



Duplex : MIDI/OSC controller framework for Renoise

This manual is valid for Duplex v0.98

Table of contents

[Chapter 1: Introduction](#)

[What is Duplex?](#)

[Intended audience](#)

[Development roadmap](#)

[Chapter 2: The front-end](#)

[The tool menu](#)

[The browser](#)

[The options dialog](#)

[Chapter 3: Bundled applications](#)

[Shared feature: paged navigation](#)

[Applications A-Z](#)

[Chapter 4: Device support](#)

[Using your device with Duplex](#)

[Supporting a new device](#)

[Troubleshooting devices](#)

[Chapter 5: Working with control-maps](#)

[A basic example](#)

[Controlling the layout : columns and rows](#)

[How to split a control-map into separate groups](#)

[Hacking the control-map, a.k.a. "dubious techniques"](#)

[Control-map reference: Group attributes](#)

[Control-map reference: Parameter attributes](#)

[Chapter 6: The device configuration](#)

[Dissecting an actual device-configuration](#)

[Device-config: application mappings](#)

[Device-config: application options](#)

[Device-config: application palette](#)

[Appendix A: Common questions and answers](#)

[Appendix B: Technical info](#)

[Appendix C: Links and resources](#)

Chapter 1: Introduction

What is Duplex?

Duplex is a collection of scripts that control various aspects of Renoise with plug-and-play support for a wide range of hardware. The applications range from controlling basic aspects of the Renoise application (like Mixer, DSP chains, Pattern Matrix), to more adventurous applications like sequencers and performance tools.

What makes Duplex special is the strong focus on bi-directional communication. In practice, this will ensure that your controller is always in sync with Renoise, no matter what you do - something which is not possible with the native MIDI-mapping capabilities of Renoise.

As a script author, using the Duplex framework can assist in creating programs that are adaptable for a wide range of hardware. It greatly simplifies tasks like creating sliders from buttons, or setting the LED lights on your controller using standard RGB values, and in addition it will automatically perform background tasks such as optimizing outgoing communication (to make controllers more responsive), etc.

Duplex is a standard Renoise tool (xrnx), can be downloaded from tools.renoise.com, and installed by dragging the file onto the Renoise application. Since Duplex is written entirely using the Renoise API, anyone can extend and modify it. The official version, however, is maintained by the Renoise team.

Intended audience

This manual is explaining Duplex in broad terms to begin with, gradually becoming more technical. To make things more organized, the available information is divided into “experience levels”



This manual is primarily aimed at users who are new with Duplex (beginners), as well as advanced users who want to modify or customize their setups (experts). But no matter the experience level, the manual should help to explain the basics, and point you in the right direction. Keep in mind, though, that nothing comes close to the Renoise forum when dealing with actual support and helpful advice.

Development roadmap

Historically, Duplex was first introduced along with scripting in Renoise v2.6, and have since been updated with numerous new features, applications and devices. For the full changelog, see [here](#)

Before you spend a lot of time figuring out if feature XYZ is missing or not, here is a short list of things that the Duplex framework isn't currently capable of supporting, but is planned for future inclusion:

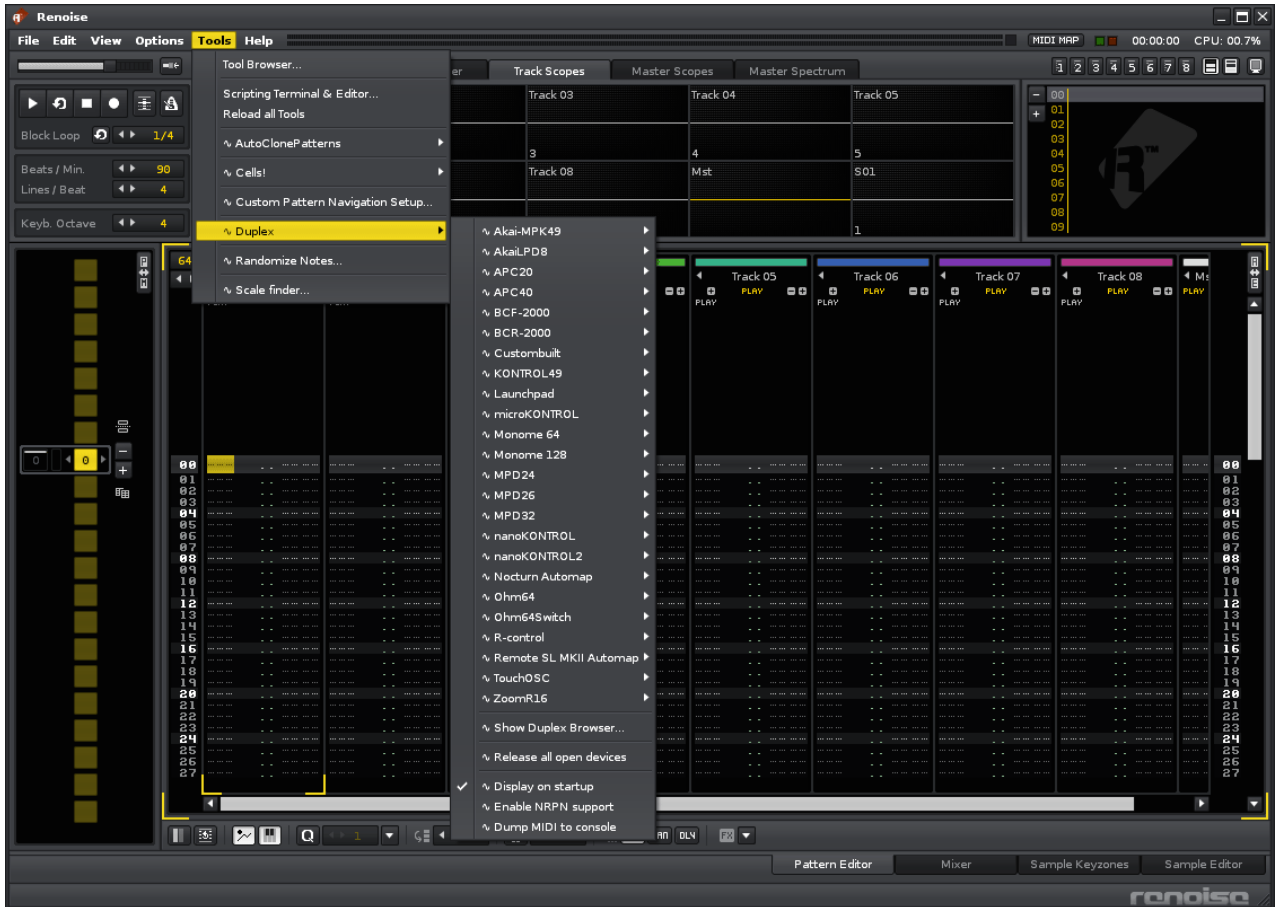
Description	Planned	Renoise API support (v3)
Relative (endless) encoder support	Yes	Yes
Real-time note triggering	Yes	No
Extended MIDI range (>7 bit, via sysex)	Yes	Yes
LCD/Text display updates on remote device	Yes	Yes

Chapter 2: The front-end

For the most part, Duplex is self-documenting - mouse over any part of the interface to get help and additional information about what each buttons/control does. The following is a quick walk-through of the most important user-interface elements, and how they work:

The tool menu

When Duplex is first installed, a sub-menu labelled Duplex will appear in the tools menu



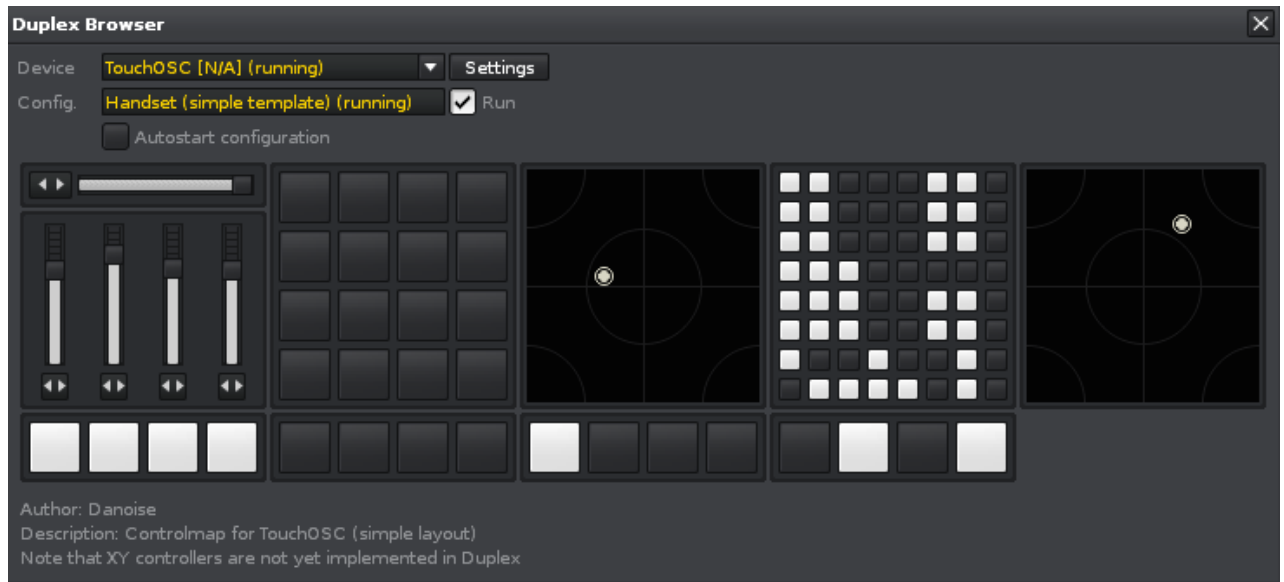
Show browser	Display the Duplex browser, the primary UI for displaying and navigating devices
Display on startup	Enable this to show the Duplex browser on startup. This will only have an effect when one or more device configurations have been selected to "autostart"
Release all...	Close all running devices & configurations that are currently running
Dump MIDI...	This is useful for debugging faulty MIDI devices

Furthermore, the menu provides quick access to all of Duplex' presets (a.k.a. device configurations). Each one comes with a descriptive name such as "Launchpad Mixer + Matrix", or "Simple TouchOSC template", categorized by the device name. Each of these configurations represent a mix of application that we have found to be useful, sometimes selected by popular vote. Select any one of them, and the Duplex Browser dialog will appear.

The browser

From the Duplex browser, you can turn device configurations on/off, and switch between any device & it's configurations. The window also contain a unique feature in Duplex, the virtual representation of the hardware using native Renoise UI components (a.k.a. the virtual control surface).

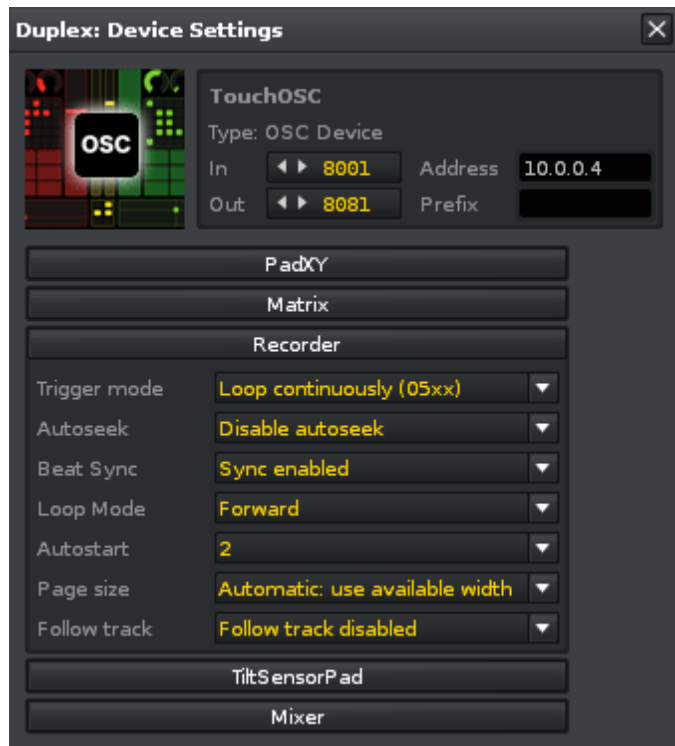
The browser is a fully multi-tasking environment - each device configuration can define multiple applications, and multiple devices can be opened simultaneously. When a device is presently active and running, it will contain an additional "(running)" suffix appended to it's name.



Run	Specify if the application is presently active / responding to input
Autostart	Make the selected application start when Renoise is first launched
Options	Bring up the device-config options

The options dialog

With each device configuration you have an options dialog, which contain the device settings, as well as application-specific options. Note that these settings (including the device settings) are unique per device configuration, so you can have the Mixer application configured differently in two applications. The options are persistent (remembered between sessions) and applied in real-time, so any change you make should immediately be reflected on the device.



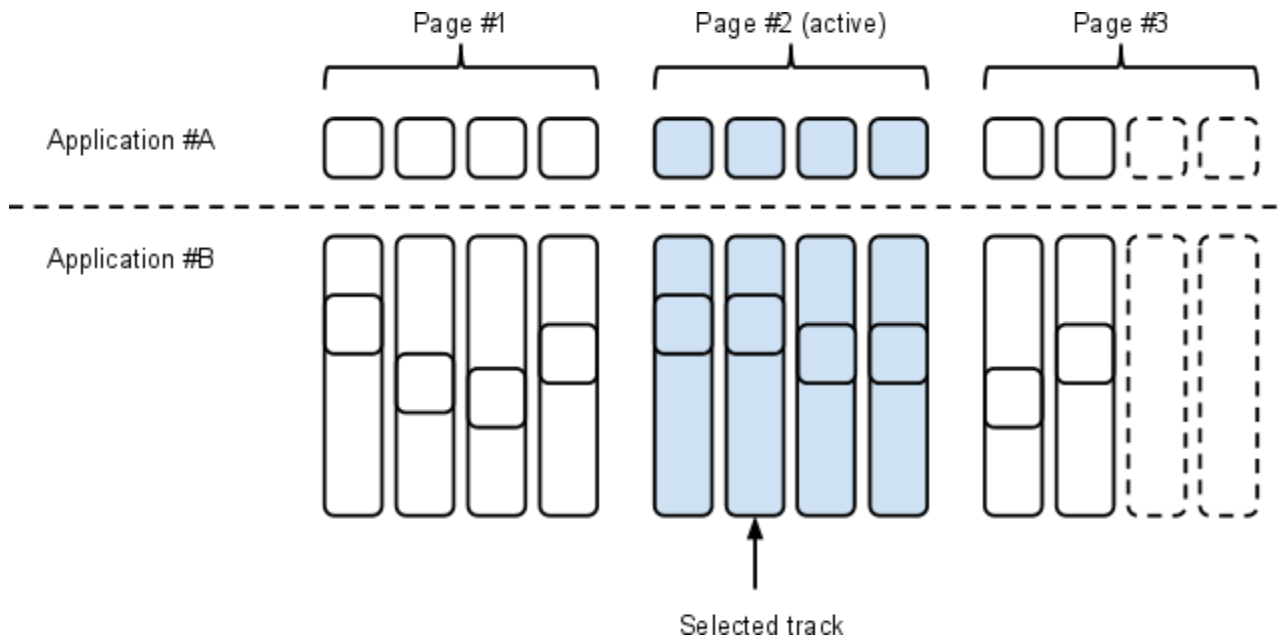
Chapter 3: Bundled applications

In Duplex, an “application” is a piece of code which perform a specific task and can be controlled via your device. It does not come with it’s own user-interface, *your controller is the interface*.

Shared feature: paged navigation

Most of the applications have been designed to integrate well with each other. One of the techniques involved is “paged navigation”, which basically means that the application is actively listening to the current track/pattern/device, and acting upon this information. In most cases, it’s a question of dividing the available number of tracks, patterns etc. into equally sized segments (a.k.a. “pages”), and then to update the display when the current position has changed.

The following illustration depicts how 10 tracks is automatically divided into three groups when the page size is set to 4 (obviously, both application #A and #B need to have the same page-size before this will work).



Paged navigation can be made to work without following/modifying the active selection in Renoise, too. This will reduce the paged navigation to a simple “previous/next page” style navigation, which might be useful if you are planning on using the controller while another person is using Renoise (as two people fighting over the active selection in Renoise could lead to a lot of confusion). But unfortunately this freedom comes with a price, as you can’t sync multiple applications to the same position then (as there is no position to track).

Any application that support paged navigation should always come with these two options:

- Follow_pos (or follow_track, follow_pattern): actively track the current position within Renoise.
- Page size: specify the size of each page (when pressing next/previous)

Applications A-Z

This section contain a brief description of each application, giving you an idea about it's purpose. The information is not exhaustive, though - if you are looking for detailed instructions on how to use them, which mappings they expose, please check the links - the actual source code of the application will contain all the necessary info for setting up customized mappings, etc.

Name	Description	Links
Matrix	The Matrix application is aimed at controlling the pattern matrix as well as the pattern sequence. It allows you to toggle the mute state of matrix slots, and trigger & loop patterns in the sequence in a number of ways. Paged navigation is fully supported.	Source
Mixer	The Mixer application is aimed at controlling the Renoise mixer. It will automatically attach itself to tracks, and expose the volume, panning, mute state etc. of each track using a paged navigation mechanism.	Source
Effect	The Effect application will enable control of every parameter in any DSP chain. The application will follow the currently selected device, and display all of it's parameters using a paged navigation mechanism. Various criteria can be established to include/exclude specific parameters: automated only (track), visible in mixer (track), all parameters (device).	Source
StepSequencer	The StepSequencer will enable you to display and edit pattern notes on an external controller, with realtime control of placement, note volume and pitch. A basic copy/paste feature has been implemented as well.	Forum Source
Transport	The Transport app will provide access to the basic transport features in Renoise: start/stop (either separate or combined), next/previous pattern, edit-mode, loop-block and metronome switches	Source
Navigator	The Navigator application will allow you to control the playback position within the current pattern, including block-loop size and position.	Source
TrackSelector	The TrackSelector will allow you traverse the song's tracks using paged navigation. In addition, you will have mappings for going to the next/previous track or page, a dial/slider which can be used for scrolling through tracks, and three explicit mappings for taking you to the (first) sequencer track, master track or send track	Source
Metronome	The Metronome is a sample application, designed to teach the basics of writing a bi-directional application	Source

Recorder	The Recorder is a special application that you can use to record any signal that you feed into Renoise, be that your voice, a guitar etc. It's designed as a live performance tool, which means that operation is designed to be really simple.;press a button to select the track and bring up the recording dialog, press again to start recording. Once the recording is done, it's (optionally) looped/synced to the beat, and you're then instantly able to switch among this, and all the other recordings you've made. The Recorder will even allow samples with different lengths to loop continuously, allowing for poly-rhythms, and maintain/restore recordings between Renoise sessions. Supports paged navigation.	Manual Source
NotesOnWheels	Notes On Wheels (N.O.W.) is a hybrid step-sequencer/phrase arpeggiator. It allows you to create a sequence between 1 and 12 steps, and control individual aspects of each step, such as the pitch, velocity, sample offset etc. A special feature is it's ability to learn from it's own output - this makes it possible to 'pick up' any sequence which has previously been written to a pattern. Input is very flexible, as you can use any controller with a few buttons and - most importantly - dials/faders to control parameters. Additional note input & transpose is supported by using either the PC keyboard or an external MIDI keyboard.	Forum Manual Source
SwitchConfiguration	The SwitchConfiguration application will allow you to switch between different configurations from your device (goto next / previous configuration). Note that you need to add the SwitchConfiguration to every configuration, or you would have a "one-way street"	Source
GridPie	Grid Pie is a performance tool which let the user combine different parts of a linear song, non-linearly, in real time, using a special looping pattern as a live drafting area. It does so by taking over the Pattern Matrix. This application supports paged navigation. Note that forum is linking to the original Grid Pie tool, with the Duplex port of the script arriving a bit later.	Forum Source
XYPad	Dedicated control of XYPads with locking support (remember mapping across Renoise sessions). Supports MIDI knobs, touchpads and multiple-value OSC messages (accelerometers, etc) as input.	Forum Source
Rotate	Rotate is able to rotate note- and (optionally) automation data for a single track or the whole pattern, by an adjustable amount.	Source

Chapter 4: Device support

Using your device with Duplex

Before anything else, check if your device is [already supported](#). Duplex has a growing list of supported devices - 24 at the time of writing - each one with at least one useful configuration. If your device is not supported, you probably have an idea about what the device might be well-suited for: for example, a device with plenty of physical knobs would be particularly good at controlling effects and volume levels, while a grid controller would be excellent for controlling the matrix, or as some sort of sequencer. It's a good idea to have realistic expectations about what your device is capable of.

Supporting a new device

For every new device, some research is needed. Depending on your experience with these matters, you might want to do it yourself, or you would like someone to help out. No matter what, the following will describe a typical device setup-process for a standard MIDI/OSC device.

First off, it's important to gain as much knowledge about the device as possible. Most manufacturers provide a detailed MIDI implementation chart or OSC protocol specifications. This is needed in order to connect and receive data from the device, at least the part which tell us the expected, acceptable values we can transmit to the device - as we can quite easily figure out what it is transmitting in our direction, by using a MIDI utility like MIDI-OX. The Duplex [list of controllers](#) contain links to typical examples of such specifications.

Once we have obtained this fundamental knowledge, the next part is about writing a proper control-map - for that, we need to determine the type of input: what *kind* of buttons does the device come with? (assuming of course, that the device has any). Again, let's consider the LED button which toggle on and off when pressed - it would be easy to write label this a toggle-button, but the reality is that Duplex can treat it as any type of button that it wishes to - perhaps the button will start to blink once it's pressed (as is the case with the Recorder application), or it might momentarily light up, and then turn dark. This is the power of bi-directional communication - the application is using the device in whichever way that seem appropriate. Therefore, it's important that you establish the type of button by connection the device *without running any music software whatsoever*. We want to know about it's "bare-bone" behaviour, so we can label the parameter in the control-map with one of the available input types ([see @type in the next chapter about Control-maps](#)).

To summarize: there is no "single approach" that work with every device in existence. Supporting a new device is, or can be, a complex task. It might even require specific initialization code, in which case Duplex would need a dedicated device class, a "driver" of sorts. And that particular subject is outside the scope of this manual, but don't be afraid to ask. Chances are, that someone on the Renoise forum can help.

Troubleshooting devices

As a general rule, whenever possible, operation should be plug-and-play. The following is a checklist you might want to go through, before creating a topic on the Renoise forum:

1. For MIDI devices, check that the right input and output ports are selected in the options dialog. For OSC devices, ensure that network communication is working, and not blocked by a firewall.
2. Open one of the device configurations, and read the comments in the lower part of the virtual control surface. Often, this might reveal details about special presets or editor files that are required in order to get the device working.
3. Enable the "MIDI dump" feature in the Duplex menu. The console should now list any communication between Renoise and the device. This is useful for debugging, and might reveal any problem in the setup.

Chapter 5: Working with control-maps

The control-map is an XML file that describe a device in terms of it's parameters and the physical layout. Duplex needs a control-map before it can do anything with a device, so they're good to know a little about if you're planning to make some modifications of your own.

A basic example

```
<Device>
  <Name>Device name</Name>
  <Author>Name of author</Author>
  <Description>Here goes a description...</Description>
  <Parameters>
    <Group name="My Group">
      <Param minimum="0" maximum="64" type="button" name="First" value="CC#10" />
      <Param minimum="0" maximum="127" type="button" name="Second" value="C-4" />
      <Param minimum="0" maximum="127" type="fader" name="Third" value="C-4|10" />
    </Group>
  </Parameters>
</Device>
```

There's a lot to chew on here, so let's start from the top: the `<Device>` is the root node of everything, and contains a section with `<Name>`, information about the `<Author>`, and a small `<Description>`. When you open a device using the Duplex browser, this is the information that is displayed beneath the control surface itself.

The `<Parameters>` node is where the actual device is being described. Inside we have a `<Group>` node, which is the container for our `<Param>` nodes. We always have to group our parameters like this, as we refer to individual parameters by their group name, and the position (index) they have inside the group.

There's a couple of important things to consider here: First one is, that you can't put `<Param>` nodes anywhere else but inside a `<Group>` node. Second one is that we always need to assign a name to the `<Group>` (we don't have to do that for the parameters, but it's generally a good idea).

Controlling the layout : columns and rows

If we want to control the layout of our control-map, we have the structural tags `<Column>` and `<Row>`. They're very simple to use, as they have no attributes - you simply put your content inside a `<Column>` or `<Row>` node, including other `<Column>` and `<Row>` nodes. Just remember to close any tags you create, to create "well-formed" XML (note that Duplex will validate the control-map each time a device preset is loaded, and tell you if you forgot to close a node, or is missing an attribute. Hopefully this will help you to locate and fix any errors that might occur)

How to split a control-map into separate groups

When you want to customize your device configuration, you might have to split groups within the control-map. As an example of this, see the Launchpad : Matrix + Mixer + Transport configuration. It utilizes a split control-map to fit two applications into the main grid space.

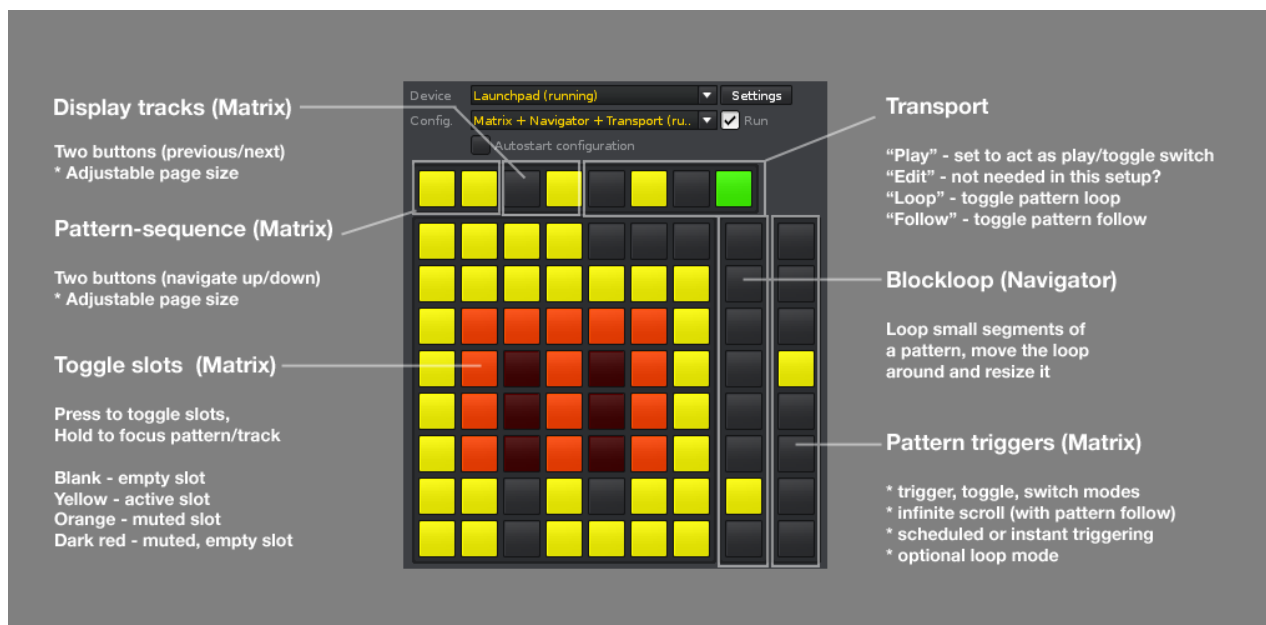
Basically, you need to open the control-map using an external editor and re-group <param> nodes into the desired layout. For this, you can use the <Column> and <Row> elements. Again, see the Launchpad control-maps, as they are a good example of how to split a large group into separate elements.

It might take a little practice to get it right, but since the control-map will be validated on each load, you should get some useful feedback from Duplex if something is broken.

Hacking the control-map, a.k.a. “dubious techniques”

Once you have a basic understanding of the control-map syntax, you might even want to get creative with it. Since the purpose of the control-map is to describe the physical layout of the controller, it’s possible to trick Duplex into thinking that your controller has a different layout than it really has.

For example, take this standard Launchpad layout running the Matrix application:



By changing the number of columns for the main “Grid” from 8 to 32, we get a completely different layout. Now, we are able to toggle 32 tracks in two patterns (but it doesn’t really look like a Launchpad anymore).



Control-map reference: Group attributes

@orientation

Specify a vertical or horizontal orientation using the @orientation attribute. This will cause the contents (the <Param> nodes inside the group) to orient themselves - either going from left-to-right (horizontal) or from top-to-bottom (vertical).

@columns

If the device has a lot of controls organized in a grid, define a columns attribute. This will "wrap" contents inside a group, creating the number of columns defined in this attribute

@colorspace

Some devices have multi-colored LEDs that are able to display a range of colors. The colorspace attribute is a simple way of defining what colors are accessible, and will also quantize the colors displayed in the virtual control surface.

Here is a Launchpad color-space - 4 levels of red and green:

```
colorspace = "4,4,0"
```

Here we have another device, the monochromatic monome:

```
colorspace = "1,1,1"
```

Extra, device-specific code is needed for interpreting the colors and converting them into a value that the device can understand. To learn more about color-spaces, please check the [Duplex API documentation](#)

Control-map reference: Parameter attributes

@name

You can assign any name you like to a parameter, it's only used to make the control-map easier to read, and not a required attribute as such.

@minimum, @maximum

Duplex will automatically use the minimum and maximum values for the switched-on/turned-off state, while something like a fader or dial can be anywhere in between.

Most (but not all) MIDI devices are limited to a 7-bit range for their communication. So when you turn a knob to control a filter, the knob will most likely output a value between 0-127 (7 bit). But sometimes, a device will expect a different value to work - for example, you could take a look on the first parameter in the example, and you'll see that it's set to "64" as maximum. This is what the Ohm64 will expect - the exact values depend on the device, so it's probably a good idea to look for a MIDI implementation chart if you are in doubt as to the correct and expected values.

@value

Another essential attribute is the value. Whenever we receive a message from a device, the value is the first thing we look at. For MIDI devices, it can tell us the "kind" of message we are dealing with, thanks to the special syntax. OSC messages are more generic in nature, and in most cases easier to read than MIDI messages.

Correction: the previous version of this manual accidentally left out the "Ch" for identifying channels!

Example	Message type	Matched because
CC#2	MIDI Control-change	Starts with "CC"
CC#7 Ch6	MIDI Control-change + channel	Starts with "CC"

C-4	MIDI Note	Has "-" or "#" as second character
G#5 Ch4	MIDI Note + channel	Has "-" or "#" as second character
C--3	MIDI Note (negative octave)	Has "-" or "#" as second character
PB	MIDI PitchBend	Starts with "PB"
PB Ch3	MIDI PitchBend + channel	Starts with "PB"
/led 3 2 %i /xyz %f %f %f	OSC message*	Starts with a slash







* OSC messages can contain embedded string directives, "%i" means integer and "%f" a floating-point value

@action

The "action" attribute is only used with OSC devices, and only when the device expects to receive a differently formatted message than the one it sends (monome is an example of this). In such a case, the "action" attribute is the value which is transmitted to the device when the user does something, and the "value" is the message we expect to receive from the device.

@type

The "type" is an important attribute. It tells Duplex exactly what kind of control we are dealing with, and how to represent it in the virtual control surface. Note that the names "button", "togglebutton" etc. has nothing to do with the built-in UIComponent types.

Type	Looks like	Attributes	Description
button		@aspect	Standard bidirectional button which output a value on press & release, but does not change / control it's internal state. Examples are the buttons of a Launchpad or monome.
togglebutton		@aspect	Bidirectional button which output a value when pressed, and (might) toggle it's state internally. This type of control does not support release & hold events. Examples are buttons on the BCF/BCR controller, and Automap "normal/toggle" buttons
pushbutton		@aspect	Bidirectional button which will output values on press & release while controlling it's state internally. Some examples are Automap "momentary" buttons, or TouchOSC pushbuttons
xypad		@invert_y @invert_x	The XY pad is special, as it can be controlled with two "paired" values. See the TouchOSC XYPad application for an example of how
fader		@orientation	Manual fader
dial			Basic rotary encoder

@size

Every parameter accepts the `@size` attribute, which will determine the relative size of the parameter on the virtual control surface, with "2" being approximately twice of the default size. You can enter a fractional value like "1.33" to fine-adjust the size if you like.

@orientation

The orientation is only valid for the fader component, it's purpose should be self-explanatory.

@aspect

This attribute is only valid for buttons. Define the attribute to control the height of the control as a factor of it's width - for example, a value of "0.5" will make the height 50% of it's width.

@invert_x, invert_y

Specific for XYPad, allow you to invert the values that you are receiving on each axis (accepted values: "true" and "false")

@skip_echo

Basically, this means that we never send any updates to the control (when it's not bi-directional)

Chapter 6: The device configuration

A device configuration is where it all comes together: control-maps, device classes and applications. Any device configuration is a simple text file, which is parsed by the lua interpreter when Duplex is first loaded. So, while it's not actual code, it's still important that the syntax is 100% accurate, or you will receive an error.

Note: Satobox has posted an excellent write-up on how to work with the various files (control-maps, device configurations etc.). [Read his article here](#) (Renoise forum)

Dissecting an actual device-configuration

Let's start by studying this sample configuration:

```
duplex_configurations:insert {
  name = "Matrix + Navigator + Transport",
  pinned = true,
  device = {
    class_name = "Launchpad",
    display_name = "Launchpad",
    device_port_in = "Launchpad",
    device_port_out = "Launchpad",
    control_map = "Controllers/Launchpad/Controlmaps/Launchpad_Matrix.xml",
    thumbnail = "Controllers/Launchpad/Launchpad.bmp",
    protocol = device_MIDI_PROTOCOL,
  },
  applications = {
    Matrix = {
      mappings = {
        matrix = {
          group_name = "Grid",
        },
        triggers = {
          group_name = "Triggers",
        },
        sequence = {
          group_name = "Controls",
          index = 1,
        },
        track = {
          group_name = "Controls",
          index = 3,
        }
      },
      options = {
        sequence_mode = 2,
      }
    },
    Navigator = {
      mappings = {
        blockpos = {
          group_name = "Navigator",
        }
      }
    },
    Transport = {
      mappings = {
        edit_mode = {
          group_name = "Controls",
          index = 5,
        },
      },
    },
  },
}
```

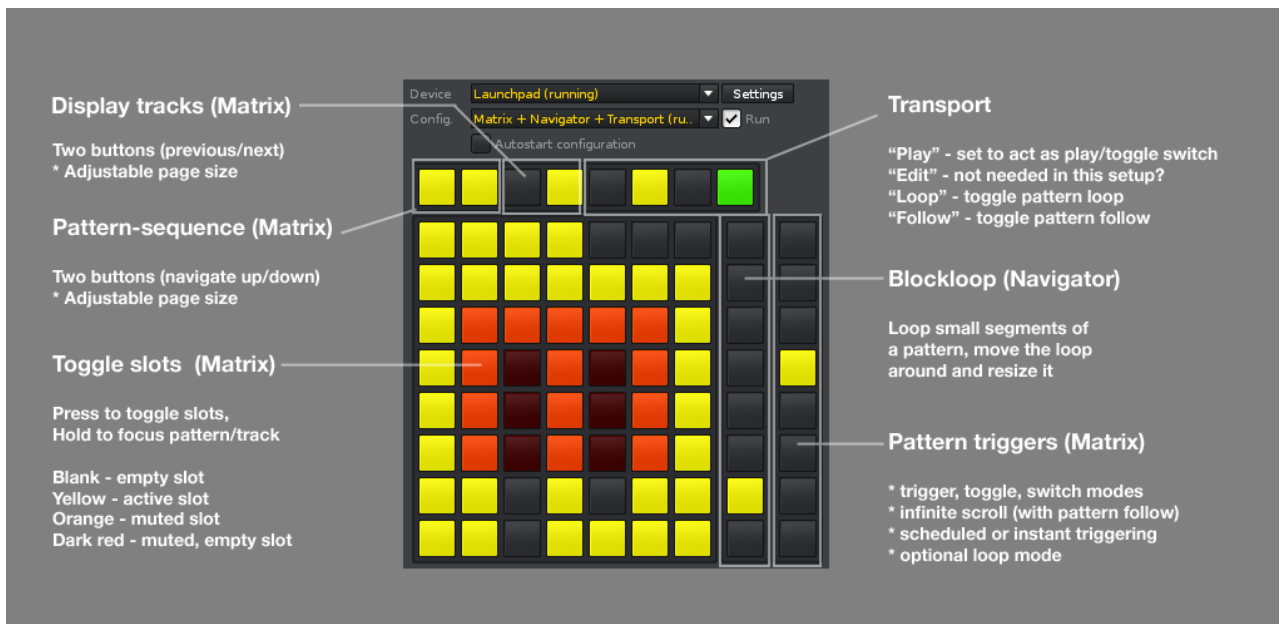


```

start_playback = {
  group_name = "Controls",
  index = 6,
},
loop_pattern = {
  group_name = "Controls",
  index = 7,
},
follow_player = {
  group_name = "Controls",
  index = 8,
},
},
options = {
  pattern_play = 3,
}
},
}
}

```

This is the result, as it will appear on the Launchpad



As you can see, we managed to squeeze quite a few features on to the device, while the syntax of the device configuration is pretty straight-forward. A couple of things should be mentioned though:

- The "name" of the application needs to be unique (only one configuration per device can have the same name).
- The "pinned" property means that the device configuration is visible in the tools menu. Set this property to "false" if you have a configuration you want to hide (it's still accessible through the browser)
- As for the device properties, they should largely be self-explanatory. There is a difference between OSC and MIDI devices, though, so if you are planning to support your own OSC device you probably want to take a copy of an existing OSC device and base your own configuration around that.

Device-config: application mappings

Each application comes with its own set of assignable mappings, roughly named after its purpose (like “master” for the Mixer master volume). This name isn't necessarily representative of something in Renoise. For example, if someone decided to make a Tetris game, you would probably have something like a “rotate_brick” mapping.

Basically, there's two different kinds of mappings: “indexed” and “greedy” mappings, indexed mappings being the ones with an explicitly defined index. But, what does this mean? Well, indexed mappings are relevant for specifying a particular position within a control-map group, whereas the “greedy” mapping will happily consume every available parameter inside the provided control-map group, assigning multiple parameters in one go. As an example of such a mapping, consider that the Matrix only has a single mapping for its entire grid, which may consist of hundreds of buttons. The indexed mapping, on the other hand, is more like a traditional mapping in the sense that you specify a position where you want a particular feature to be located.

Here we have a typical indexed mapping (a “play” button for the Transport app)

```
Transport = {
  mappings = {
    start_playback = {
      group_name = "Controls",
      index = 6,
    },
    [more mappings...]
  }
}
```

The mapping's name is “start_playback”, and it's referring to the 6th parameter in a control-map group named “Controls”. To learn what the various mappings are called, we need to open the application file itself. We then look for the first occurrence of the text “self.mappings” - probably located in the very beginning of the file. The section for the Transport would look like this:

```
self.mappings = {
  stop_playback = {
    description = "Stop playback",
  },
  start_playback = {
    description = "Start playback",
  },
  loop_pattern = {
    description = "Toggle pattern looping",
  },
  [more mappings...]
}
```

Some mappings also support the “orientation” property. This value is specified alongside the group_name and index, and is used to determine if the given mapping should be horizontal or vertical. It really is up to each individual application to decide if this property is supported, so check the application source (look for “self.mappings”) to see if this is the case.

Device-config: application options

Much like application mappings, each application come with a set of options to control its behaviour. The options are spelled out in the device configuration, and consist of a simple numeric value. For the most part, you will not need to deal with them, as you can much more easily access options using the options dialog in the Duplex front-end. But by specifying them as part of the device configuration, you are providing default values that are maintained even if you delete your preferences.

The following snippet is for launching the Matrix application with triggers scheduling patterns:

```
Matrix = {
  options = {
    switch_mode = 4,
  }
}
```

Now, "4" is not simply a number that came out of the blue sky, it's in fact referring to a particular option in the application. You need to study the application class itself to learn the exact values that you can set. You do it like this: Locate the application class, and look for the first occurrence of the text "self.options" - probably located in the very beginning of the file. The section for the Matrix would look like this:

```
self.options = {
  switch_mode = {
    label = "When switching",
    description = "What to do when switching from one pattern to another",
    items = {"Stop playback", "Switch to pattern", "Trigger pattern", "Schedule"},
    default = 2,
  },
  [more options...]
}
```

As you can see, the default value is "2", which means that the default behavior is to switch to a pattern as soon as the pattern trigger is pressed. By specifying "4" in the device configuration, you are telling the application to start with the 4th option (schedule the pattern) for that particular device. A little further down in the application, "option constants" reveal which choice each numeric values is referring to. In case of our Matrix trigger-option, the various choices would be:

```
self.SWITCH_MODE_STOP = 1
self.SWITCH_MODE_SWITCH = 2
self.SWITCH_MODE_TRIG = 3
self.SWITCH_MODE_SCHEDULE = 4
```

Device-config: application palette

Just like application mappings and options, you can define a custom palette for the application. The palette consist of two values: color and text. Color is obviously describing the RGB color (ranging from 0X00 to 0XFF), while the text is displayed on the virtual control surface when the target device doesn't support the use of color. You need to check the application source to see which values you can define.

Here we have the default palette for the XYPad application. To see this palette being customized, please take a look at the [Launchpad XYPad device-config](#)

```
self.palette = {
  foreground = {
    color={0xFF,0xFF,0x00},
    text="■",
  },
  background = {
    color={0x40,0x40,0x00},
    text="□",
  },
}
```

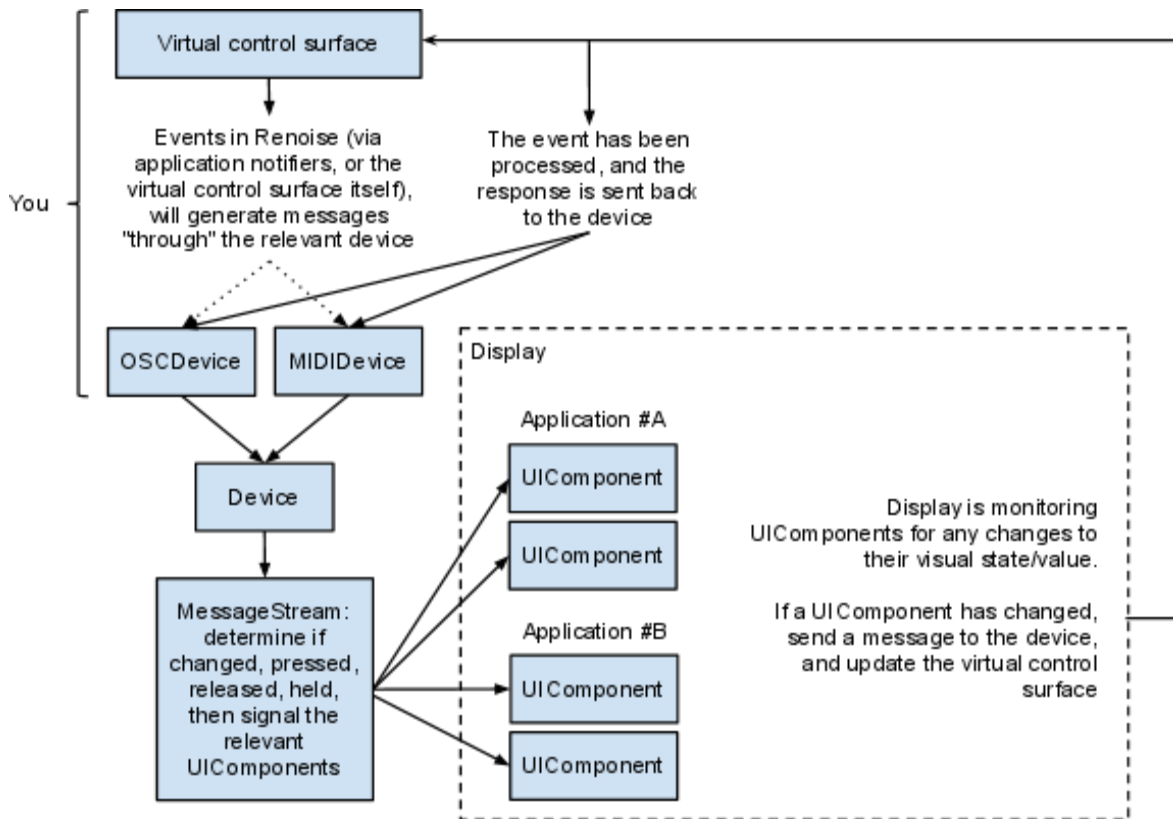
Appendix A: Common questions and answers

Question	Answer
Is my device supported?	First, install Duplex, and check to see if the menu list your device. If this is the case, then the answer is obviously “yes”. Otherwise, consult the “list of controllers” (with pending controllers)
My device is not working?	If you experience trouble getting the device to do anything, please read the chapter “troubleshooting your device”
Where do I locate the Duplex folder?	The Duplex folder is located in the Renoise Tools folder, named <i>com.renoise.Duplex.xrnx</i> . While in Renoise, you can open the Duplex folder like this: go to the tools menu, select the Tool Browser and locate Duplex. Right-click the tool, and there should be an option to reveal the folder (in explorer, finder etc.)
How do I locate an application class?	The applications are located in the Duplex folder, under Duplex/Applications
How do I locate a device configuration?	The device configuration is located in Duplex/Controllers/[Device Name]/[Device Name]/Configurations/ <i>Note: replace [Device Name] with the name of your device.</i>
How do locate a control-map?	The control-map is located in Duplex/Controllers/[Device Name]/[Device Name]/Controlmaps/ <i>Note: replace [Device Name] with the name of your device.</i> If you're unsure which control-map you're looking for, but you know which device configuration is using the control-map, open the device configuration in a text editor and look for the "control_map" property inside each configuration.
Can I reset my device configurations?	Duplex settings are stored in preferences.xml). If you want to go back to the default settings, delete this file when Renoise is not running.
My personal preferences (application settings) are gone?	When you install a new version of Duplex, your preferences might be deleted. To avoid this, take a backup of the file (preferences.xml) first. You can copy the file back to your folder once Renoise isn't running.
When I use my MIDI controller, a lot of “garbage” commands are entered into the patterns?	It's most likely because the controller is also configured in Renoise as an input device. Simply uncheck the device from Renoise preferences > MIDI input devices. The commands you're seeing are probably CC pattern commands (channel/parameter/value).
Is it possible to trigger notes using Duplex?	Depends on the definition of “triggering notes”. If you mean generating notes on-the-fly (writing them to the pattern editor), the answer would be yes - but you need some application that does this for you, e.g. one of the bundled sequencers. If you mean telling some instrument to play, without going through the pattern editor, the answer would be no. The current Renoise API does not include such "realtime methods".

<p>I have two Launchpads. Can I use them side-by-side ? Is it possible to attach multiple devices of the same kind?</p>	<p>Yes. It's possible to copy an existing device configuration, and make it appear under a different name. In this way, you can run a similar setup on multiple devices that are connected to different ports. Open the relevant device configuration and make a copy of all the desired configuration entries. Then, change the "display_name" in the new configurations to a new name, like "Launchpad (2)". The next you start Renoise, the Duplex browser will contain the new device as a separate entry. Don't forget to assign the correct input/output ports to the new device as well.</p>
<p>Can I use the same device to control multiple instances of Renoise?</p>	<p>Using the the same device in two Renoise instances is tricky, but possible. You'd probably want to split the control-map into two sections, each one controlling it's own instance. See the section on how to split a control-map for more details.</p>
<p>How do I turn off all devices?</p>	<p>Open the Duplex browser and select "none" in the device list. This will release all currently active devices. Alternatively, use the Duplex menu item "release all devices"</p>
<p>When I close Renoise, my controller still have some lights turned on?</p>	<p>Duplex does not actively turn off lights when an application is shut down, but this should not matter as the device is completely re-initialized when starting up the next time.</p>
<p>Can I somehow see the communication between Renoise and my MIDI controller?</p>	<p>Yes, go to the Tools > Duplex menu and select "dump MIDI". This will output the MIDI messages in the console</p>
<p>Can I change the location of individual buttons and sliders? How can I add more applications to my device?</p>	<p>Yes, see chapter 5: Device configurations for extensive information on how to customize your controller</p>
<p>How to enable the scripting console in Renoise</p>	<p>The xrnx code repository has instructions for enabling the scripting console</p>

Appendix B: Technical info

This illustration depicts the signal flow in Duplex.



A new signal is generated either from the virtual control surface (when we interact with the "emulated device"), or from the actual OSC/MIDI device (when we turn a physical knob, press a key etc.)

A signal leaves Duplex when the message has been processed by the Display (been through an application), and it has been determined that the control has changed.

Appendix C: Links and resources

1. The [Duplex beta-version thread](#) (Renoise forum) always contain the latest version of this tool
2. [Duplex list of controllers](#), a spreadsheet containing all pending & supported devices
3. Satobox has posted an excellent write-up on how to work with the various files (control-maps, device configurations etc.). [Read his article here](#) (Renoise forum)
4. [How to create a new Duplex application](#), first of a series of in-depth articles (Renoise forum)
5. Hackers might want to study the [Duplex API docs](#) to learn about the available classes, their properties and methods