat.iXML
- ARDOUR.HeaderFormat.RF64
- ARDOUR.HeaderFormat.RF64\_WAV
- ARDOUR.HeaderFormat.MBWF

## 152.209.30 ARDOUR.InsertMergePolicy

- ARDOUR.InsertMergePolicy.Reject
- ARDOUR.InsertMergePolicy.Relax
- ARDOUR.InsertMergePolicy.Replace
- ARDOUR.InsertMergePolicy.TruncateExisting
- ARDOUR.InsertMergePolicy.TruncateAddition
- ARDOUR.InsertMergePolicy.Extend

## 152.209.31 ARDOUR.AFLPosition

- ARDOUR.AFLPosition.AFLFromBeforeProcessors
- ARDOUR.AFLPosition.AFLFromAfterProcessors

## 152.209.32 ARDOUR.PFLPosition

- ARDOUR.PFLPosition.PFLFromBeforeProcessors
- ARDOUR.PFLPosition.PFLFromAfterProcessors

## 152.209.33 ARDOUR.AutoReturnTarget

- ARDOUR.AutoReturnTarget.LastLocate
- ARDOUR.AutoReturnTarget.RangeSelectionStart
- ARDOUR.AutoReturnTarget.Loop
- ARDOUR.AutoReturnTarget.RegionSelectionStart

### 152.209.34 ARDOUR.FadeShape

- ARDOUR.FadeShape.FadeLinear
- ARDOUR.FadeShape.FadeFast
- ARDOUR.FadeShape.FadeSlow
- ARDOUR.FadeShape.FadeConstantPower
- ARDOUR.FadeShape.FadeSymmetric

### 152.209.35 ARDOUR.DenormalModel

- ARDOUR.DenormalModel.DenormalNone
- ARDOUR.DenormalModel.DenormalFTZ
- ARDOUR.DenormalModel.DenormalDAZ
- ARDOUR.DenormalModel.DenormalFTZDAZ

### 152.209.36 ARDOUR.BufferingPreset

- ARDOUR.BufferingPreset.Small
- ARDOUR.BufferingPreset.Medium
- ARDOUR.BufferingPreset.Large
- ARDOUR.BufferingPreset.Custom

### 152.209.37 ARDOUR.EditMode

- ARDOUR.EditMode.Slide
- ARDOUR.EditMode.Splice
- ARDOUR.EditMode.Ripple
- ARDOUR.EditMode.Lock

### 152.209.38 ARDOUR.AutoConnectOption

- ARDOUR.AutoConnectOption.ManualConnect
- ARDOUR.AutoConnectOption.AutoConnectPhysical
- ARDOUR.AutoConnectOption.AutoConnectMaster

### 152.209.39 ARDOUR.LayerModel

- ARDOUR.LayerModel.LaterHigher
- ARDOUR.LayerModel.Manual

## 152.209.40 ARDOUR.ListenPosition

- ARDOUR.ListenPosition.AfterFaderListen
- ARDOUR.ListenPosition.PreFaderListen

## 152.209.41 ARDOUR.MonitorModel

- ARDOUR.MonitorModel.HardwareMonitoring
- ARDOUR.MonitorModel.SoftwareMonitoring
- ARDOUR.MonitorModel.ExternalMonitoring

## 152.209.42 ARDOUR.RegionSelectionAfterSplit

- ARDOUR.RegionSelectionAfterSplit.None
- ARDOUR.RegionSelectionAfterSplit.NewlyCreatedLeft
- ARDOUR.RegionSelectionAfterSplit.NewlyCreatedRight
- ARDOUR.RegionSelectionAfterSplit.NewlyCreatedBoth
- ARDOUR.RegionSelectionAfterSplit.Existing
- ARDOUR.RegionSelectionAfterSplit.ExistingNewlyCreatedLeft
- ARDOUR.RegionSelectionAfterSplit.ExistingNewlyCreatedRight
- ARDOUR.RegionSelectionAfterSplit.ExistingNewlyCreatedBoth

## 152.209.43 ARDOUR.ShuttleBehaviour

- ARDOUR.ShuttleBehaviour.Sprung
- ARDOUR.ShuttleBehaviour.Wheel

## 152.209.44 ARDOUR.ShuttleUnits

- ARDOUR.ShuttleUnits.Percentage
- ARDOUR.ShuttleUnits.Semitones

## 152.209.45 ARDOUR.SyncSource

- ARDOUR.SyncSource.Engine
- ARDOUR.SyncSource.MTC
- ARDOUR.SyncSource.MIDIClock
- ARDOUR.SyncSource.LTC

## 152.209.46 ARDOUR.TracksAutoNamingRule

- ARDOUR.TracksAutoNamingRule.UseDefaultNames
- ARDOUR.TracksAutoNamingRule.NameAfterDriver

## 152.209.47 ARDOUR.Session.RecordState

- ARDOUR.Session.RecordState.Disabled
- ARDOUR.Session.RecordState.Enabled
- ARDOUR.Session.RecordState.Recording

## 152.209.48 ARDOUR.Location.Flags

- ARDOUR.LocationFlags.IsMark
- ARDOUR.LocationFlags.IsAutoPunch
- ARDOUR.LocationFlags.IsAutoLoop
- ARDOUR.LocationFlags.isHidden
- ARDOUR.LocationFlags.IsCDMarker
- ARDOUR.LocationFlags.IsRangeMarker
- ARDOUR.LocationFlags.IsSessionRange
- ARDOUR.LocationFlags.IsSkip
- ARDOUR.LocationFlags.IsSkipping

## 152.209.49 ARDOUR.DSP.Biquad.Type

- ARDOUR.DSP.BiquadType.LowPass
- ARDOUR.DSP.BiquadType.HighPass
- ARDOUR.DSP.BiquadType.BandPassSkirt
- ARDOUR.DSP.BiquadType.BandPass0dB
- ARDOUR.DSP.BiquadType.Notch
- ARDOUR.DSP.BiquadType.AllPass
- ARDOUR.DSP.BiquadType.Peaking
- ARDOUR.DSP.BiquadType.LowShelf
- ARDOUR.DSP.BiquadType.HighShelf

## 152.209.50 Cairo.LineCap

- Cairo.LineCap.Butt
- Cairo.LineCap.Round
- Cairo.LineCap.Square

## 152.209.51 Cairo.LineJoin

- Cairo.LineJoin.Miter
- Cairo.LineJoin.Round
- Cairo.LineJoin.Bevel

## 152.209.52 Cairo.Operator

- Cairo.Operator.Clear
- Cairo.Operator.Source
- Cairo.Operator.Over
- Cairo.Operator.Add

## 152.209.53 Cairo.Format

- Cairo.Format.ARGB32
- Cairo.Format.RGB24

## 152.209.54 Pango.EllipsizeMode

- Cairo.EllipsizeMode.None
- Cairo.EllipsizeMode.Start
- Cairo.EllipsizeMode.Middle
- Cairo.EllipsizeMode.End

## 152.209.55 Pango.WrapMode

- Cairo.WrapMode.Word
- Cairo.WrapMode.Char
- Cairo.WrapMode.WordChar

## 152.209.56 LuaDialog.Message.MessageType

- LuaDialog.MessageType.Info
- LuaDialog.MessageType.Warning
- LuaDialog.MessageType.Question
- LuaDialog.MessageType.Error

## 152.209.57 `LuaDialog.Message.ButtonType`

- `LuaDialog.ButtonType.OK`
- `LuaDialog.ButtonType.Close`
- `LuaDialog.ButtonType.Cancel`
- `LuaDialog.ButtonType.Yes_No`
- `LuaDialog.ButtonType.OK_Cancel`

## 152.209.58 `LuaDialog.Response`

- `LuaDialog.Response.OK`
- `LuaDialog.Response.Cancel`
- `LuaDialog.Response.Close`
- `LuaDialog.Response.Yes`
- `LuaDialog.Response.No`
- `LuaDialog.Response.None`

## 152.209.59 `RouteDialogs.InsertAt`

- `ArdourUI.InsertAt.BeforeSelection`
- `ArdourUI.InsertAt.AfterSelection`
- `ArdourUI.InsertAt.First`
- `ArdourUI.InsertAt.Last`

## 152.209.60 `ArdourMarker.Type`

- `ArdourUI.MarkerType.Mark`
- `ArdourUI.MarkerType.Tempo`
- `ArdourUI.MarkerType.Meter`
- `ArdourUI.MarkerType.SessionStart`
- `ArdourUI.MarkerType.SessionEnd`
- `ArdourUI.MarkerType.RangeStart`
- `ArdourUI.MarkerType.RangeEnd`
- `ArdourUI.MarkerType.LoopStart`
- `ArdourUI.MarkerType.LoopEnd`
- `ArdourUI.MarkerType.PunchIn`
- `ArdourUI.MarkerType.PunchOut`

## 152.209.61 Selection.Operation

- ArdourUI.SelectionOp.Toggle
- ArdourUI.SelectionOp.Set
- ArdourUI.SelectionOp.Extend
- ArdourUI.SelectionOp.Add

## 152.209.62 Editing.SnapType

- Editing.SnapToCDFrame
- Editing.SnapToTimecodeFrame
- Editing.SnapToTimecodeSeconds
- Editing.SnapToTimecodeMinutes
- Editing.SnapToSeconds
- Editing.SnapToMinutes
- Editing.SnapToBeatDiv128
- Editing.SnapToBeatDiv64
- Editing.SnapToBeatDiv32
- Editing.SnapToBeatDiv28
- Editing.SnapToBeatDiv24
- Editing.SnapToBeatDiv20
- Editing.SnapToBeatDiv16
- Editing.SnapToBeatDiv14
- Editing.SnapToBeatDiv12
- Editing.SnapToBeatDiv10
- Editing.SnapToBeatDiv8
- Editing.SnapToBeatDiv7
- Editing.SnapToBeatDiv6
- Editing.SnapToBeatDiv5
- Editing.SnapToBeatDiv4
- Editing.SnapToBeatDiv3
- Editing.SnapToBeatDiv2
- Editing.SnapToBeat
- Editing.SnapToBar
- Editing.SnapToMark
- Editing.SnapToRegionStart
- Editing.SnapToRegionEnd

- Editing.SnapToRegionSync
- Editing.SnapToRegionBoundary

## 152.209.63 Editing.SnapMode

- Editing.SnapOff
- Editing.SnapNormal
- Editing.SnapMagnetic

## 152.209.64 Editing.MouseMode

- Editing.MouseObject
- Editing.MouseRange
- Editing.MouseCut
- Editing.MouseTimeFX
- Editing.MouseAudition
- Editing.MouseDraw
- Editing.MouseContent

## 152.209.65 Editing.ZoomFocus

- Editing.ZoomFocusLeft
- Editing.ZoomFocusRight
- Editing.ZoomFocusCenter
- Editing.ZoomFocusPlayhead
- Editing.ZoomFocusMouse
- Editing.ZoomFocusEdit

## 152.209.66 Editing.DisplayControl

- Editing.FollowPlayhead
- Editing.ShowMeasures
- Editing.ShowWaveforms
- Editing.ShowWaveformsRecording

## 152.209.67 Editing.ImportMode

- Editing.ImportAsRegion
- Editing.ImportToTrack
- Editing.ImportAsTrack

- Editing.ImportAsTapeTrack

## 152.209.68 Editing.ImportPosition

- Editing.ImportAtTimestamp
- Editing.ImportAtEditPoint
- Editing.ImportAtPlayhead
- Editing.ImportAtStart

## 152.209.69 Editing.ImportDisposition

- Editing.ImportDistinctFiles
- Editing.ImportMergeFiles
- Editing.ImportSerializeFiles
- Editing.ImportDistinctChannels

## 152.209.70 LuaSignal.LuaSignal

- LuaSignal.ConfigChanged
- LuaSignal.EngineRunning
- LuaSignal.EngineStopped
- LuaSignal.EngineHalted
- LuaSignal.EngineDeviceListChanged
- LuaSignal.BufferSizeChanged
- LuaSignal.SampleRateChanged
- LuaSignal.FeedbackDetected
- LuaSignal.SuccessfulGraphSort
- LuaSignal.StartTimeChanged
- LuaSignal.EndTimeChanged
- LuaSignal.Exported
- LuaSignal.Change
- LuaSignal.SessionConfigChanged
- LuaSignal.TransportStateChange
- LuaSignal.DirtyChanged
- LuaSignal.StateSaved
- LuaSignal.Xrun
- LuaSignal.TransportLooped
- LuaSignal.SoloActive

- `LuaSignal.SoloChanged`
- `LuaSignal.IsolatedChanged`
- `LuaSignal.MonitorChanged`
- `LuaSignal.RecordStateChanged`
- `LuaSignal.RecordArmStateChanged`
- `LuaSignal.AudioLoopLocationChanged`
- `LuaSignal.AudioPunchLocationChanged`
- `LuaSignal.LocationsModified`
- `LuaSignal.AuditionActive`
- `LuaSignal.BundleAddedOrRemoved`
- `LuaSignal.PositionChanged`
- `LuaSignal.Located`
- `LuaSignal.RoutesReconnected`
- `LuaSignal.RouteAdded`
- `LuaSignal.RouteGroupPropertyChanged`
- `LuaSignal.RouteAddedToRouteGroup`
- `LuaSignal.RouteRemovedFromRouteGroup`
- `LuaSignal.StepEditStatusChange`
- `LuaSignal.RouteGroupAdded`
- `LuaSignal.RouteGroupRemoved`
- `LuaSignal.RouteGroupsReordered`
- `LuaSignal.PluginListChanged`
- `LuaSignal.PluginStatusesChanged`
- `LuaSignal.DiskOverrun`
- `LuaSignal.DiskUnderrun`
- `LuaSignal.RegionPropertyChanged`
- `LuaSignal.LuaTimerDS`
- `LuaSignal.SetSession`

## 152.210 Class Index

- `ARDOUR`
- `ARDOUR:Amp`
- `ARDOUR:AudioBackend`
- `ARDOUR:AudioBackendInfo`
- `ARDOUR:AudioBuffer`

- *ARDOUR:AudioEngine*
- *ARDOUR:AudioPlaylist*
- *ARDOUR:AudioPort*
- *ARDOUR:AudioRange*
- *ARDOUR:AudioRangeList*
- *ARDOUR:AudioRegion*
- *ARDOUR: AudioSource*
- *ARDOUR: AudioTrack*
- *ARDOUR: AudioTrackList*
- *ARDOUR: Automatable*
- *ARDOUR: AutomatableSequence*
- *ARDOUR: AutomationControl*
- *ARDOUR: AutomationList*
- *ARDOUR: BackendVector*
- *ARDOUR: BeatsFramesConverter*
- *ARDOUR: BufferSet*
- *ARDOUR: ChanCount*
- *ARDOUR: ChanMapping*
- *ARDOUR: ControlList*
- *ARDOUR: ControlListPtr*
- *ARDOUR.DSP*
- *ARDOUR:DSP:Biquad*
- *ARDOUR:DSP:DspShm*
- *ARDOUR:DSP:FFTSpectrum*
- *ARDOUR:DSP:LowPass*
- *ARDOUR: DataType*
- *ARDOUR: Delivery*
- *ARDOUR: DeviceStatus*
- *ARDOUR: DeviceStatusVector*
- *ARDOUR: DoubleBeatsFramesConverter*
- *ARDOUR: EventList*
- *ARDOUR: FileSource*
- *ARDOUR: FluidSynth*
- *ARDOUR: GainControl*
- *ARDOUR: IO*
- *ARDOUR: IOProcessor*

- *ARDOUR:InterThreadInfo*
- *ARDOUR:Location*
- *ARDOUR:LocationList*
- *ARDOUR:Locations*
- *ARDOUR.LuaAPI*
- *ARDOUR:LuaAPI:Vamp*
- *ARDOUR:LuaOSC:Address*
- *ARDOUR:LuaProc*
- *ARDOUR:LuaTableRef*
- *ARDOUR:Meter*
- *ARDOUR:MeterSection*
- *ARDOUR:MetricSection*
- *ARDOUR:MidiBuffer*
- *ARDOUR:MidiModel*
- *ARDOUR:MidiModel:DiffCommand*
- *ARDOUR:MidiModel:NoteDiffCommand*
- *ARDOUR:MidiPlaylist*
- *ARDOUR:MidiPort*
- *ARDOUR:MidiRegion*
- *ARDOUR:MidiSource*
- *ARDOUR:MidiTrack*
- *ARDOUR:MidiTrackList*
- *ARDOUR:MonitorProcessor*
- *ARDOUR:MusicFrame*
- *ARDOUR:MuteControl*
- *ARDOUR:NotePtrList*
- *ARDOUR:OwnedPropertyList*
- *ARDOUR:PannerShell*
- *ARDOUR:ParameterDescriptor*
- *ARDOUR:PeakMeter*
- *ARDOUR:PhaseControl*
- *ARDOUR:Playlist*
- *ARDOUR:Plugin*
- *ARDOUR:Plugin:IOPortDescription*
- *ARDOUR:PluginControl*
- *ARDOUR:PluginInfo*

- *ARDOUR:PluginInsert*
- *ARDOUR:Port*
- *ARDOUR:PortEngine*
- *ARDOUR:PortList*
- *ARDOUR:PortManager*
- *ARDOUR:PortSet*
- *ARDOUR:PresentationInfo*
- *ARDOUR:PresetRecord*
- *ARDOUR:PresetVector*
- *ARDOUR:Processor*
- *ARDOUR:ProcessorList*
- *ARDOUR:ProcessorVector*
- *ARDOUR:Progress*
- *ARDOUR:Properties:BoolProperty*
- *ARDOUR:Properties:FloatProperty*
- *ARDOUR:Properties:FrameposProperty*
- *ARDOUR:PropertyChange*
- *ARDOUR:PropertyList*
- *ARDOUR:RCCConfiguration*
- *ARDOUR:ReadOnlyControl*
- *ARDOUR:Readable*
- *ARDOUR:Region*
- *ARDOUR:RegionFactory*
- *ARDOUR:RegionList*
- *ARDOUR:RegionListPtr*
- *ARDOUR:RegionMap*
- *ARDOUR:RegionVector*
- *ARDOUR:Route*
- *ARDOUR:Route:ProcessorStreams*
- *ARDOUR:RouteGroup*
- *ARDOUR:RouteGroupList*
- *ARDOUR:RouteList*
- *ARDOUR:RouteListPtr*
- *ARDOUR:Session*
- *ARDOUR:SessionConfiguration*
- *ARDOUR:SessionObject*

- *ARDOUR:SessionObjectPtr*
- *ARDOUR:SideChain*
- *ARDOUR:Slavable*
- *ARDOUR:SlavableAutomationControl*
- *ARDOUR:SoloControl*
- *ARDOUR:SoloIsolateControl*
- *ARDOUR:SoloSafeControl*
- *ARDOUR:Source*
- *ARDOUR:SourceList*
- *ARDOUR:Stripable*
- *ARDOUR:StripableList*
- *ARDOUR:Tempo*
- *ARDOUR:TempoMap*
- *ARDOUR:TempoSection*
- *ARDOUR:Track*
- *ARDOUR:UnknownProcessor*
- *ARDOUR:VCA*
- *ARDOUR:VCAList*
- *ARDOUR:VCAManager*
- *ARDOUR:Weak AudioSourceList*
- *ARDOUR:WeakRouteList*
- *ARDOUR:WeakSourceList*
- *ArdourUI*
- *ArdourUI:ArdourMarker*
- *ArdourUI:ArdourMarkerList*
- *ArdourUI:AxisView*
- *ArdourUI:Editor*
- *ArdourUI:MarkerSelection*
- *ArdourUI:RegionSelection*
- *ArdourUI:RegionView*
- *ArdourUI:RouteTimeAxisView*
- *ArdourUI:RouteUI*
- *ArdourUI:Selectable*
- *ArdourUI:Selection*
- *ArdourUI:SelectionList*
- *ArdourUI:StripableTimeAxisView*

- *ArdourUI:TimeAxisView*
- *ArdourUI:TimeAxisViewItem*
- *ArdourUI:TimeSelection*
- *ArdourUI:TrackSelection*
- *ArdourUI:TrackViewList*
- *ArdourUI:TrackViewStdList*
- *C:ByteArray*
- *C:DoubleVector*
- *C:FloatArray*
- *C:FloatArrayVector*
- *C:FloatVector*
- *C:Int64List*
- *C:IntArray*
- *C:StringList*
- *C:StringVector*
- *Cairo:Context*
- *Cairo:ImageSurface*
- *Cairo:PangoLayout*
- *Evoral:Beats*
- *Evoral:Control*
- *Evoral:ControlEvent*
- *Evoral:ControlList*
- *Evoral:ControlSet*
- *Evoral:Event*
- *Evoral:NotePtr*
- *Evoral:Parameter*
- *Evoral:ParameterDescriptor*
- *Evoral:Range*
- *Evoral:Sequence*
- *LuaDialog:Dialog*
- *LuaDialog:Message*
- *LuaSignal:Set*
- *PBD*
- *PBD:Command*
- *PBD:Configuration*
- *PBD:Controllable*

- *PBD:ID*
- *PBD:IdVector*
- *PBD:RingBuffer8*
- *PBD:RingBufferF*
- *PBD:RingBufferI*
- *PBD:Stateful*
- *PBD:StatefulDestructible*
- *PBD:StatefulDestructiblePtr*
- *PBD:StatefulDiffCommand*
- *PBD:StatefulPtr*
- *PBD:XMLNode*
- *Timecode:BBT\_TIME*
- *Timecode:Time*
- *Vamp:Plugin*
- *Vamp:Plugin:Feature*
- *Vamp:Plugin:FeatureList*
- *Vamp:Plugin:FeatureSet*
- *Vamp:Plugin:OutputDescriptor*
- *Vamp:Plugin:OutputList*
- *Vamp:PluginBase*
- *Vamp:PluginBase:ParameterDescriptor*
- *Vamp:PluginBase:ParameterList*
- *Vamp:RealTime*
- *os*

Ardour 5.12 - Sat, 16 Sep 2017 16:18:16 +0200



---

CHAPTER  
**THREE**

---

**PART XIII - APPENDIX**



## 84 - LIST OF MENU ACTIONS

### 154.1 Menu actions

Every single menu item in Ardour's GUI is accessible by control surfaces or scripts.

The list below shows all available values of *action-name* as of Ardour 5.12. You can get the current list at any time by running Ardour with the -A flag.

Action Name	Menu Name
ProcessorMenu/ab_plugins	A/B Plugins
ProcessorMenu/activate_all	Activate All
ProcessorMenu/backspace	Delete
ProcessorMenu/clear	Clear (all)
ProcessorMenu/clear_post	Clear (post-fader)
ProcessorMenu/clear_pre	Clear (pre-fader)
ProcessorMenu/controls	Controls
ProcessorMenu/copy	Copy
ProcessorMenu/cut	Cut
ProcessorMenu/deactivate_all	Deactivate All
ProcessorMenu/delete	Delete
ProcessorMenu/deselectall	Deselect All
ProcessorMenu/edit	Edit...
ProcessorMenu/edit-generic	Edit with generic controls...
ProcessorMenu/manage-pins	Pin Connections...
ProcessorMenu/newaux	New Aux Send ...
ProcessorMenu/newinsert	New Insert
ProcessorMenu/newplugin	New Plugin
ProcessorMenu/newsend	New External Send ...
ProcessorMenu/paste	Paste
ProcessorMenu/rename	Rename
ProcessorMenu/selectall	Select All
ProcessorMenu/send_options	Send Options
Common/Hide	Hide
Common/NewMIDITracer	MIDI Tracer
Common/Quit	Quit
Common/Save	Save
Common/ToggleMaximalEditor	Maximise Editor Space
Common/ToggleMaximalMixer	Maximise Mixer Space
Common/ToggleMixerList	Toggle Mixer List

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Common/ToggleMonitorSection	Toggle Monitor Section Visibility
Common/ToggleRecordEnableTrack1	Toggle Record Enable Track 1
Common/ToggleRecordEnableTrack10	Toggle Record Enable Track 10
Common/ToggleRecordEnableTrack11	Toggle Record Enable Track 11
Common/ToggleRecordEnableTrack12	Toggle Record Enable Track 12
Common/ToggleRecordEnableTrack13	Toggle Record Enable Track 13
Common/ToggleRecordEnableTrack14	Toggle Record Enable Track 14
Common/ToggleRecordEnableTrack15	Toggle Record Enable Track 15
Common/ToggleRecordEnableTrack16	Toggle Record Enable Track 16
Common/ToggleRecordEnableTrack17	Toggle Record Enable Track 17
Common/ToggleRecordEnableTrack18	Toggle Record Enable Track 18
Common/ToggleRecordEnableTrack19	Toggle Record Enable Track 19
Common/ToggleRecordEnableTrack2	Toggle Record Enable Track 2
Common/ToggleRecordEnableTrack20	Toggle Record Enable Track 20
Common/ToggleRecordEnableTrack21	Toggle Record Enable Track 21
Common/ToggleRecordEnableTrack22	Toggle Record Enable Track 22
Common/ToggleRecordEnableTrack23	Toggle Record Enable Track 23
Common/ToggleRecordEnableTrack24	Toggle Record Enable Track 24
Common/ToggleRecordEnableTrack25	Toggle Record Enable Track 25
Common/ToggleRecordEnableTrack26	Toggle Record Enable Track 26
Common/ToggleRecordEnableTrack27	Toggle Record Enable Track 27
Common/ToggleRecordEnableTrack28	Toggle Record Enable Track 28
Common/ToggleRecordEnableTrack29	Toggle Record Enable Track 29
Common/ToggleRecordEnableTrack3	Toggle Record Enable Track 3
Common/ToggleRecordEnableTrack30	Toggle Record Enable Track 30
Common/ToggleRecordEnableTrack31	Toggle Record Enable Track 31
Common/ToggleRecordEnableTrack32	Toggle Record Enable Track 32
Common/ToggleRecordEnableTrack4	Toggle Record Enable Track 4
Common/ToggleRecordEnableTrack5	Toggle Record Enable Track 5
Common/ToggleRecordEnableTrack6	Toggle Record Enable Track 6
Common/ToggleRecordEnableTrack7	Toggle Record Enable Track 7
Common/ToggleRecordEnableTrack8	Toggle Record Enable Track 8
Common/ToggleRecordEnableTrack9	Toggle Record Enable Track 9
Common/add-location-from-playhead	Add Mark from Playhead
Common/addExistingAudioFiles	Import
Common/alt-finish-range	Finish Range
Common/alt-start-range	Start Range
Common/alternate-add-location-fro m-playhead	Add Mark from Playhead
Common/alternate-jump-backward-to -mark	Jump to Previous Mark
Common/alternate-jump-forward-to- mark	Jump to Next Mark
Common/alternate-remove-location- from-playhead	Remove Mark at Playhead
Common/attach-editor	Attach
Common/attach-mixer	Attach
Common/attach-preferences	Attach
Common/change-editor-visibility	Change
Common/change-mixer-visibility	Change
Common/change-preferences-visibil ity	Change
Common/chat	Chat
Common/cheat-sheet	Cheat Sheet

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Common/detach-editor	Detach
Common/detach-mixer	Detach
Common/detach-preferences	Detach
Common/finish-loop-range	Finish Loop Range
Common/finish-punch-range	Finish Punch Range
Common/finish-range	Finish Range
Common/finish-range-from-playhead	Finish Range from Playhead
Common/forums	User Forums
Common/hide-editor	Hide
Common/hide-mixer	Hide
Common/hide-preferences	Hide
Common/howto-report	How to Report a Bug
Common/jump-backward-to-mark	Jump to Previous Mark
Common/jump-forward-to-mark	Jump to Next Mark
Common/key-change-editor-visibility	Change
Common/key-change-mixer-visibility	Change
Common/key-change-preferences-visibility	Change
Common/manual	Manual
Common/menu-show-preferences	Preferences
Common/next-tab	Next Tab
Common/nudge-next-backward	Nudge Next Earlier
Common/nudge-next-forward	Nudge Next Later
Common/nudge-playhead-backward	Nudge Playhead Backward
Common/nudge-playhead-forward	Nudge Playhead Forward
Common/playhead-backward-to-grid	Playhead to Previous Grid
Common/playhead-forward-to-grid	Playhead to Next Grid
Common/previous-tab	Previous Tab
Common/reference	Reference
Common/remove-location-from-playhead	Remove Mark at Playhead
Common/set-session-end-from-playhead	Set Session End from Playhead
Common/set-session-start-from-playhead	Set Session Start from Playhead
Common/show-preferences	Show
Common/start-loop-range	Start Loop Range
Common/start-punch-range	Start Punch Range
Common/start-range	Start Range
Common/start-range-from-playhead	Start Range from Playhead
Common/toggle-editor-and-mixer	Toggle Editor & Mixer
Common/toggle-location-at-playhead	Toggle Mark at Playhead
Common/toggle-luawindow	Scripting
Common/toggle-meterbridge	Meterbridge
Common/tracker	Report a Bug
Common/website	Ardour Website
Common/website-dev	Ardour Development
MIDI/panic	Panic (Send MIDI all-notes-off)
Main/AddTrackBus	Add Track, Bus or VCA...
Main/Archive	Archive...
Main/CleanupPeakFiles	Reset Peak Files
Main/CleanupUnused	Clean-up Unused Sources...
Main/Close	Close

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Main/CloseVideo	Remove Video
Main/EditMetadata	Edit Metadata...
Main/Escape	Escape (deselect all)
Main/Export	Export
Main/ExportAudio	Export to Audio File(s)...
Main/ExportVideo	Export to Video File...
Main/FlushWastebasket	Flush Wastebasket
Main/ImportMetadata	Import Metadata...
Main/ManageTemplates	Templates
Main/Metadata	Metadata
Main/New	New...
Main/Open	Open...
Main/OpenVideo	Open Video...
Main/QuickSnapshotStay	Quick Snapshot (& keep working on current version) ...
Main/QuickSnapshotSwitch	Quick Snapshot (& switch to new version) ...
Main/Recent	Recent...
Main/Rename	Rename...
Main/SaveAs	Save As...
Main/SaveTemplate	Save Template...
Main/Scripting	Scripting
Main/SnapshotStay	Snapshot (& keep working on current version) ...
Main/SnapshotSwitch	Snapshot (& switch to new version) ...
Main/StemExport	Stem export...
Main/cancel-solo	Cancel Solo
Main/close-current-dialog	Close Current Dialog
Main/duplicate-routes	Duplicate Tracks/Busses...
Main_menu/AudioFileFormat	Audio File Format
Main_menu/AudioFileFormatData	Sample Format
Main_menu/AudioFileFormatHeader	File Type
Main_menu/Cleanup	Clean-up
Main_menu/ControlSurfaces	Control Surfaces
Main_menu/Denormals	Denormal Handling
Main_menu/DetachMenu	Detach
Main_menu/EditorMenu	Editor
Main_menu/Help	Help
Main_menu/KeyMouseActions	Misc. Shortcuts
Main_menu/Metering	Metering
Main_menu/MeteringFallOffRate	Fall Off Rate
Main_menu/MeteringHoldTime	Hold Time
Main_menu/MixerMenu	Mixer
Main_menu/Plugins	Plugins
Main_menu/PrefsMenu	Preferences
Main_menu/Session	Session
Main_menu/Sync	Sync
Main_menu/TransportOptions	Options
Main_menu/WindowMenu	Window
Options/SendMMC	Send MMC
Options/SendMTC	Send MTC
Options/SendMidiClock	Send MIDI Clock

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Options/UseMMC	Use MMC
Transport/Forward	Forward
Transport/ForwardFast	Forward (Fast)
Transport/ForwardSlow	Forward (Slow)
Transport/GotoEnd	Go to End
Transport/GotoStart	Go to Start
Transport/GotoWallClock	Go to Wall Clock
Transport/GotoZero	Go to Zero
Transport/Loop	Play Loop Range
Transport/PlayPreroll	Play w/Preroll
Transport/PlaySelection	Play Selection
Transport/Record	Enable Record
Transport/RecordCountIn	Record w/Count-In
Transport/RecordPreroll	Record w/Preroll
Transport/Rewind	Rewind
Transport/RewindFast	Rewind (Fast)
Transport/RewindSlow	Rewind (Slow)
Transport/Roll	Roll
Transport/SessionMonitorDisk	All Disk
Transport/SessionMonitorIn	All Input
Transport/Stop	Stop
Transport/ToggleAutoInput	Auto Input
Transport/ToggleAutoPlay	Auto Play
Transport/ToggleAutoReturn	Auto Return
Transport/ToggleClick	Click
Transport/ToggleExternalSync	Use External Positional Sync Source
Transport/ToggleFollowEdits	Follow Range
Transport/TogglePunch	Punch In/Out
Transport/TogglePunchIn	Punch In
Transport/TogglePunchOut	Punch Out
Transport/ToggleRoll	Start/Stop
Transport/ToggleRollForgetCapture	Stop and Forget Capture
Transport/ToggleRollMaybe	Start/Continue/Stop
Transport/ToggleTimeMaster	Time Master
Transport/ToggleVideoSync	Sync Startup to Video
Transport/TransitionToReverse	Transition to Reverse
Transport/TransitionToRoll	Transition to Roll
Transport/Transport	Transport
Transport/alternate-GotoStart	Go to Start
Transport/alternate-ToggleRoll	Start/Stop
Transport/alternate-numpad-decima 1	Numpad Decimal
Transport/alternate-record-roll	Start Recording
Transport/focus-on-clock	Focus On Clock
Transport/numpad-0	Numpad 0
Transport/numpad-1	Numpad 1
Transport/numpad-2	Numpad 2
Transport/numpad-3	Numpad 3
Transport/numpad-4	Numpad 4
Transport/numpad-5	Numpad 5

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Transport/numpad-6	Numpad 6
Transport/numpad-7	Numpad 7
Transport/numpad-8	Numpad 8
Transport/numpad-9	Numpad 9
Transport/numpad-decimal	Numpad Decimal
Transport/primary-clock-bbt	Bars & Beats
Transport/primary-clock-minsec	Minutes & Seconds
Transport/primary-clock-samples	Samples
Transport/primary-clock-timecode	Timecode
Transport/record-roll	Start Recording
Transport/secondary-clock-bbt	Bars & Beats
Transport/secondary-clock-minsec	Minutes & Seconds
Transport/secondary-clock-samples	Samples
Transport/secondary-clock-timecode	Timecode
Window/show-mixer	Show Mixer
Window/toggle-about	About
Window/toggle-add-routes	Add Tracks/Busses
Window/toggle-add-video	Add Video
Window/toggle-audio-connection-manager	Audio Connections
Window/toggle-audio-midi-setup	Audio/MIDI Setup
Window/toggle-big-clock	Big Clock
Window/toggle-bundle-manager	Bundle Manager
Window/toggle-idle-o-meter	Idle'o'Meter
Window/toggle-inspector	Tracks and Busses
Window/toggle-key-editor	Keyboard Shortcuts
Window/toggle-locations	Locations
Window/toggle-midi-connection-manager	MIDI Connections
Window/toggle-script-manager	Script Manager
Window/toggle-session-options-editor	Properties
Window/toggle-speaker-config	Speaker Configuration
Window/toggle-video-export	Video Export Dialog
Editor/SnapMode	Snap Mode
Editor/SnapTo	Snap to
Editor/ToggleGroupTabs	Show Group Tabs
Editor/ToggleJadeo	Video Monitor
Editor/ToggleMeasureVisibility	Show Measure Lines
Editor/ToggleSummary	Show Summary
Editor/addExistingPTFiles	Import PT session
Editor/addExternalAudioToRegionList	Import to Region List...
Editor/alternate-alternate-redo	Redo
Editor/alternate-editor-delete	Delete
Editor/alternate-redo	Redo
Editor/alternate-select-all-after-edit-cursor	Select All After Edit Point
Editor/alternate-select-all-before-edit-cursor	Select All Before Edit Point
Editor/alternate-tab-to-transient-backwards	Move to Previous Transient
Editor/alternate-tab-to-transient-forwards	Move to Next Transient
Editor/bring-into-session	Bring all media into session folder
Editor/center-edit-cursor	Center Edit Point
Editor/center-playhead	Center Playhead

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Editor/crop	Crop
Editor/cycle-edit-mode	Cycle Edit Mode
Editor/cycle-edit-point	Change Edit Point
Editor/cycle-edit-point-with-marker	Change Edit Point Including Marker
Editor/cycle-snap-mode	Next Snap Mode
Editor/cycle-zoom-focus	Next Zoom Focus
Editor/deselect-all	Deselect All
Editor/duplicate	Duplicate
Editor/edit-at-mouse	Mouse
Editor/edit-at-playhead	Playhead
Editor/edit-at-selected-marker	Marker
Editor/edit-current-meter	Edit Current Meter
Editor/edit-current-tempo	Edit Current Tempo
Editor/edit-cursor-to-next-region -end	To Next Region End
Editor/edit-cursor-to-next-region -start	To Next Region Start
Editor/edit-cursor-to-next-region -sync	To Next Region Sync
Editor/edit-cursor-to-previous-region-end	To Previous Region End
Editor/edit-cursor-to-previous-region-start	To Previous Region Start
Editor/edit-cursor-to-previous-region-sync	To Previous Region Sync
Editor/edit-cursor-to-range-end	To Range End
Editor/edit-cursor-to-range-start	To Range Start
Editor/edit-to-playhead	Active Mark to Playhead
Editor/editor-copy	Copy
Editor/editor-crop	Crop
Editor/editor-cut	Cut
Editor/editor-delete	Delete
Editor/editor-fade-range	Fade Range Selection
Editor/editor-paste	Paste
Editor/editor-separate	Separate
Editor/expand-tracks	Expand Track Height
Editor/export-audio	Export Audio
Editor/export-range	Export Range
Editor/fit-selection	Fit Selection (Vertical)
Editor/fit_16_tracks	Fit 16 Tracks
Editor/fit_1_track	Fit 1 Track
Editor/fit_2_tracks	Fit 2 Tracks
Editor/fit_32_tracks	Fit 32 Tracks
Editor/fit_4_tracks	Fit 4 Tracks
Editor/fit_8_tracks	Fit 8 Tracks
Editor/fit_all_tracks	Fit All Tracks
Editor/goto-mark-1	Locate to Mark 1
Editor/goto-mark-2	Locate to Mark 2
Editor/goto-mark-3	Locate to Mark 3
Editor/goto-mark-4	Locate to Mark 4
Editor/goto-mark-5	Locate to Mark 5
Editor/goto-mark-6	Locate to Mark 6
Editor/goto-mark-7	Locate to Mark 7
Editor/goto-mark-8	Locate to Mark 8
Editor/goto-mark-9	Locate to Mark 9

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Editor/goto-visual-state-1	Go to View 1
Editor/goto-visual-state-10	Go to View 10
Editor/goto-visual-state-11	Go to View 11
Editor/goto-visual-state-12	Go to View 12
Editor/goto-visual-state-2	Go to View 2
Editor/goto-visual-state-3	Go to View 3
Editor/goto-visual-state-4	Go to View 4
Editor/goto-visual-state-5	Go to View 5
Editor/goto-visual-state-6	Go to View 6
Editor/goto-visual-state-7	Go to View 7
Editor/goto-visual-state-8	Go to View 8
Editor/goto-visual-state-9	Go to View 9
Editor/importFromSession	Import from Session
Editor/insert-time	Insert Time
Editor/invert-selection	Invert Selection
Editor/lock	Lock
Editor/main-menu-play-selected-regions	Play Selected Regions
Editor/move-range-end-to-next-region-boundary	Move Range End to Next Region Boundary
Editor/move-range-end-to-previous-region-boundary	Move Range End to Previous Region Boundary
Editor/move-range-start-to-next-region-boundary	Move Range Start to Next Region Boundary
Editor/move-range-start-to-previous-region-boundary	Move Range Start to Previous Region Boundary
Editor/move-selected-tracks-down	Move Selected Tracks Down
Editor/move-selected-tracks-up	Move Selected Tracks Up
Editor/multi-duplicate	Multi-Duplicate...
Editor/next-snap-choice	Next Snap Choice
Editor/next-snap-choice-music-only	Next Musical Snap Choice
Editor/play-edit-range	Play Edit Range
Editor/play-from-edit-point	Play from Edit Point
Editor/play-from-edit-point-and-return	Play from Edit Point and Return
Editor/playhead-to-edit	Playhead to Active Mark
Editor/playhead-to-next-region-boundary	Playhead to Next Region Boundary
Editor/playhead-to-next-region-boundary-noselection	Playhead to Next Region Boundary (No Track Selection)
Editor/playhead-to-next-region-end	Playhead to Next Region End
Editor/playhead-to-next-region-start	Playhead to Next Region Start
Editor/playhead-to-next-region-sync	Playhead to Next Region Sync
Editor/playhead-to-previous-region-boundary	Playhead to Previous Region Boundary
Editor/playhead-to-previous-region-boundary-noselection	Playhead to Previous Region Boundary (No Track Selection)
Editor/playhead-to-previous-region-end	Playhead to Previous Region End
Editor/playhead-to-previous-region-start	Playhead to Previous Region Start
Editor/playhead-to-previous-region-sync	Playhead to Previous Region Sync
Editor/playhead-to-range-end	Playhead to Range End
Editor/playhead-to-range-start	Playhead to Range Start
Editor/prev-snap-choice	Previous Snap Choice
Editor/prev-snap-choice-music-only	Previous Musical Snap Choice
Editor/quantize	Quantize
Editor/redo	Redo
Editor/redo-last-selection-op	Redo Selection Change
Editor/remove-last-capture	Remove Last Capture
Editor/remove-time	Remove Time

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Editor/remove-track	Remove
Editor/save-visual-state-1	Save View 1
Editor/save-visual-state-10	Save View 10
Editor/save-visual-state-11	Save View 11
Editor/save-visual-state-12	Save View 12
Editor/save-visual-state-2	Save View 2
Editor/save-visual-state-3	Save View 3
Editor/save-visual-state-4	Save View 4
Editor/save-visual-state-5	Save View 5
Editor/save-visual-state-6	Save View 6
Editor/save-visual-state-7	Save View 7
Editor/save-visual-state-8	Save View 8
Editor/save-visual-state-9	Save View 9
Editor/script-action-1	Unset #1
Editor/script-action-2	Unset #2
Editor/script-action-3	Unset #3
Editor/script-action-4	Unset #4
Editor/script-action-5	Unset #5
Editor/script-action-6	Unset #6
Editor/script-action-7	Unset #7
Editor/script-action-8	Unset #8
Editor/script-action-9	Unset #9
Editor/scroll-backward	Scroll Backward
Editor/scroll-forward	Scroll Forward
Editor/scroll-playhead-backward	Playhead Backward
Editor/scroll-playhead-forward	Playhead Forward
Editor/scroll-tracks-down	Scroll Tracks Down
Editor/scroll-tracks-up	Scroll Tracks Up
Editor/select-all-after-edit-curs or	Select All After Edit Point
Editor/select-all-before-edit-cur sor	Select All Before Edit Point
Editor/select-all-between-cursors	Select All Overlapping Edit Range
Editor/select-all-in-loop-range	Select All in Loop Range
Editor/select-all-in-punch-range	Select All in Punch Range
Editor/select-all-objects	Select All Objects
Editor/select-all-tracks	Select All Tracks
Editor/select-all-within-cursors	Select All Inside Edit Range
Editor/select-from-regions	Select Range to Selected Regions
Editor/select-loop-range	Select Range to Loop Range
Editor/select-next-route	Select Next Track or Bus
Editor/select-next-stripable	Select Next Strip
Editor/select-prev-route	Select Previous Track or Bus
Editor/select-prev-stripable	Select Previous Strip
Editor/select-punch-range	Select Range to Punch Range
Editor/select-range-between-cursors	Select Edit Range
Editor/select-topmost	Select Topmost Track
Editor/selected-marker-to-next-region-boundary	To Next Region Boundary
Editor/selected-marker-to-next-region-boundary-noselection	To Next Region Boundary (No Track Selection)
Editor/selected-marker-to-previous-region-boundary	To Previous Region Boundary
Editor/selected-marker-to-previous-region-boundary-noselection	To Previous Region Boundary (No Track Selection)

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Editor/separate-from-loop	Separate Using Loop Range
Editor/separate-from-punch	Separate Using Punch Range
Editor/set-auto-punch-range	Set Auto Punch In/Out from Playhead
Editor/set-edit-lock	Lock
Editor/set-edit-point	Active Marker to Mouse
Editor/set-edit-ripple	Ripple
Editor/set-edit-slide	Slide
Editor/set-loop-from>Edit-range	Set Loop from Selection
Editor/set-playhead	Playhead to Mouse
Editor/set-punch-from>Edit-range	Set Punch from Selection
Editor/set-session-from>Edit-range	Set Session Start/End from Selection
Editor/set-tempo-from>Edit-range	Set Tempo from Edit Range = Bar
Editor/show-editor-list	Show Editor List
Editor/show-editor-mixer	Show Editor Mixer
Editor/show-marker-lines	Show Marker Lines
Editor/shrink-tracks	Shrink Track Height
Editor/snap-magnetic	Magnetic
Editor/snap-normal	Grid
Editor/snap-off	No Grid
Editor/sound-midi-notes	Sound Selected MIDI Notes
Editor/split-region	Split/Separate
Editor/step-mouse-mode	Step Mouse Mode
Editor/step-tracks-down	Step Tracks Down
Editor/step-tracks-up	Step Tracks Up
Editor/tab-to-transient-backwards	Move to Previous Transient
Editor/tab-to-transient-forwards	Move to Next Transient
Editor/temporal-zoom-in	Zoom In
Editor/temporal-zoom-out	Zoom Out
Editor/toggle-follow-playhead	Follow Playhead
Editor/toggle-log-window	Log
Editor/toggle-midi-input-active	Toggle MIDI Input Active for Editor-Selected Tracks/E
Editor/toggle-skip-playback	Use Skip Ranges
Editor/toggle-stationary-playhead	Stationary Playhead
Editor/toggle-track-active	Toggle Active
Editor/toggle-vmon-frame	Frame number
Editor/toggle-vmon-fullscreen	Fullscreen
Editor/toggle-vmon-letterbox	Letterbox
Editor/toggle-vmon-on-top	Always on Top
Editor/toggle-vmon-osdbg	Timecode Background
Editor/toggle-vmon-timecode	Timecode
Editor/toggle-zoom	Toggle Zoom State
Editor/track-height-large	Large
Editor/track-height-larger	Larger
Editor/track-height-largest	Largest
Editor/track-height-normal	Normal
Editor/track-height-small	Small
Editor/track-mute-toggle	Toggle Mute
Editor/track-record-enable-toggle	Toggle Record Enable
Editor/track-solo-isolate-toggle	Toggle Solo Isolate

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Editor/track-solo-toggle	Toggle Solo
Editor/undo	Undo
Editor/undo-last-selection-op	Undo Selection Change
Editor/zoom-to-extents	Zoom to Extents
Editor/zoom-to-selection	Zoom to Selection
Editor/zoom-to-selection-horiz	Zoom to Selection (Horizontal)
Editor/zoom-to-session	Zoom to Session
Editor/zoom-vmon-100	Original Size
Editor/zoom_100_ms	Zoom to 100 ms
Editor/zoom_10_min	Zoom to 10 min
Editor/zoom_10_ms	Zoom to 10 ms
Editor/zoom_10_sec	Zoom to 10 sec
Editor/zoom_1_min	Zoom to 1 min
Editor/zoom_1_sec	Zoom to 1 sec
Editor/zoom_5_min	Zoom to 5 min
EditorMenu/AlignMenu	Align
EditorMenu/Autoconnect	Autoconnect
EditorMenu/Crossfades	Crossfades
EditorMenu/Edit	Edit
EditorMenu/EditCursorMovementOptions	Move Selected Marker
EditorMenu/EditPointMenu	Edit Point
EditorMenu/EditSelectRangeOptions	Select Range Operations
EditorMenu/EditSelectRegionOptions	Select Regions
EditorMenu/FadeMenu	Fade
EditorMenu/LatchMenu	Latch
EditorMenu/Link	Link
EditorMenu/LocateToMarker	Locate to Markers
EditorMenu/LuaScripts	Lua Scripts
EditorMenu/MIDI	MIDI Options
EditorMenu/MarkerMenu	Markers
EditorMenu/MeterFalloff	Meter falloff
EditorMenu/MeterHold	Meter hold
EditorMenu/MiscOptions	Misc Options
EditorMenu/Monitoring	Monitoring
EditorMenu/MoveActiveMarkMenu	Active Mark
EditorMenu/MovePlayHeadMenu	Playhead
EditorMenu/PlayMenu	Play
EditorMenu/PrimaryClockMenu	Primary Clock
EditorMenu/Pullup	Pullup / Pulldown
EditorMenu/RegionEditOps	Region operations
EditorMenu/RegionGainMenu	Gain
EditorMenu/RegionMenu	Region
EditorMenu/RegionMenuDuplicate	Duplicate
EditorMenu/RegionMenuEdit	Edit
EditorMenu/RegionMenuFades	Fades
EditorMenu/RegionMenuGain	Gain
EditorMenu/RegionMenuLayering	Layering
EditorMenu/RegionMenuMIDI	MIDI
EditorMenu/RegionMenuPosition	Position

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
EditorMenu/RegionMenuRanges	Ranges
EditorMenu/RegionMenuTrim	Trim
EditorMenu/RulerMenu	Rulers
EditorMenu/SavedViewMenu	Views
EditorMenu/ScrollMenu	Scroll
EditorMenu/SecondaryClockMenu	Secondary Clock
EditorMenu>Select	Select
EditorMenu>SelectMenu	Select
EditorMenu/SeparateMenu	Separate
EditorMenu/SetLoopMenu	Loop
EditorMenu/SetPunchMenu	Punch
EditorMenu/Solo	Solo
EditorMenu/Subframes	Subframes
EditorMenu/SyncMenu	Sync
EditorMenu/TempoMenu	Tempo
EditorMenu/Timecode	Timecode fps
EditorMenu/Tools	Tools
EditorMenu/TrackHeightMenu	Height
EditorMenu/TrackMenu	Track
EditorMenu/VideoMonitorMenu	Video Monitor
EditorMenu/View	View
EditorMenu/ZoomFocus	Zoom Focus
EditorMenu/ZoomFocusMenu	Zoom Focus
EditorMenu/ZoomMenu	Zoom
MouseMode/set-mouse-mode-audition	Audition Tool
MouseMode/set-mouse-mode-content	Content Tool
MouseMode/set-mouse-mode-cut	Cut Tool
MouseMode/set-mouse-mode-draw	Note Drawing Tool
MouseMode/set-mouse-mode-object	Object Tool
MouseMode/set-mouse-mode-object-range	Smart Object Mode
MouseMode/set-mouse-mode-range	Range Tool
MouseMode/set-mouse-mode-timefx	Time FX Tool
Region/add-range-marker-from-region	Add Single Range Marker
Region/add-range-markers-from-region	Add Range Marker Per Region
Region/align-regions-end	Align End
Region/align-regions-end-relative	Align End Relative
Region/align-regions-start	Align Start
Region/align-regions-start-relative	Align Start Relative
Region/align-regions-sync	Align Sync
Region/align-regions-sync-relative	Align Sync Relative
Region/alternate-nudge-backward	Nudge Earlier
Region/alternate-nudge-forward	Nudge Later
Region/alternate-set-fade-in-length	Set Fade In Length
Region/alternate-set-fade-out-length	Set Fade Out Length
Region/boost-region-gain	Boost Gain
Region/bounce-regions-processed	Bounce (with processing)
Region/bounce-regions-unprocessed	Bounce (without processing)
Region/choose-top-region	Choose Top...
Region/choose-top-region-context-menu	Choose Top...

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Region/close-region-gaps	Close Gaps
Region/combine-regions	Combine
Region/cut-region-gain	Cut Gain
Region/duplicate-region	Duplicate
Region/export-region	Export...
Region/fork-region	Unlink from other copies
Region/insert-patch-change	Insert Patch Change...
Region/insert-patch-change-contex t	Insert Patch Change...
Region/insert-region-from-region- list	Insert Region from Region List
Region/legatize-region	Legatize
Region/loop-region	Loop
Region/loudness-analyze-region	Loudness Analysis...
Region/lower-region	Lower
Region/lower-region-to-bottom	Lower to Bottom
Region/multi-duplicate-region	Multi-Duplicate...
Region/naturalize-region	Move to Original Position
Region/normalize-region	Normalize...
Region/nudge-backward	Nudge Earlier
Region/nudge-backward-by-capture- offset	Nudge Earlier by Capture Offset
Region/nudge-forward	Nudge Later
Region/nudge-forward-by-capture-o ffset	Nudge Later by Capture Offset
Region/pitch-shift-region	Pitch Shift...
Region/place-transient	Place Transient
Region/play-selected-regions	Play selected Regions
Region/quantize-region	Quantize...
Region/raise-region	Raise
Region/raise-region-to-top	Raise to Top
Region/region-fill-track	Fill Track
Region/remove-overlap	Remove Overlap
Region/remove-region	Remove
Region/remove-region-sync	Remove Sync
Region/rename-region	Rename...
Region/reset-region-gain	Reset Gain
Region/reset-region-gain-envelope s	Reset Envelope
Region/reset-region-scale-amplitu de	Reset Gain
Region/reverse-region	Reverse
Region/separate-under-region	Separate Under
Region/sequence-regions	Sequence Regions
Region/set-fade-in-length	Set Fade In Length
Region/set-fade-out-length	Set Fade Out Length
Region/set-loop-from-region	Set Loop Range
Region/set-punch-from-region	Set Punch
Region/set-region-sync-position	Set Sync Position
Region/set-selection-from-region	Set Range Selection
Region/set-tempo-from-region	Set Tempo from Region = Bar
Region/show-region-list-editor	List Editor...
Region/show-region-properties	Properties...
Region/show-rhythm-ferret	Rhythm Ferret...
Region/snap-regions-to-grid	Snap Position to Grid

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Region/spectral-analyze-region	Spectral Analysis...
Region/split-multichannel-region	Make Mono Regions
Region/split-region-at-transients	Split at Percussion Onsets
Region/strip-region-silence	Strip Silence...
Region/toggle-opaque-region	Opaque
Region/toggle-region-fade-in	Fade In
Region/toggle-region-fade-out	Fade Out
Region/toggle-region-fades	Fades
Region/toggle-region-gain-envelop e-active	Envelope Active
Region/toggle-region-lock	Lock
Region/toggle-region-lock-style	Glue to Bars and Beats
Region/toggle-region-mute	Mute
Region/toggle-region-video-lock	Lock to Video
Region/transform-region	Transform...
Region/transpose-region	Transpose...
Region/trim-back	Trim End at Edit Point
Region/trim-front	Trim Start at Edit Point
Region/trim-region-to-loop	Trim to Loop
Region/trim-region-to-punch	Trim to Punch
Region/trim-to-next-region	Trim to Next
Region/trim-to-previous-region	Trim to Previous
Region/uncombine-regions	Uncombine
RegionList/RegionListSort	Sort
RegionList/SortAscending	Ascending
RegionList/SortByRegionEndinFile	By Region End in File
RegionList/SortByRegionLength	By Region Length
RegionList/SortByRegionName	By Region Name
RegionList/SortByRegionPosition	By Region Position
RegionList/SortByRegionStartinFil e	By Region Start in File
RegionList/SortByRegionTimestamp	By Region Timestamp
RegionList/SortBySourceFileCreati onDate	By Source File Creation Date
RegionList/SortBySourceFileLength	By Source File Length
RegionList/SortBySourceFileName	By Source File Name
RegionList/SortBySourceFilesystem	By Source Filesystem
RegionList/SortDescending	Descending
RegionList/removeUnusedRegions	Remove Unused
RegionList/rlAudition	Audition
RegionList/rlHide	Hide
RegionList/rlShow	Show
RegionList/rlShowAll	Show All
RegionList/rlShowAuto	Show Automatic Regions
Rulers/toggle-bbt-ruler	Bars & Beats
Rulers/toggle-cd-marker-ruler	CD Markers
Rulers/toggle-loop-punch-ruler	Loop/Punch
Rulers/toggle-marker-ruler	Markers
Rulers/toggle-meter-ruler	Meter
Rulers/toggle-minsec-ruler	Min:Sec
Rulers/toggle-range-ruler	Ranges
Rulers/toggle-samples-ruler	Samples

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Rulers/toggle-tempo-ruler	Tempo
Rulers/toggle-timecode-ruler	Timecode
Rulers/toggle-video-ruler	Video
Snap/snap-to-asixteenthbeat	Snap to Sixteenths
Snap/snap-to-bar	Snap to Bar
Snap/snap-to-beat	Snap to Beat
Snap/snap-to-cd-frame	Snap to CD Frame
Snap/snap-to-eighths	Snap to Eighths
Snap/snap-to-fifths	Snap to Fifths
Snap/snap-to-fourteenths	Snap to Fourteenths
Snap/snap-to-halves	Snap to Halves
Snap/snap-to-mark	Snap to Mark
Snap/snap-to-minutes	Snap to Minutes
Snap/snap-to-onetwentyeighths	Snap to One Twenty Eighths
Snap/snap-to-quarters	Snap to Quarters
Snap/snap-to-region-boundary	Snap to Region Boundary
Snap/snap-to-region-end	Snap to Region End
Snap/snap-to-region-start	Snap to Region Start
Snap/snap-to-region-sync	Snap to Region Sync
Snap/snap-to-seconds	Snap to Seconds
Snap/snap-to-sevenths	Snap to Sevenths
Snap/snap-to-sixths	Snap to Sixths
Snap/snap-to-sixtyfourths	Snap to Sixty Fourths
Snap/snap-to-tenths	Snap to Tenths
Snap/snap-to-thirds	Snap to Thirds
Snap/snap-to-thirtyseconds	Snap to Thirty Seconds
Snap/snap-to-timecode-frame	Snap to Timecode Frame
Snap/snap-to-timecode-minutes	Snap to Timecode Minutes
Snap/snap-to-timecode-seconds	Snap to Timecode Seconds
Snap/snap-to-twelfths	Snap to Twelfths
Snap/snap-to-twentieths	Snap to Twentieths
Snap/snap-to-twentyeighths	Snap to Twenty Eighths
Snap/snap-to-twentyfourths	Snap to Twenty Fourths
Zoom/zoom-focus-center	Zoom Focus Center
Zoom/zoom-focus-edit	Zoom Focus Edit Point
Zoom/zoom-focus-left	Zoom Focus Left
Zoom/zoom-focus-mouse	Zoom Focus Mouse
Zoom/zoom-focus-playhead	Zoom Focus Playhead
Zoom/zoom-focus-right	Zoom Focus Right
Mixer/ab-plugins	Toggle Selected Plugins
Mixer/copy-processors	Copy Selected Processors
Mixer/cut-processors	Cut Selected Processors
Mixer/decrement-gain	Increase Gain on Mixer-Selected Tracks/Busses
Mixer/delete-processors	Delete Selected Processors
Mixer/increment-gain	Decrease Gain on Mixer-Selected Tracks/Busses
Mixer/mute	Toggle Mute on Mixer-Selected Tracks/Busses
Mixer/paste-processors	Paste Selected Processors
Mixer/recenable	Toggle Rec-enable on Mixer-Selected Tracks/Busses
Mixer scroll-left	Scroll Mixer Window to the left

Continued on next page

Table 1 – continued from previous page

Action Name	Menu Name
Mixer/scroll-right	Scroll Mixer Window to the right
Mixer/select-all-processors	Select All (visible) Processors
Mixer/select-all-tracks	Select All Tracks
Mixer/select-none	Deselect all strips and processors
Mixer/show-editor	Show Editor
Mixer/solo	Toggle Solo on Mixer-Selected Tracks/Busses
Mixer/toggle-midi-input-active	Toggle MIDI Input Active for Mixer-Selected Tracks/Busses
Mixer/toggle-processors	Toggle Selected Processors
Mixer/unity-gain	Set Gain to 0dB on Mixer-Selected Tracks/Busses

---

**CHAPTER****FIVE**

---

**85 - ARDOUR MONITOR MODES**

The table below details what will be seen on the meter and heard on the monitor according to Ardour's settings.

Ref	Monitoring Mode (System Prefs)	Tape Machine Mode (System Prefs)	Track Rec Enable	Master Rec Enable
1	Ardour	Off	Off	Off
2	Ardour	Off	Off	Off
3	Ardour	Off	Off	Off
4	Ardour	Off	Off	Off
5	Ardour	Off	Off	On
6	Ardour	Off	Off	On
7	Ardour	Off	Off	On
8	Ardour	Off	Off	On
9	Ardour	Off	On	Off
10	Ardour	Off	On	Off
11	Ardour	Off	On	Off
12	Ardour	Off	On	Off
13	Ardour	Off	On	On
14	Ardour	Off	On	On
15	Ardour	Off	On	On
16	Ardour	Off	On	On
17	Ardour	On	Off	Off
18	Ardour	On	Off	Off
19	Ardour	On	Off	Off
20	Ardour	On	Off	Off
21	Ardour	On	Off	On
22	Ardour	On	Off	On
23	Ardour	On	Off	On
24	Ardour	On	Off	On
25	Ardour	On	On	Off
26	Ardour	On	On	Off
27	Ardour	On	On	Off
28	Ardour	On	On	Off
29	Ardour	On	On	On
30	Ardour	On	On	On
31	Ardour	On	On	On
32	Ardour	On	On	On
33	Audio Hardware	Off	Off	Off
34	Audio Hardware	Off	Off	Off

Table 1 – continued from previous page

Ref	Monitoring Mode (System Prefs)	Tape Machine Mode (System Prefs)	Track Rec Enabled	Master Rec Enabled
35	Audio Hardware	Off	Off	Off
36	Audio Hardware	Off	Off	Off
37	Audio Hardware	Off	Off	On
38	Audio Hardware	Off	Off	On
39	Audio Hardware	Off	Off	On
40	Audio Hardware	Off	Off	On
41	Audio Hardware	Off	On	Off
42	Audio Hardware	Off	On	Off
43	Audio Hardware	Off	On	Off
44	Audio Hardware	Off	On	Off
45	Audio Hardware	Off	On	On
46	Audio Hardware	Off	On	On
47	Audio Hardware	Off	On	On
48	Audio Hardware	Off	On	On
49	Audio Hardware	On	Off	Off
50	Audio Hardware	On	Off	Off
51	Audio Hardware	On	Off	Off
52	Audio Hardware	On	Off	Off
53	Audio Hardware	On	Off	On
54	Audio Hardware	On	Off	On
55	Audio Hardware	On	Off	On
56	Audio Hardware	On	Off	On
57	Audio Hardware	On	On	Off
58	Audio Hardware	On	On	Off
59	Audio Hardware	On	On	Off
60	Audio Hardware	On	On	Off
61	Audio Hardware	On	On	On
62	Audio Hardware	On	On	On
63	Audio Hardware	On	On	On
64	Audio Hardware	On	On	On

## 86 - FILES AND DIRECTORIES ARDOUR KNOWS ABOUT

### 156.1 Configuration Directory

Ardour stores configuration files in two places. The system configuration directory and the user configuration directory. The system configuration directory is used for stock configuration files at install time. The user configuration directory is used by Ardour to store configuration changes made in the GUI as well as being a place the user can add control surface device files, scripts etc.

Ardour tries to use standard places for these directories for the platform it is running on.

#### 156.1.1 Linux

The user configuration directory will be somewhere inside the user's home directory. The home directory on a linux system is normally `/home/$USER/`, but should also be returned by `$HOME` or `~`. A normal place to find this is `$HOME/.config/ardour*/` where `*` is the major version. However this can be set by the system with the `$XDG_CONFIG_HOME` environment variable to something else. If you cannot find `$HOME/.config/` on your system try `echo ${XDG_CONFIG_HOME}` to see if your distro is using something else. In any case Ardour appends the `ardour*` directory to the result where `*` is the major version number. For example, `ardour5` where the Ardour version is 5.6.

In Linux, all path names are lowercase and case-sensitive.

#### 156.1.2 macOS

The user configuration directory on macOS is `$HOME/Library/Preferences/Ardour*/` where `*` is the major version number. For example, `Ardour5` where the Ardour version is 5.6.

#### 156.1.3 Windows

Windows users are not expected to hand edit configuration files at all. It is expected configuration options are changed with some sort of GUI tool. For the most part all of Ardour's configuration is taken care of by the GUI in preferences. However, there are devices that may need a custom file and that would be in the users configuration directory.

Ardour asks the system for this directory and then appends `Ardour*` to the path where `*` is the major version number. For example, `Ardour5` where the Ardour version is 5.6. The official path would look like: `%localappdata%\Ardour5\` Windows expands `%localappdata%` to a real path.

An example of a configuration path in Window 10 would be: `C:\<User>\AppData\Local\Ardour5\` The user in the path would be the user's account name.

The above is only an example and may not even be true for all installations of Windows 10.

## 156.2 Plugins

Plugins will be installed in various places, some by standard and some by developer whim. Some are installed incorrectly by distro policy.

### 156.2.1 Linux

In linux there are 3 kinds of plugins Ardour can use. LADSPA, LV2 (LADSPA version 2) or lxvst (VSTs compiled as native linux binaries). While it is possible with some strange magic to run *Windows VSTs* on linux, their whereabouts would follow the Windows info below.

#### LADSPA

LADSPA plugins should be found in `/usr/lib/ladspa/`, `/usr/local/lib/ladspa/` or in a directory mentioned in your LADSPA\_PATH environment variable. The most common mistake made by distro packagers, is to use a path like `/usr/lib/$ARCH/ladspa/` and find that Ardour will not find that by default. The user can either add a link from this actual directory to the standard directory or add this path to LADSPA\_PATH.

#### LV2

LADSPA plugins should be found in `/usr/lib/lv2/`, `/usr/local/lib/lv2/` or in a directory mentioned in your LV2\_PATH environment variable. The most common mistake made by distro packagers, is to use a path like `/usr/lib/$ARCH/lv2/` and find that Ardour will not find that by default. The user can either add a link from this actual directory to the standard directory or add this path to LV2\_PATH.

#### Linux VST or lxvst

They are typically installed in `/usr/lib/lxvst`, `/usr/local/lib/lxvst` or a directory mentioned in your LXVST\_PATH environment variable. However, this is not a standard and the VST plugin developer may install the plugin just about anywhere. Therefore Ardour allows the user to set extra VST paths in the preferences GUI under Plugins>VST.

### 156.2.2 macOS

On the Mac, plugins are expected to be installed correctly Ardour uses the system tool to scan for AU style plugins and LV2s should be in the right place. LV2 should be in `$HOME/Library/Audio/Plug-Ins/LV2/` `/Library/Audio/Plug-Ins/LV2/` `/usr/local/lib/lv2/` `/usr/lib/lv2/` If an AU or LV2 plugin does not show up on a Mac it is probably a development fault with the plugin and the plugin will not work with anything. Ardour in Ardour 5.6 has support for native VST plugins. That is VST plugins built for OSX. I am not sure if these have a standard place to be, but as with other VSTs the search path can be edited at Plugins>VST.

### 156.2.3 Windows

The most common plugins on Windows are VSTs. However, LADSPA and LV2 plugins are available for windows as well. In fact Ardour's built in plugins are LV2s. The biggest advantage of LV2 plugins is that they are the most likely to be cross platform and therefore allow the same Ardour project to be worked on in Windows, OSX and Linux.

#### VST

As with other platforms, VSTs on Windows do not have a standard place to reside. Ardour Preferences>Plugins>VST allows setting the VST path from the GUI.

#### LV2

The LV2 standard for Windows is %APPDATA%/LV2/ (On Windows 10: C:\<User>\AppData\Roaming\LV2\)%COMMONPROGRAMFILES%/LV2/ (On Windows 10: C:\Program Files\LV2\).

## 156.3 Project Directory

Ardour places a project directory where the user tells it to. This directory is chosen when creating a project. In most cases the user does not need to know about the files inside of the project directory. However there are a few sub-directories worth noting.

### 156.3.1 export

This is the sub-directory where exported files end up.



---

**CHAPTER  
SEVEN**

---

## **87 - MIDI NOTES REFERENCE**

The table below lists the MIDI notes, numbers and frequency. Ardour uses the *middle C = C4 (note 60)* convention, meaning that the first (lowest) octave is -1.

Frequency calculations are based on  $A_4 = 440 \text{ Hz}$ .

MIDI number	MIDI (english) Note Name	German Note Name	Neo-Latin Note Name	Octave	Frequency (Hz) <i>Rough</i>
0	C	C	Do	-1	8.176
1	C#/D	C#/D	Do#/Re	-1	8.662
2	D	D	Re	-1	9.177
3	D#/E	D#/E	Re#/Mi	-1	9.723
4	E	E	Mi	-1	10.301
5	F	F	Fa	-1	10.913
6	F#/G	F#/G	Fa#/Sol	-1	11.562
7	G	G	Sol	-1	12.250
8	G#/A	G#/A	Sol#/La	-1	12.978
9	A	A	La	-1	13.750
10	A#/B	A#/B	La#/Si	-1	14.568
11	B	H	Si	-1	15.434
12	C	C	Do	0	16.352
13	C#/D	C#/D	Do#/Re	0	17.324
14	D	D	Re	0	18.354
15	D#/E	D#/E	Re#/Mi	0	19.445
16	E	E	Mi	0	20.602
17	F	F	Fa	0	21.827
18	F#/G	F#/G	Fa#/Sol	0	23.125
19	G	G	Sol	0	24.500
20	G#/A	G#/A	Sol#/La	0	25.957
21	A	A	La	0	27.500
22	A#/B	A#/B	La#/Si	0	29.135
23	B	H	Si	0	30.868
24	C	C	Do	1	32.703
25	C#/D	C#/D	Do#/Re	1	34.648
26	D	D	Re	1	36.708
27	D#/E	D#/E	Re#/Mi	1	38.891
28	E	E	Mi	1	41.203
29	F	F	Fa	1	43.654
30	F#/G	F#/G	Fa#/Sol	1	46.249
31	G	G	Sol	1	48.999
32	G#/A	G#/A	Sol#/La	1	51.913

Continue

Table 1 – continued from previous page

33	A	A	La	1	55.000
34	A#/B	A#/B	La#/Si	1	58.270
35	B	H	Si	1	61.735
36	C	C	Do	2	65.406
37	C#/D	C#/D	Do#/Re	2	69.296
38	D	D	Re	2	73.416
39	D#/E	D#/E	Re#/Mi	2	77.782
40	E	E	Mi	2	82.407
41	F	F	Fa	2	87.307
42	F#/G	F#/G	Fa#/Sol	2	92.499
43	G	G	Sol	2	97.999
44	G#/A	G#/A	Sol#/La	2	103.826
45	A	A	La	2	110.000
46	A#/B	A#/B	La#/Si	2	116.541
47	B	H	Si	2	123.471
48	C	C	Do	3	130.813
49	C#/D	C#/D	Do#/Re	3	138.591
50	D	D	Re	3	146.832
51	D#/E	D#/E	Re#/Mi	3	155.563
52	E	E	Mi	3	164.814
53	F	F	Fa	3	174.614
54	F#/G	F#/G	Fa#/Sol	3	184.997
55	G	G	Sol	3	195.998
56	G#/A	G#/A	Sol#/La	3	207.652
57	A	A	La	3	220.000
58	A#/B	A#/B	La#/Si	3	233.082
59	B	H	Si	3	246.942
60	C	C	Do	4	261.626
61	C#/D	C#/D	Do#/Re	4	277.183
62	D	D	Re	4	293.665
63	D#/E	D#/E	Re#/Mi	4	311.127
64	E	E	Mi	4	329.628
65	F	F	Fa	4	349.228
66	F#/G	F#/G	Fa#/Sol	4	369.994
67	G	G	Sol	4	391.995
68	G#/A	G#/A	Sol#/La	4	415.305
69	A	A	La	4	440.000
70	A#/B	A#/B	La#/Si	4	466.164
71	B	H	Si	4	493.883
72	C	C	Do	5	523.251
73	C#/D	C#/D	Do#/Re	5	554.365
74	D	D	Re	5	587.330
75	D#/E	D#/E	Re#/Mi	5	622.254
76	E	E	Mi	5	659.255
77	F	F	Fa	5	698.456
78	F#/G	F#/G	Fa#/Sol	5	739.989
79	G	G	Sol	5	783.991
80	G#/A	G#/A	Sol#/La	5	830.609
81	A	A	La	5	880.000
82	A#/B	A#/B	La#/Si	5	932.328

Continue

Table 1 – continued from previous page

83	B	H	Si	5	987.767
84	C	C	Do	6	1 046.502
85	C#/D	C#/D	Do#/Re	6	1 108.731
86	D	D	Re	6	1 174.659
87	D#/E	D#/E	Re#/Mi	6	1 244.508
88	E	E	Mi	6	1 318.510
89	F	F	Fa	6	1 396.913
90	F#/G	F#/G	Fa#/Sol	6	1 479.978
91	G	G	Sol	6	1 567.982
92	G#/A	G#/A	Sol#/La	6	1 661.219
93	A	A	La	6	1 760.000
94	A#/B	A#/B	La#/Si	6	1 864.655
95	B	H	Si	6	1 975.533
96	C	C	Do	7	2 093.005
97	C#/D	C#/D	Do#/Re	7	2 217.461
98	D	D	Re	7	2 349.318
99	D#/E	D#/E	Re#/Mi	7	2 489.016
100	E	E	Mi	7	2 637.020
101	F	F	Fa	7	2 793.826
102	F#/G	F#/G	Fa#/Sol	7	2 959.955
103	G	G	Sol	7	3 135.963
104	G#/A	G#/A	Sol#/La	7	3 322.438
105	A	A	La	7	3 520.000
106	A#/B	A#/B	La#/Si	7	3 729.310
107	B	H	Si	7	3 951.066
108	C	C	Do	8	4 186.009
109	C#/D	C#/D	Do#/Re	8	4 434.922
110	D	D	Re	8	4 698.636
111	D#/E	D#/E	Re#/Mi	8	4 978.032
112	E	E	Mi	8	5 274.041
113	F	F	Fa	8	5 587.652
114	F#/G	F#/G	Fa#/Sol	8	5 919.911
115	G	G	Sol	8	6 271.927
116	G#/A	G#/A	Sol#/La	8	6 644.875
117	A	A	La	8	7 040.000
118	A#/B	A#/B	La#/Si	8	7 458.620
119	B	H	Si	8	7 902.133
120	C	C	Do	9	8 372.018
121	C#/D	C#/D	Do#/Re	9	8 869.844
122	D	D	Re	9	9 397.273
123	D#/E	D#/E	Re#/Mi	9	9 956.063
124	E	E	Mi	9	10 548.082
125	F	F	Fa	9	11 175.303
126	F#/G	F#/G	Fa#/Sol	9	11 839.822
127	G	G	Sol	9	12 543.854