TouchOSC

Next generation modular control surface



Introduction

TouchOSC is a modular control surface builder for designing and constructing custom controllers that can be used on a multitude of operating systems and devices.



TouchOSC can be used on touch screen mobile devices as well as desktop operating systems using traditional computer interfaces.

TouchOSC can communicate with other software and hardware using the MIDI and Open Sound Control protocols in a variety of ways and via many different types of wired and wireless connections, simultaneously.

All versions of TouchOSC include the powerful, integrated editor, to customize every detail of your controller setup directly on your mobile device. Controllers can be created from scratch or by using one of the many available templates using the easy scripting features and the supplied documentation and communication support forums.

Any control surface document you create can be freely exchanged between all versions of the application, and documents can be shared with other users via email, Dropbox, or any other service that can handle binary files.

If you want to send MIDI messages from your computer to your device, you'll need the Bridge application running on the PC/Mac OS X/ or Windows touch-based computer to send and receive data over the network.

Please note that a manual for an application as complex as TouchOSC will always be a work in progress, much like the application itself. We've done our best to provide comprehensive documentation of the application's operation, all which is free and up to date.

So please do check back from time to time for updates and additions to the material presented here.

Hexler Deep Interactive - 2021

Next Page: Getting Started

Getting Started

No time time to read manuals, right?

To get you started as quickly as possible, we've compiled brief tutorials for the minimum steps required to set up TouchOSC to work with a few of the most popular applications and hardware MIDI control surface applications out there.

These are not meant to be exhaustive instructions on how to set up TouchOSC, but more of a **quick start** to make you familiar with the basic concepts involved.

If you run into trouble with these, or want a more in-depth answer to a question, please see the extensive documentation and support forums on the TouchOSC website. Chances are that someone else has had the same problem, and we've likely already written a how-to or FAQ on the subject.

- MIDI
- OSC
- Ableton Live

Next Page: Getting Started - MIDI

Getting Started - MIDI

In this short tutorial, we will be setting up the sending of MIDI messages from TouchOSC to another application on the same computer.

We'll be using the TouchOSC Bridge utility application, but the TouchOSC Editor MIDI messages over the network, so this is a common use case to control software running on mobile touch screen devices.



- Protokol
- TouchOSC Bridge
- MIDI Monitor
- Send MIDI messages

Protokol ↗

We'll be using the Protokol application, as it has tools for testing OSC and MIDI connections and messages. This is the one control we use in Hexler for testing our applications.

Protokol is available for all the same platforms as TouchOSC and in a free download.



- Download and install any version of Protokol ↗
- Launch the application, switch to the **MIDI** tab page, and check the checkbox next to **Enabled**
- Protokol is now waiting for incoming MIDI messages on all available MIDI ports.

TouchOSC Bridge ↗

- Download and install TouchOSC Bridge ↗
- Launch the application in either tray or menu bar mode (or use Bonjour on macOS, in the task tray on Windows
- Protokol should now display the connection of the new MIDI port created by TouchOSC Bridge

TouchOSC ↗

Next we'll configure TouchOSC to send MIDI messages to TouchOSC Bridge and load one of the included example layouts.



- Switch to the **MIDI** tab page
- Make sure **Enabled** and **Connected** if and use the **Browse** button to select **<Bridge>** for both the **Send Port** and **Receive Port** menus

Now we'll load one of TouchOSC's included example layouts.



- Open the local of included example layouts, depending on your operating system.
- **For the desktop version**, find the menu at **Help > Examples**
- **For the mobile version**, find the button shown at the top of the yellow page on the right, and press the **Examples** button in the menu that appears.

With the document now loaded, and the connection configured per TouchOSC, the **control surface mode** is pressing the **play** button on the yellow page, in the entire the TRAL CONSOLE keyboard should now be an operative platforms.

Send MIDI messages ↗

With both the receiver and sender configured and ready to go, we'll send some MIDI messages.



Move a fader, press a button or interact with any other control on the control surface in TouchOSC and you should see the received MIDI messages being printed in Protokol.

Next Page: Getting Started - OSC

Getting Started - OSC

In this short tutorial, we will be setting up the sending of OSC messages from TouchOSC to another application.



- Protokol
- TouchOSC
- Send OSC messages

Protokol ↗

We'll be using the Protokol application, as it has tools for testing OSC and MIDI connections and messages. This is the one control we use in Hexler for testing our applications.

Protokol is available for all the same platforms as TouchOSC and in a free download.



- Download and install any version of Protokol ↗
- Launch the application, switch to the **OSC** tab page, and check the checkbox next to **Enabled**
- Protokol is now waiting for incoming OSC messages on all available OSC ports on the network.

TouchOSC ↗

Next we'll configure TouchOSC to send OSC messages to Protokol and load one of the included example layouts.



- Launch TouchOSC and open the **Connections** configuration by pressing the button with the cloud icon on the yellow toolbar
- Switch to the **OSC** tab page
- Enable the checkbox next to **Connection 1** and use the **Browse** button to select the name of the host or IP address where you're running Protokol
- Both **Host** and **Send Port** fields should now be filled in.

Now we'll load one of TouchOSC's included example layouts.

- Open the local of included example layouts, depending on your operating system.
- **For the desktop version**, find the menu at **Help > Examples**
- **For the mobile version**, find the button shown at the top of the yellow page on the right, and press the **Examples** button in the menu that appears.

With the document now loaded, and the connection configured per TouchOSC, the **control surface mode** is pressing the **play** button on the yellow page, in the entire the TRAL CONSOLE keyboard should now be an operative platforms.

Send OSC messages ↗

With both the receiver and sender configured and ready to go, we'll send some OSC messages.



Move a fader, press a button or interact with any other control on the control surface in TouchOSC and you should see the received OSC messages being printed in Protokol.

Getting Started - Ableton Live

In this short tutorial, we will work through mapping MIDI messages from TouchOSC to interface elements in Ableton Live.

You should complete the steps in the Getting Started - MIDI tutorial first, as we have a working MIDI connection between our host setting up Ableton Live.

Even though we are using Ableton Live in this examples, other applications can usually be configured in a similar fashion. Please refer to the application's manual, specifically the MIDI implementation chapter, to learn how you can map MIDI messages to application functions.

Open Ableton Live's **Preferences** dialog, select the pane labelled "MIDI / Sync" and enable the **Track** and **Remote** columns for the MIDI Ports labelled **TouchOSC Bridge**.

Now Ableton Live is receiving and sending MIDI messages through TouchOSC Bridge. To confirm this, move a slider and press a button in TouchOSC's **Basic** layout (which we loaded in the MIDI tutorial) and you should see the two messages appear in the **MIDI Monitor** window in Protokol.

Enter Ableton Live's MIDI Mapping mode by pressing the button labelled **MIDI**, that in the far right corner. This will highlight all elements in Live's UI that can be assigned to MIDI controllers with a blue overlay. Select one of Live's mixer channel faders, and then move a fader in TouchOSC's **Basic** layout. Live's MIDI mapping should now be set to be receptive to the kind of control.

Move a press any of the controls in TouchOSC's **Basic** layout and you will see the color add a new to the **MIDI Mappings** window and will add another mapping right above the assignment statement displaying above the MIDI message such as Channel and type.

Enter Ableton Live's MIDI Mapping mode by pressing the button labelled **MIDI**, that in the far right corner. This will highlight all elements in Live's UI that can be assigned to MIDI controllers with a blue overlay. Select one of Live's mixer channel faders, and then move a fader in TouchOSC's **Basic** layout. Live's MIDI mapping should now be set to be receptive to the kind of control.

Introduction
Getting Started
MIDI
OSC
Ableton Live
MIDI
Interfaces
Control
Properties
Layouts
OSC
Interfaces
Control
Layouts
Bonjour
Network
Connections
MIDI
Bridge
Network
Protocols
Protokol
Editor
Control Surface
Bridge
Network
Control Reference
Interfaces
Protokol - MIDI
Protokol - OSC
Bridge
Android USB MIDI
Raspberry Pi
Control Surface
Tablet Pro
Layouts
Scripting API
Introduction
Scripting Basics
Utility
Widgets
Tables and Arrays
Objects
Control Reference
Interfaces
Control
Properties
Properties & Values
Examples

The standard is best integrated for development between TouchOSC and Live. You can even sort Live's MIDI Mapping mode by clicking on the **MIDI** button in the top right corner again and start customizing your own TouchOSC. For more info on configuring MIDI and OSC please see the Editor and Bridge manuals.

Editor

TouchOSC comes with a powerful, integrated editor on all platforms, for creating and editing control surface documents.



TouchOSC can open touchosc layout files made for TouchOSC Mk1 and some aspects of the import can be configured in the preferences.

Documents will be saved in a new touchosc document format that is more compact, and in a **.tosc** compatible with the TouchOSC Mk1 application.

Multiple documents can be opened at the same time and all parts of the layout can be copied from one document to another, even between platforms.

Multiple instances of the application can be started if you want to edit documents on different screens or edit multiple controls in real time, using the standard Copy/Paste shortcuts (ctrl/cmd+c/v/x).

While there is minor differences between the desktop and mobile versions, most functions work identical across all supported platforms and each panel can be switched between Desktop and Mobile mode.

Editor - Interface

The main interface of TouchOSC can be one of two modes, **Editor** and **Control Surface** mode.

Switch to control surface mode by pressing the **play** button in the toolbar, and the **edit** button in the Editor to switch back to control surface. By starting anywhere from the Play, Edit or Control sections you can be used to toggle between the modes.

The position of the **strobe** button in the control surface can be configured in the preferences.

- Control Surface Editor
- Panel
- Toolbar
- Log View
- Message Mapping
- Font Viewer

Control Surface Editor

The main control surface editing area is where most editing actions take place.

Right clicking on an empty area of the document (or long pressing on mobile platforms) will bring up the **Create** context menu for adding new controls.



Controls can be selected either by direct touch/press on a control's 'frame' or using a lasso selection by pressing and dragging from an area close to the document. Shift control to select/deselect. Pressing on the edit and lasso context buttons will toggle the selection mode. Pressing on the lock button will lock the selection in place.

When dancing controls using the keyboard the order of the selected controls is the order you started pressing them. The following modifier keys can be held down to the keyboard to change behaviour:

- **SHIFT** - Resize with original aspect ratio locked
- **CONTROL** - Resize around the center point of the control

Right clicking on the selection (or long pressing on mobile platforms) will bring up the **Edit** context menu.



When a control is locked, it cannot be selected. Locked control can be unlocked using the document tree view.

Some types of controls use combinations for other controls, selecting **Edit container** from the **Edit** context menu will select the controls and descend into the controls for editing.

To start editing a container, use the additional **Edit container** menu item of the **Create** menu, or double click outside of the container's frame (handle) to descend.



The editor panel on the right will display a navigator view of the whole document, the document tree/hierarchy, properties/attributes and the script editor. Each of these panels can be detached, docked, minimized, maximized and rearranged in the control surface editor.

All sections of the editor panel can be collapsed and expanded using the section header bar.

Hit **enable panning**, the top of the panel will be used as a navigator view and controls that would usually be found in the toolbar will be displayed in the panel.



From left to right, the functions are as follows:

- Toggle the **editor panel**
- Toggle the **log view**
- Open the **document browser** dialog
- Open the **tidy assistant**
- **Undo** a change
- **Redo** a change
- Toggle **enable panning**
- Application **preferences** and **about** screen

Toolbar

The editor toolbar holds functions and menus for various control surface editor functions.

PLEASE NOTE: Accessing preferences and about screens have been moved to the control panel to display all toolbar buttons, in which case the toolbar can be omitted to be more touch, and slightly transparent, while controls can be displayed at either end of the screen.



From left to right, the functions are as follows:

- **Undo**
- **Redo**
- **Copy** selection
- **Paste** copied/cut contents
- **Duplicate** selection
- **Delete** selection
- **Lasso selection** - Add an adjustable container (solid lasso selection), this function becomes available to edit container controls.
- **Edit container** - When editing the contents of an editable container control type, this function becomes available to exit the container and return to the parent control and be edited by the parent.
- **Lock** and unlock controls
- **Down to 1**
- **Up to 1**
- **Bring view and colors** forward
- Switch to **control surface view**
- Open **preferences**
- **Editor Elements** and information

Log View

The log view provides a way for various sections of the application to print information about the internal state. This can be helpful when troubleshooting and debugging problems or for confirming that a process or action is working as expected.



The log view will display:

- MIDI/OSC Received and MIDI and OSC messages
- UDP/IP, Bonjour transmitted using zeroconf messages
- Bonjour, UDP/IP and TCP/IP connection and disconnection notifications
- ALERT! Script errors and warnings from the MIDI/OSC engine

PLEASE NOTE: When the log view is opening the application will enable a slower code path to export and display the log messages. This will have a performance impact on the application, it is recommended to only open and enable **network not time performance** this when trying to trace down issues or troubleshooting problems during development.

The additional buttons in the top right provide the following functions, from left to right:

- **Pause** the logging of messages
- Display **timestamp** for messages
- **Clear** the log of all messages
- **Close** the log

On desktop platforms, it is possible to open the log view in a separate window using an option in the preferences.

Message Mapping

The Message Mapping dialog displays a sortable table of all MIDI, OSC, LOCAL, and GARBAGE messages configured in the control surface document.

The table can be sorted by each column in the table by clicking/tapping on the table header and each column is resized **network not time performance**. Re-sorting is by user-this may slow when troubleshooting problems during development.



The additional buttons in the top right provide the following functions, from left to right:

- **Clear** the log of all messages in the table
- **Close** the dialog

The columns for each type of message are as follows:

- MIDI
  - **Control** - The control the message is configured for
  - **Key** - The value object that will trigger the send of the message
  - **Value** - The value of the message
  - **Ch** - The channel of the MIDI message
  - **Type** - The type of the message in the MIDI message
  - **Val** - The value of the second data byte of the MIDI message if any
  - **Live** - The connections the message is configured to be sent/received on (left to right from first to last connection, if enabled, if enabled)
- OSC
  - **Control** - The control the message is configured for
  - **Path** - The OSC path of the message, or the name of the message
  - **Address** - The OSC address of the message
  - **Arg** - The type of the first argument of the message
  - **Live** - The connections the message is configured to be sent/received on (left to right from first to last connection, if enabled, if enabled)
- LOCAL
  - **Control** - The control the message is configured for
  - **Path** - The OSC path of the message, or the name of the message
  - **Arg** - The type of the first argument of the message
  - **Live** - The connections the message is configured to be received on (left to right from first to last connection, if enabled, if enabled)
- GARBAGE
  - **Control** - The control the message is configured for
  - **Path** - The OSC path of the message, or the name of the message
  - **Arg** - The type of the first argument of the message
  - **Live** - The connections the message is configured to be received on (left to right from first to last connection, if enabled, if enabled)

Font Viewer

The Font Viewer provides a way to browse all characters of the typefaces included with the application.

The application includes a number of fonts available to all control surfaces, to guarantee that any document shared between operating systems and platforms will be rendered exactly the same - without errors or missing symbols.

At the top there is a number of font options, **Default** and **Monospaced** workspaces in the **font** panel of the **Editor** and **Control** modes.

Clicking or tapping any character in the font viewer will copy that character to the clipboard.



When rendering any character, TouchOSC will search a number of fallback typefaces:

- Roboto
  - Liberation Mono
  - Noto Sans
  - Noto Serif
- Monospaced
  - Bitstream Vera Mono
  - Noto Sans
  - Noto Serif

Unsharpening the **Include fallback fonts** option will limit the display of characters to the first font in the fallback chain only.

PLEASE NOTE: The option only affects the display in the font viewer itself, and not the rendering of **Label** and **Text** controls.

Editor - Document

A document is the top level container for all controls and the root node of the control hierarchy tree.

A document's properties are edited from the display and document properties panel, global scripting options for messages and the device, and the document's background color.

A document can also have a set of attached and define control callback functions, which will be called before any of the local callbacks and allow to capture and override global behavior from any Groups/etc in the target controls. See Global Callback Functions for details.

Note that also controls will be scaled proportionally to the available display size when rendered in control surface mode. This is to allow for different screen resolutions and aspect ratios to be supported. Use the anchor properties to control which part doesn't scale, and the application's behavior for how scaling is performed.

- Width
- Height
- Scale
- Running
- Comments
- Background Color

Width

The width of the document.

Height

The height of the document.

Color

The background color of the document.

Routing

The global routing dialog determines whether copies of received messages or input will pass any other type of configured messages to be sent or suppressed.

The left column lists all possible types of inputs that can change a control's values. The bottom row lists all possible types of outputs that can be sent in response to a control's value changing.

Any event changing a control's value will be tagged with the type of action that caused the value change. When a control's value is changed by receiving a message, the message will be tagged with the type of input that the control is set to change. The application will then consult the routing section of the document to determine which of the outputs the message should be sent to, and which should be suppressed.

For **example** if an incoming local unselected MIDI message that will update the control's value on receipt, and an enabled OSC message that updates the value as a trigger, the incoming MIDI message will only be sent to the OSC message to be sent if the close all the interaction between MIDI input in the left column and OSC output in the bottom row is enabled.

Comments

Comments can be entered and the current selection will be highlighted in the text structure. For each control in the document a **comment** section has can be toggled with the use and not control in the right hand panel.

Controls that are containers and hence can contain child controls will be displayed as two notes that can be entered and collapsed using the arrow icon to the left of the control name.

Document Tree

The document tree displays a hierarchical view of all controls in the document.

Controls can be selected and the current selection will be highlighted in the tree structure. For each control in the document a **comment** section has can be toggled with the use and not control in the right hand panel.

Controls that are containers and hence can contain child controls will be displayed as two notes that can be entered and collapsed using the arrow icon to the left of the control name.

Editor - Control

A control is the main object and basic building block for constructing control surfaces.

A control has properties and **scripts** that define its appearance and behaviour.

Control properties can be changed in the editor and by messages and scripts.

All control has a core set of common properties and another set of properties specific to their type. Not all control types might use all the common properties or use them in the same way.

For a list of control type-specific properties, see the control reference.

- Type
- Name
- Tag
- Frame: x, y, w, h
- Color
- Visible
- Active
- Interactive
- Touch Events
- Scripts
- Background
- Parent
- Children
- Information

Editor - Control - Properties

Control properties can be changed in the editor and by messages and scripts.

All control has a core set of common properties and another set of properties specific to their type. Not all control types might use all the common properties or use them in the same way.

For a list of control type-specific properties, see the control reference.

- Type
- Name
- Tag
- Frame: x, y, w, h
- Color
- Visible
- Active
- Interactive
- Touch Events
- Scripts
- Background
- Parent
- Children
- Information



Type +

The type of the control.

See the [control messages](#) for all possible types of controls.

Name +

The name of the control.

A name may be set freely assigned to a control and does not have to be unique. Use the property to identify controls in messages or scripts.

Tag +

A tag for the control.

A tag may be set freely assigned to a control and does not have to be unique. Use the property to identify controls in messages or scripts.

X, Y, W, H +

The x and y position and the width and height of the control.

Note: All are integer values.

Color +

The color of the control.

Depending on the type of control, the property will be used as a base color from which multiple shades will be generated for different parts of the control.

Locked +

The locked state of the control.

When a control is locked, it cannot be started or otherwise modified in the editor. To unlock a control, right-click to bring up the context menu and choose **Unlock** from the **Control** menu.

Visible +

The visible state of the control.

If the control property is set to off, it (and) hide the control not being rendered, it will also not be considered as the target for any potential interactions, same as the [Interactive](#) property is also set to off.

Similarly to the locked state, it can still be selected in the editor when directly or by lasso selection, to change the visible state back to on.

Note: The set of partners currently associated with the control will not be modified when the property changes.

Interactive +

The interactive state of the control.

The property determines if the control can be considered when picking the target of a pointer interaction.

Note: The set of partners currently associated with the control will not be modified when the property changes.

Background +

Determines if the control should be rendered with a background.

Outline +

Determines if the control should be rendered with an outline and which style of outline.

Possible outline styles are:

- FULL
- DASHED
- DOTS

Grab Focus +

Determines if the control will capture pointers that are interacting with it.

If the property is set to off, a pointer will only be associated with the control for as long as the pointer's position is within the control's bounds. Moving outside will release the control.

If the property is set to on, the control will be added to the list of the control's pointers after a first entry the control's bounds. Moving outside will not release the control.

Pointer +

Determines which pointer status priority if there's multiple pointers interacting with the control.

The type of control can only react according to its interaction flags. For example, a button can receive a value like 0.5 if it's touched by two pointers at the same time. The pointer priority property is used to determine which pointer will be associated as the most significant one. The one that will be used when querying the control's state and values.

Possible values are:

- SUBSET
- MOST

Corner +

Determines the amount of corner rounding the control will be rendered with.

A value in the range that is 0 will be sharp, with no rounding to rendered corners.

Orientation +

Determines the orientation of the control, which will influence the control's rendering and behavior depending on control type.

Editor - Control - Values

A control can have multiple value objects that are related to a control's interface state.

Values that are [bounded](#) to their max or min state will trigger the [messages](#) and [event](#) behavior accordingly.

A value object can be changed by pointers, [messages](#) and [scripts](#). Whenever one value object changes, the application will notify the control of a control event and will update the [control](#) interface. It is defined what kind of messages that specific the changed value or their list of triggers.

All controls have a **Touch** value object, the BOOLEAN value of which indicates if there are any pointers currently touching the control.

- Locked
- Type
- Current
- Default Pull
- Default Current Lock



Locked +

The locked state of the value object.

If a value is locked it cannot be changed by any actions, messages or scripts.

Type +

The type of the value object.

Possible value types are:

- BOOLEAN
- INTEGER
- FLOAT
- STRING

Default +

The default value of the value object.

The value that this value object will hold when the document is loaded initially.

Current +

The current value of the value object.

Default Pull +

A factor ranging from 0 to 1.0f that specifies how strongly the **Current** value will be pulled towards the **Default** value when the control is not currently the target of any pointer.

Note: The value property is currently only available for values of type FLOAT.

Default Current Lock +

The lock size used for the **Default** and **Current** value fields can be used to lock both for the same value, meaning if one changes, the other will also change.

Enabling this property will make sure that the value object will be saved with its current value as the default value, which will then be restored when the document is loaded.

Editor - Messages

TouchOSC supports bi-direction message types such as sockets.

MIDI, OSC and local messages can be sent in response to control [value object](#) changes using a message [trigger](#) property.

MIDI and OSC messages can also be sent using [script functions](#) and can be received using [callback functions](#).

Messages can be edited by pressing the **plus** button at the right of the message [control](#) header in the [editor panel](#) and selecting one of the available message types.

Messages can be removed by pressing the **minus** at the right of each individual message's panel header and [confirming](#) the removal.

A control can have an unlimited number of messages configured. Messages will be processed in the order they are listed in the interface, the order of messages can be changed by dragging the message to a new position in the list.



- MIDI
- OSC
- Local
- Gamepad

MIDI +

A MIDI message is send and receive on one of the configured [MIDI connections](#).

See the section on [MIDI messages](#) configuration for details.

OSC +

An Open Sound Control message is send and receive on one of the configured [OSC connections](#).

See the section on [OSC messages](#) configuration for details.

Local +

A local message is efficiently send messages between controls in a document without the overhead of a communication protocol or network stack configuration.

See the section on [Local messages](#) configuration for details.

Gamepad +

A gamepad message for receiving inputs from a connected game controller.

See the section on [Gamepad messages](#) configuration for details.

Editor - Messages - MIDI

A control can have multiple MIDI messages configured. Messages can be both sent and received on multiple [MIDI connections](#).

When a message is enabled in [editor](#) it will be sent on the configured [connections](#) when one of the control's values is [touched](#), [changed](#), and the [trigger](#) message matches the change condition of the trigger.

When a message is enabled in [editor](#), it will be considered in the real-time against received MIDI messages if the message's [connections](#) are enabled on one of the enabled [connections](#).

Messages matching depends on the type of MIDI message. TouchOSC defines part of a MIDI message as the **address** and uses this to distinguish between the messages, and which part of the message is the **current** value.

The type of the message and the **address** part can be used to determine if the incoming message matches the configured message. Matching the **current** part of the message is then used to update the control's value with the value from the message.

See the message [matching](#) section for terms on address [matches] for each message type.

- Send
- Receive
- Feedback
- No Duplicates
- Connections
- Trigger
- Type
- Matching



TEST +

Trigger sending of the message manually.

The TEST button will send the message immediately at current [value].

All message settings will be evaluated except for the **Enabled**, **Send**, **Receive**, **Feedback**, **No Duplicates** flags and **Trigger** configurations.

Enabled +

The enabled state of the message.

If a message is not enabled, it will neither be sent or matched to received messages.

Send +

The send enable flag of the message.

Only if the send flag is set to on for the message, it will be considered for sending when any of the values enabled on [control](#) change, and the [trigger's](#) change condition is met.

Receive +

The receive enable flag of the message.

Only if the receive flag is set to on for the message, it will be considered for being matched against any incoming MIDI messages received on any of the enabled [connections](#).

Feedback +

The feedback enable flag of the message.

The feedback flag is used when receiving of the message's in situation where both the send and receive flags are matched and a received MIDI message changes a value that is also enabled for a send of MIDI and [trigger](#) for the message.

If a received MIDI message, and subsequent change of value, would be sent out as the message is to be sent again immediately, this could cause an unintentional feedback loop in the interaction with another application, hence this flag is set to off by default when creating a new message.

A No feedback enable flag is set off by default because of the potential problems this behavior could cause. Please handle with care.

A It is advised to only enable this flag if the receiving side shows problems with duplicate messages.

No Duplicates +

Prevent the sending of duplicate messages.

Enabling this flag will keep a history of the most recently sent message and prevent sending of identical messages (matches them to be equal).

Sending of duplicate messages can occur in some cases such as message overlap [actions](#), for example when scaling a value, the same message could be sent out multiple times for the same value.

A It is advised to only enable this flag if the receiving side shows problems with duplicate messages.

Connections +

The connections that the message should be sent and received on. The numbers refer to the numbered MIDI connections configured in the [MIDI preferences](#) configuration.

If the message is enabled to be sent, it will be sent out on the connections enabled here.

If the message is enabled to be received, any received MIDI messages will only be considered to be matched if it is received on one of the enabled connections.

The special connection labeled **?** enables sending/receiving messages on all connections.

Trigger +

The trigger conditions for the message to be sent.

If the [message](#) **send** flag is set to on, it will be sent when any of the value objects enabled here change, and also if the configured change condition is being met.

Any of the control's values can be enabled individually here, with one of the possible change conditions of:

- **ANY** - Any change in value will cause the message to be sent.
- **CHANGE** - The value must change to a different value for the message to be sent.
- **TRUE** - The change causing the value to become set causes the message to be sent.

Type +

The type of MIDI message.

TouchOSC supports the following types of MIDI messages:

- Note On/Off
- Pitch wheel
- Control change
- Program change
- Channel pressure
- Poly-key
- Control change
- Program change
- Channel pressure
- Poly-key

Depending on the type of message selected the configuration of the message fields will change. For any of the selected types of any given MIDI message type, the value and value part can be one of:

- **CONSTANT**
  A constant value.
- **INDEX**
  The current value of the control is its upper-left of child controls.
- **VALUE**
  The current value of one of the control's [value objects].
- **MAGNITUDE**
  The current value as one of the control's [gamepad].

All values will be converted to integer types and some value types accept the CONSTANT can optionally be scaled before sending (set to reverse after reception).

Scaling is applied using the following calculation:

```
out = out * s + o; (see : Arc)
```

Matching +

If the [receive](#) flag is set to true on a message, it will be considered as the target of any received MIDI messages if its [connections](#) are enabled.

Matches of messages will determine how an incoming message is matched to the configuration. Matching can be reversed and accumulated according to the message's configuration. Only if an incoming message's **type and address** matches the configured message will the received value be considered according to the message's configuration.

If the message configuration uses a control value or property for the [address](#) part of the message type, the value of the control will be used as the address when matching.

If the message configuration uses a control value or property for the **payload** part of the message type, the value of the control will be used as the payload when matching.

The following table lists the part of a MIDI message that the MIDI connection address and payload for each of the supported message types.

Type	Address	Payload
Note On/Off	Channel + Note	Velocity
Poly-pressure	Channel + Note	Velocity
Control change	Channel + Controller	Value
Program change	Channel	None
Channel pressure	Channel	Value
Pitch bend	Channel	Value

A Received messages of type System exclusive can only be processed using [scripting](#) and only at the document root level.

Editor - Messages - OSC

A control can have multiple OSC messages configured. Messages can be both sent and received on multiple [OSC connections](#).

When a message is enabled in [editor](#), it will be sent on the configured [connections](#) when one of the control's values is [touched](#), [changed](#), and the [trigger](#) message matches the change condition of the trigger.

When a message is enabled in [receive](#), it will be considered in the real-time against received OSC messages if the incoming message is received on one of the enabled [connections](#) and the [address pattern](#) matches the received message.

- Send
- Receive
- Feedback
- Relative
- Accumulate
- No Duplicates
- Trigger
- Arguments
- Patch Matrix

TEST +

Trigger sending of the message manually.

The TEST button will send the message immediately at currently configured.

All message settings will be evaluated except for the **Enabled**, **Send**, **Receive**, **No Duplicates** flags and **Trigger** configurations.

Enabled +

The enabled state of the message.

If a message is not enabled, it will neither be sent or matched to received messages.

Send +

The send enable flag of the message.

Only if the send flag is set to on for the message, it will be considered for sending when any of the values enabled as triggers change, and the trigger's change condition is met.

Receive ↑

The receive enable flag of the message.

Only if the receive flag is set to on for the message, it will be considered for being matched against any incoming OSC messages received on any of the connections enabled for the message.

Feedback ↑

The feedback enable flag of the message.

The feedback flag controls the processing of the message in situations where both the send and receive flags are enabled and a received OSC message changes a value that is also enabled in the list of send triggers for the message.

If a received OSC message, and subsequent change of value, would also cause the same message to be sent again immediately, this could cause an undesirable feedback loop in the interaction with another application, hence this flag controls if the message should be sent again in this case.

The feedback enable flag is set to off by default because of the potential problems this behavior could cause. Please handle with care.

No Duplicates ↑

Prevent the sending of duplicate messages.

Enabling this flag will keep a history of the most recently sent message and prevent sending of identical messages multiple times in a row.

Sending of duplicate messages can occur in certain cases and message configurations, for example when scaling the high resolution floating point value of a control to a smaller value range or integer/boolean values.

⚠ It is advised to only enable this flag if the receiving side shows problems with duplicate messages.

Connections ↑

The connections that the message should be sent and received on. The numbers refer to the numbered OSC connections configured in the OSC connections configuration.

If the message is enabled to be sent, it will be sent only on the connections enabled here.

If the message is enabled to be received, any received OSC messages will only be considered to be matched to this message if received on one of the connections enabled here and the address part matches the received message.

The special connection labelled '∞' enables sending/receiving messages on all configured connections.

Trigger ↑

The trigger conditions for the message to be sent.

If the message send flag is set to on, it will be sent only if any of the value objects enabled here change, and only if the configured change condition is being met.

Any of the control's values can be enabled individually here, with one of the possible change conditions of:

- **ANY** - Any change in value will cause the message to be sent
- **RISE** - Any change causing the value to increase will cause the message to be sent
- **FALL** - Any change causing the value to decrease will cause the message to be sent

Address ↑

The address part of the message.

The address of the message can be configured with a series of partials that will be evaluated to produce the final address string.

The address of the message will be used to match any incoming OSC messages and determine if the received message should be routed to this control.

Arguments ↑

The arguments of the message.

The arguments of the message can be configured with a series of partials, each one representing a single argument to the message.

When sending messages, all argument partials will be evaluated and replaced with the actual value depending on each partial's configuration.

When receiving messages, the received message's arguments will be matched sequentially against the list of partials, and any value objects and properties used in partials will be updated with the received values according to each partial's configuration.

Partials ↑

Partials are the building blocks for constructing OSC message address and argument descriptions. The partials will be evaluated from left to right according to their configuration when producing the actual address and arguments of a message.

Partials can be added to the address and argument lists using the plus button on the right and selecting one of the possible partial types.

A partial can be of the following types:

- **CONSTANT**
  A constant value.
- **INDEX**
  The current index of the control in its parent list of child controls.
- **VALUE**
  The current value of one of the control's value objects.
- **PROPERTY**
  The current value of one of the control's properties.

A partial can be selected and the selected partial can be deleted by pressing the DELETE key on desktop platforms, or right-clicking (long-pressing on mobile platforms) on the partial and confirming the delete in the context menu.

When a partial is selected, more configuration options will be shown depending on type of the partial and if the partial is part of the message address or argument list. Possible options include scaling the value and conversion to one of the following types:

- BOOLEAN
- INTEGER
- FLOAT
- STRING

Scaling is applied using the following calculation:

```
out = min + in * (max - min)
```

The order of partials can be changed by dragging a partial to a new position inside the list.

→ TouchOSC Manual

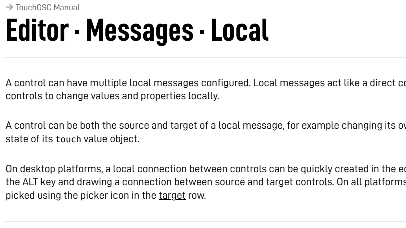
Editor · Messages · Local

A control can have multiple local messages configured. Local messages act like a direct connection between controls to change values and properties locally.

A control can be both the source and target of a local message, for example changing its own color based on the state of its touch value object.

On desktop platforms, a local connection between controls can be quickly created in the editor by holding down the ALT key and drawing a connection between source and target controls. On all platforms, the target can be picked using the picker icon in the target row.

- Enabled
- Trigger
- Source
- Target
- Scale
- Conversion



Enabled ↑

The enabled state of the message.

If a message is not enabled, it will not be sent.

Trigger ↑

The trigger conditions for the message to be sent.

If the message enabled flag is set to on, it will be sent only if any of the value objects enabled here change, and only if the configured change condition is being met.

Any of the control's values can be enabled individually here, with one of the possible change conditions of:

- **ANY** - Any change in value will cause the message to be sent
- **RISE** - Any change causing the value to increase will cause the message to be sent
- **FALL** - Any change causing the value to decrease will cause the message to be sent

Source ↑

The source value to be sent to the target control.

The source value can be one of the following types:

- **CONSTANT**
  A constant value.
- **INDEX**
  The current index of the control in its parent list of child controls.
- **VALUE**
  The current value of one of the control's value objects.
- **PROPERTY**
  The current value of one of the control's properties.

Target ↑

The target value and control of the message.

Enabling target picking mode using the picker icon on the right will assign the next selected control in the editor as the target control of the message. Pressing the picker icon again (or pressing the ESC key on desktop platforms) will cancel target picking mode.

After selecting a target control, the name of the target control will be displayed and clicking on the target control's name will select the control in the editor.

The target value menu will be populated with the available target values. The target value can be one of the following types:

- **VALUE**
  One of the target control's value objects.
- **PROPERTY**
  One of the target control's properties.

Scale ↑

The scaling to apply to the source value.

Scaling is applied using the following calculation:

```
out = min + in * (max - min)
```

Conversion ↑

The type conversion to apply to the source value.

The conversion can be one of the following types:

- BOOLEAN
- INTEGER
- FLOAT
- STRING

→ TouchOSC Manual

Editor · Messages · Gamepad

A control can have multiple gamepad messages configured. Gamepad messages can only be received and not sent.

A gamepad message is configured to wait for a certain game controller input on one or multiple configured gamepad connections, and apply the received input value to one of the control's values or properties.

For portability between platforms, and to allow TouchOSC documents to be shared with users using different input devices, all game controllers are mapped to a standard Xbox 360-like controller layout.

This is the same strategy as employed by the ubiquitous Steam game client and the popular SDL2 game development library, and we support the same mapping database format as these.

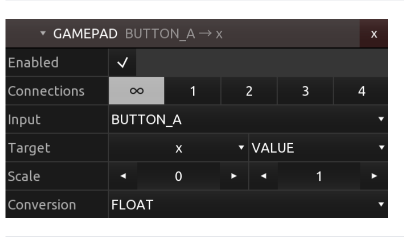
On desktop platforms, if no mapping can be found for the game controller in the included database, TouchOSC will output a warning to the log view upon connection.

It is possible to update and customize TouchOSC's internal database of controller mappings by placing a text file following SDL2's game controller database format in TouchOSC's configuration directory, found at the following location for each platform:

- **Windows:**
  `%AppData%\TouchOSC\`
- **macOS:**
  `~/Library/Application Support/net.hexler.lex/`
- **Linux:**
  `~/.TouchOSC/`

TouchOSC will look for a file called `gamecontrollerdb.txt` at this path once at application launch, and if found, will try to load and apply that configuration.

- Enabled
- Connections
- Input
- Target
- Scale
- Conversion



Enabled ↑

The enabled state of the message.

If a message is not enabled, it will not be processed when a matching input is received.

Connections ↑

The connections that the message should be received on. The numbers refer to the numbered configured game controllers in the Gamepad connections configuration.

The special connection labelled '∞' enables receiving messages on all connections.

Input ↑

The type of input received from the game controller.

The input can be one of the following:

- **STICK_LEFT_X**
  The x-axis of the left analog stick
- **STICK_LEFT_Y**
  the y-axis of the left analog stick
- **STICK_RIGHT_X**
  The x-axis of the right analog stick
- **STICK_RIGHT_Y**
  The y-axis of the right analog stick
- **TRIGGER_LEFT**
  The left trigger at the back of the controller
- **TRIGGER_RIGHT**
  The right trigger at the back of the controller
- **BUTTON_UP**
  The up button on the directional pad
- **BUTTON_DOWN**
  The down button on the directional pad
- **BUTTON_LEFT**
  The left button on the directional pad
- **BUTTON_RIGHT**
  The right button on the directional pad
- **BUTTON_A**
  The face button often labelled A
- **BUTTON_B**
  The face button often labelled B
- **BUTTON_X**
  The face button often labelled X
- **BUTTON_Y**
  The face button often labelled Y
- **BUTTON_STICK_LEFT**
  The button input pressing down on the left analog stick
- **BUTTON_STICK_RIGHT**
  The button input pressing down on the right analog stick
- **BUMPER_LEFT**
  The left bumper button at the back of the controller
- **BUMPER_RIGHT**
  The right bumper button at the back of the controller
- **BUTTON_START**
  The center button often labelled 'Start'
- **BUTTON_SELECT**
  The center button often labelled 'Select' or 'Back'
- **BUTTON_HOME**
  The button at the center of the controller, often showing a logo of the controller manufacturer or game console

Target ↑

The target value or property of the message.

When the configured input is detected on any of the enabled connections, the received numeric value of the input will be written to the target value or property selected here, after scaling and conversion has been applied.

The target can be one of the following types:

- **VALUE**
  One of the target control's value objects.
- **PROPERTY**
  One of the target control's properties.

Scaling ↑

The scaling to apply to the input value.

The input value will be scaled in the same way as other received message values according to the following formula, which is the inverse of the scaling that is applied to outgoing messages of all message types:

```
value = (input - min) / (max - min)
```

It's easiest to think of this inverse scaling this way: Setting `min` and `max` to the expected minimum and maximum input values received from the game controller, this transformation will produce a `value` ranging from `0` to `1`.

Conversion ↑

The type conversion to apply to the input value.

The conversion can be one of the following types:

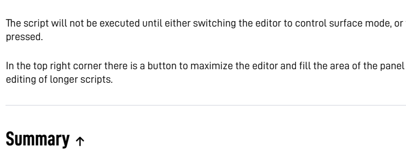
- BOOLEAN
- INTEGER
- FLOAT
- STRING

→ TouchOSC Manual

Editor · Script

The script editor UI for editing script source code. Please see the Scripting API section of the manual for details about the script language and available functions and objects.

- Editor
- Summary
- Log
- Find/Replace
- Run



Editor ↑

The text editor for editing script source code with syntax highlighting and completion.

Every time the text editor looses input focus, the script will be compiled and checked for errors, and if any, the error source line will be highlighted and a short error summary will be displayed in the summary display. For the full error message, open the log view.

The script will not be executed until either switching the editor to control surface mode, or the run button is pressed.

In the top right corner there is a button to maximize the editor and fill the area of the panel on the right for easier editing of longer scripts.

Summary ↑

A short summary of the error message when compiling the script source.

Log ↑

Open the application's log view and show the script messages page.

Find/Replace ↑

Toggle the editor's find/replace tools.

On desktop operating systems the standard keyboard shortcuts for find/replace can also be used to show these tools.

Run ↑

Compile and execute the script code immediately.

Script code is automatically compiled and checked for errors every time the text editor looses input focus, and all scripts will be executed when switching from editor to control surface mode.

The run button will compile and execute the script code **immediately**, making it possible to test and run scripts in the editor without switching to control surface mode.

→ TouchOSC Manual

Editor · Shared Scripts

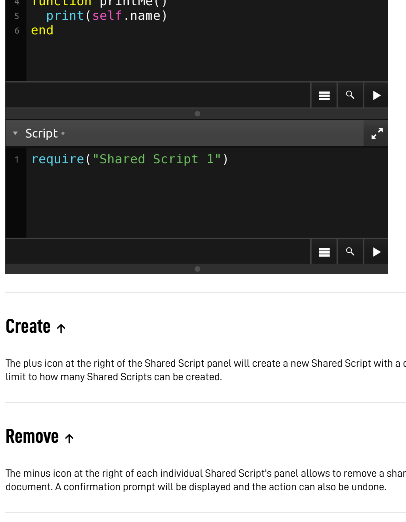
Shared Scripts allow to define common pieces of code that can be included in other scripts.

Shared Scripts can be included in control scripts by name, using the global script function `require`.

The script editor is mostly identical to the editor for control scripts, so we will focus on the differences here.

The Shared Scripts panel will be displayed in the editor panel only at the document root level with no controls selected. If you'd prefer the panel to always show together with the Script Editor, you can enable the Show Shared Scripts panel always preference option.

- Create
- Remove
- Name



Create ↑

The plus icon at the right of the Shared Script panel will create a new Shared Script with a default name. There is no limit to how many Shared Scripts can be created.

Remove ↑

The minus icon at the right of each individual Shared Script's panel allows to remove a shared script from the document. A confirmation prompt will be displayed and the action can also be undone.

Name ↑

Change the Shared Script's name.

As the name is used to identify the script when calling the `require` function, this name has to be unique. An error message will be displayed if the new name matches any of the existing items.

→ TouchOSC Manual

Editor · Network

The application's editor network feature allows one instance of the application to act as editor server on the network and lets multiple other instances on the network connect as clients.

The editor server will broadcast the current document and all editor actions to all connected clients in real-time.

The editor clients will be locked into control surface mode and not be able to make any edits while connected to the server. Once disconnected from the server, the document can be edited and saved on the client.

Connected clients will still be able to use the control surface with all messages and scripts fully functional, making it possible to test the configuration of a control surface immediately, while making changes on the server.

> **NOTE** that the current values of a control's value objects will only be transmitted **once after the initial connect** or if **explicitly edited** on the server, as it would interfere with the ability to test a control surface's functionality and use, if values would be continually updated or reset to the current state of the server document.

- Client
  - Manual Connect
  - Available Servers
- Server

- Enabled
- Disabled



Client ↵

Manual Connect ↵

Enter an address and IP address or host name and port number manually to connect. The eye-filled part number field toggles between hiding and showing the current network address.

Available Servers ↵

The list of available servers will be populated with discovered hosts that have the [server](#) option enabled.

The TouchOSC uses Zero-configuration networking ([bonjour](#)) to discover other network servers on the local network. Make sure your network configuration allows for Bonjour traffic to be able to find servers.

Clicking/touching a host will be displayed with more info and a connect button to connect to the connection to the selected server.

If an option server host is reachable via multiple network interfaces or IP addresses, a button will be displayed to request details about the possible IP addresses to connect to and determine the best one individually.

Server ↵

Enabled ↵

Enables the builtin network's server portion to listen for incoming client connections.

NOTE: For security reasons the state of this flag is not automatically saved and restored between application runs. You might want to remember to turn the function off and add devices to trusted list.

NOTE: Starting with version 1.x.x, due to stacking protocols, the server can be automatically enabled on startup using the command line parameter `--server`, example: `./core.AIRHAX`

Connections

TouchOSC supports multiple connection types for [messages](#) to be sent and received on.

Each type of connection can have between four and ten separate, simultaneous connections configured, each with different settings.

Each of these numbered connections can be enabled individually in [messages](#) configuration.

- MIDI
- OSC
- BSP
- Gamepad

MIDI ↵

MIDI connections can be used for sending and receiving messages according to the MIDI 1.0 protocol specification [↗](#).

We are assuming to add support for the MIDI 2.0 protocol specification [↗](#) in the future, once more supporting hardware and software is available.

See the section on MIDI connections configuration for details.

OSC ↵

OSC connections can be used for sending and receiving messages according to the Open Sound Control [↗](#) protocol.

See the section on OSC connections configuration for details.

Bridge ↵

Bridge connections act as a virtual MIDI input port that transmit MIDI messages over the network to and from a compatible software application like TouchOSC Bridge for desktop.

An OS-dependent is also provided to connect to an instance of the TouchOSC Bridge application using a USB cable connection.

Each of these configured connections can be individually enabled to send and receive ports in the MIDI connection configuration.

See the section on Bridge connection configuration for details.

Gamepad ↵

Gamepad connections allows receiving input from connected game controllers.

See the section on Gamepad connection configuration for details.

Connections - MIDI

TouchOSC allows to configure ten separate MIDI connections, each with freely assignable input and output ports. Each of these connections can be individually enabled to connect a MIDI [message](#) to be sent and received on.

Connections to TouchOSC Bridge connections can also be configured that have a virtual interface port over the network to and from a compatible software application like TouchOSC Bridge for desktop, which can be used in addition to hardware ports.

On macOS, iOS and Android operating systems there will be additional buttons at the top of this page, to switch to the TouchOSC Bridge connections configuration, and to open the configuration dialog for Bluetooth MIDI connections.



Enable ↵

The checkbox in the top-left toggles the enabled state of the connection. If a connection is not enabled it will not send or receive any messages.

Expand ↵

The arrow button in the top-right toggles the display between editable details of the connection and a short, read-only preview.

Send Port ↵

The name of the MIDI output port to send messages on. This can be any name you enter into the text field or an available network Bonjour service name from the [Browse](#) menu.

If you leave the field empty or no MIDI output port by the specified name is found at runtime, no messages will be sent.

Receive Port ↵

The name of the MIDI input port to receive messages from. This can be any name you enter into the text field or an available network Bonjour service name from the [Browse](#) menu.

If you leave the field empty or no MIDI input port by the specified name is found at runtime, no messages will be received.

Browse ↵

The browse menu will list all sending and received ports currently found on the system, as well as all enabled TouchOSC Bridge connections for selection. Both lists are updated in real time and will show any port added while browsing.



Enable ↵

The checkbox in the top-left toggles the enabled state of the connection. If a connection is not enabled it will not send or receive any messages.

Expand ↵

The arrow button in the top-right toggles the display between editable details of the connection and a short, read-only preview.

Type ↵

The type of OSC connection to establish. TouchOSC supports three connection types to transport OSC messages on:

- **UDP**
  - A UDP connection
  - This is the most common OSC connection type. It is the most common protocol for communicating between OSC-enabled applications and hardware. If you are unsure what type of connection to choose, it is highly likely that this is the correct one.
- **TCP Client**
  - A TCP client connection to another host.
  - If the connection to the other host fails for any reason, the connection will automatically attempt to re-establish the connection.
- **TCP Server**
  - A TCP server connection other clients can connect to.

When using one of the TCP-IP connection types, the **Framing** option will become available to choose one of the framing methods for sending and receiving packets.

- **SLIP [↗](#)**
  - The SLIP [↗](#) protocol uses the simple message framing scheme (con-ouch-ernative) suggested by the RFC 1055 specification [↗](#).
  - This ensures should begin and an end giving the size of the packet, followed by the contents of the packet. This is the default framing method for TouchOSC.
- **OSC 1.0**
  - TouchOSC can use the 4-byte OSC-packet size header framing scheme (con-ouch-ernative) suggested by the OSC 1.0 specification [↗](#).

Host ↵

For the connection types **UDP** and **TCP Client** a host name or IP address can be entered as the destination for sent messages. To browse for other applications and devices that advertise their OSC services on the network, see the [Browse](#) menu.

If you leave the field empty or the destination host is not found at runtime, no messages will be sent.

Send Port ↵

For the connection types **UDP** and **TCP Client** a port number can be entered as the destination for sent messages. The host and port can be completed when selecting one of the discovered services using the [Browse](#) button.

If you leave the field empty no messages will be sent.

Receive Port ↵

For the connection types **UDP** and **TCP Server** a port number can be entered to listen for received messages at TouchOSC's network interface.

If you leave the field empty no messages will be received.

Network Info ↵

The local info box next to the [Receive Port](#) input field will open a dialog listing all IP addresses of all available and active network interfaces on the [Receive Port](#) number.

By default all IPv4 and IPv6 addresses are shown, to exclude IPv6 addresses, press the **Include IPv6** checkbox.

To refresh the list manually, click the **Refresh** button to update the list.

Zeroconf ↵

If a [Host](#) name has been entered, the option to advertise the connection's OSC service on the network using the Zeroconf [↗](#) protocol becomes available with the following options:

- **Enabled**
  - Enable the advertising of the connection's OSC service on the network.
- **Name**
  - The application will use the local name of the device for advertising the connection's OSC service on the network.
- **Custom**
  - Use a custom service name for advertising the connection's OSC receiver on the network.
- **Local Part**
  - Use a custom Bonjour/DNS.-SD hostname for advertising the connection's OSC receiver on the network for use with the provided LocalPart name.
- **Legacy Bonjour**
  - Use a custom Bonjour/DNS.-SD hostname for advertising the connection's OSC receiver on the network for use with the provided LegacyBonjour name.

Browse ↵

The browse menu will list all found, active connections that host and run an OSC connection compatible with the connection's type found on the network using [Zeroconf](#) and [Bonjour](#).

Clicking one of the available hosts will fill in both the [Host](#) and [Send Port](#) fields for the connection. In case the discovered OSC receiver is advertised using multiple IP addresses, those will be listed in a sub-menu under the discovered host's name.

The list of hosts is updated in real time and will show any additional receivers discovered while browsing.

Software Version

Connections - Bridge

TouchOSC allows to configure five separate connections to connect to instances of the TouchOSC Bridge application.

TouchOSC Bridge is an application for desktop computers that sends and receives MIDI messages in real time and that can be used to connect to various USB-connected devices and software.

Each enabled Bridge connection can be individually selected as MIDI input and output port in the [MIDI connections](#) configuration.

- Enable
- Controller
- Browse



Enable ↵

The checkbox at the top-left toggles the enabled state of the connection. If a connection is not enabled it will not receive or send any messages.

Controller ↵

The only visible and named of the game controller to receive input from. Choose any of the four available slots for game controllers using the [Browse](#) menu.

Connected and assigned to this slot is available for the game controller data to be sent to the other Bridge connection.

The configured controller can be removed from the connected using the x button.

Starting with version 1.x.x the slot number will be saved and restored when connections in this slot will be used. This is to have modern games handle game controller connections.

Prior to version 1.x.x the configured controller slot number was not saved, but assigned to the next available slot. This could lead to issues when multiple controllers were added or connected. This meant that only that specific model of controller would be used for the connection.

Browse ↵

The browse button will list all connected game controllers currently found on the system. The list will be updated in real time and will show any additional game controllers added while browsing.

Preferences - General

- User Interface scale
- Options
- Show comments after opening
- Send anonymous usage statistics
- Enable debug logging

User interface scale ↵

Global scaling factor for all of TouchOSC's user interface. Needs a restart of the application to be applied.

Note: This is an experimental feature and maybe cause some user interface controls to behave in certain unexpected ways.

Updates ↵

Automatically control the hosted net website to check for an updated version of the application on startup, at most once a day.

The **Check now** allows to manually check for updates. If there is a new version available, a dialog will be shown with the option to download the new version. Otherwise a dialog will be shown that no new version is available.

Note: No additional information other than the application version, operating system type and architecture will be transmitted to the net website during the check.

Show comments after loading ↵

If a document has values for the "Creator" or "Comments" document properties, display those after loading has finished.

Send anonymous usage statistics ↵

Time window sending of anonymous usage statistics back to the hosted net website.

The only data that is sent about your system and use of the application, such as operating system, version and language, hardware specifics such as amount of RAM and number of CPU cores, type of GPU, and general usage statistics.

We use this anonymous data to determine the future direction of the application's design and the creation of command support for new popular hardware/software configurations. Therefore we strongly recommend leaving this setting enabled, to help us make informed decisions about the application's future.

For more information see our Privacy Policy.

Enable debug logging ↵

The only reason for application to log much larger amount of information about its internal operation.

Note: The print more extremely output could be performance and slow down, and is only encouraged when asked to do so by one of our developers to support staff and only when diagnosing a problem.

The **Clear logs** button will open the operating system's file browser at the location of TouchOSC's log files.

Preferences - Editor

- Appearance
  - Show Grid
  - Snap to Grid
  - Grid Size
- Behaviour
  - Enable Touch MIDI messages
  - Create default OSC messages
  - Create default MIDI messages
  - Flash recognition color with RGB
- Scripts
  - Tab Size
  - Word Wrap
  - Auto Completion
  - Code Conventions
  - Show Deprecated Symbols

Appearance ↵

Show Grid

Toggle the rendering of a grid in the background of the editing surface and configure spacing appearance and colour.

Snap to Grid

Toggle whether controls will snap to the editor grid when being positioned.

Show Rulers

Toggle the rendering of rulers at the top and left edges of the editing surface.

Behaviour ↵

Create default MIDI messages

When creating a new control, automatically create a control and configure a suitable MIDI message as well.

Create default OSC messages

When creating a new control, automatically assign a new name to the control.

Finger navigation only with ALT key

When enabled, pressing a finger control's touchpoint that is traditionally bound to left/right and wheel to switch between values can be used without a control selection mode. This action enables this behaviour only when the ALT key is held down on the keyboard.

Note: This option is only available on desktop platforms.

Script ↵

Font Size

Font size used by the script editor.

Word Wrap

Enables word wrapping in the script editor.

Code Highlighting

Enables colored code highlighting in the script editor.

Code Completion

Enables code completion in the script editor.

Show Shared Scripts panel always

Enables the Shared Scripts panel to always show together with the Script Editor. Otherwise, the panel will only be visible when there are shared scripts and when switching tabs back.

Preferences - Control Surface

Preferences - Control Surface

- Display
- Behaviour
- Color
- Mute Solos
- Fader
- Back Button
- Double Tap
- Long Press
- Horizontal

Preferences - Display

Display ↵

Orientation

Determines the control mode/orientation the control surface will be rendered with.

AUTO will switch the default mode the control surface the same as provisioned in the editor.

AUTO will cause the orientation to be automatically chosen to maximize the space the control surface will occupy, considering the screen size and current aspect ratio.

NOTE: The switching does not change your device's automatic rotation or rotation lock preferences in any way. Please make sure that your device is not rotation locked if you want to use the automatic orientation switching or manually select a landscape orientation for your settings.

Filter

This filter selection allows an additional post-processing image filter to be applied to the control surface when rendered.

Allow sleep

Allow the device to sleep when in control surface mode.

The default is to keep the device awake when in control surface mode.

NOTE: This setting is only available on mobile devices.

Padding ↵

Configure the amount of space to reserve at the top, bottom, left and right edges of the screen when rendering the control surface area.

Back Button ↵

Options related to the back button for setting the control surface view and returning to the initial view.

Double-tap

Configure the back button to require a double-tap instead of a single press.

Vertical

Configure the vertical alignment of the back button.

Horizontal

Configure the the horizontal alignment of the back button.

Preferences - Script

Preferences - Script

- Timeout
- Invalid API Usage

Timeout ↵

In Control Surface mode, long running scripts will be terminated to prevent the application from stalling.

- **SHORT**
  The default setting only one script is allowed to take up to ~200ms.
- **LONG**
  This will allow scripts running multiple seconds it might be helpful to extend the timeout to ~2000ms. Please use this setting with caution and only if absolutely necessary.

When in edit mode, the timeout will be increased to a longer than either of these settings to allow for long running manually launched scripts to help with editing scripts.

Invalid API Usage ↵

When invalid API usage is detected (e.g. accessing non-existent control properties or values), an editing API validation error will be generated. The following options are available:

- **ERROR**
  Log this as a runtime error message will be printed to the SCRIPT log and the script will continue to execute.
- **FAIL**
  Log this as a runtime warning message will be printed to the SCRIPT log and the script will continue to execute. If the API call in question results in a return value, a nil will be returned.
- **IGNORE**
  Ignore the error and print to the SCRIPT log and the script will continue to execute. If the API call in question results in a return value, a nil will be returned.

Preferences - Log View

Preferences - Log View

- Font Size
- Keep Log Lines
- Timestamps
  - Control Value
  - Control Name on path
  - Incoming message connection routing
  - Outgoing message
  - Color on routing editor

Font Size ↵

Font size used by the log view.

Keep log lines ↵

Number of log lines to keep in the scroll back history.

Note: Setting this option to a large value may dramatically impact performance.

Display ↵

Timestamps

Display a timestamp for each logged event.

Control name on path

Display control name in path, with all parent container name separated by a forward slash.

Incoming message connection routing

Display additional information about which connection the incoming message(s) are routed to, or nothing if none of the configured connections match for received message.

Open in separate window

Open the log view in a separate window instead of a split view embedded in the main application window.

Clear on leaving editor

Clear the log view automatically when switching from editor mode to control surface mode.

Preferences - Import

Preferences - Import

Options that apply to the import of touchosc files.

TouchOSC Mk3 defines some OSC messages that are either enabled globally in the application's OSC options or configured per control. Enable this option to automatically create these OSC messages as part of the importing process as the file option to generate these messages per control.

- Appearance
  - Apply Classic Style
  - Apply Color Messages
- Ping
- Accelerometer
- Raw OSC Messages
- Velocity
- Touch
- Visibility
- Position
- Size
- Color
- Control Key Messages
  - Convert to OSC messages

Appearance ↵

Apply Classic Style

This option will configure the properties of all imported controls to closely resemble the look of the original control rendering of the TouchOSC Mk1 app.

Global OSC Messages ↵

Ping

If enabled, will create a script to be generated at document load to periodically send a /sync OSC message on all configured OSC connections.

Note: As the globally configured value for the delay of this message can be set from the TouchOSC Mk1 app's OSC options, this value sets the script interval of the document level to manually change this value.

Accelerometer

If enabled, will create a script to be generated at document load to send a /accxyz OSC message with three float arguments for the x/y/z values of the device's accelerometer. Values are in the range of -1.0 to 1.0 for each axis.

Velocity

If enabled, will create a script to be generated at document load to receive a /velocity OSC message and trigger a short vibration using the current active haptic feedback.

Control OSC Messages ↵

Touch

If enabled, will create a touch only /<control> path (/<control>/x) OSC message to be generated that will send when the control's touch state changes.

Visibility

If enabled, will create receive only /<control> path (/<control>/visible) OSC message to be generated that will apply the received value to the control's visible property.

Position

If enabled, will create receive only /<control> path (/<control>/x) OSC message to be generated that will apply the received value to the control's x position property.

Size

If enabled, will create receive only /<control> path (/<control>/w) OSC message to be generated that will apply the received value to the control's width property.

Color

If enabled, will create a receive only /<control> path (/<control>/color) OSC message and an associated script to be generated that will apply the received string value to the control's color property. The script handles the conversion of RGB(A) HEX to float color values to accepted RGBA color values.

Preferences - MCP Server

Control Key Messages ↵

Convert to OSC messages

If enabled, will convert all key messages found in the source layout to OSC messages.

This option is only for the legacy import from the TouchOSC Mk layout exports, and therefore can be used to trigger the same key press and release events as TouchOSC Mk1.

Preferences - MIDI

- General
  - Auto-MIDI OFF message velocity
- Virtual Ports
  - Create MIDI input
  - Create MIDI output

General ↵

Send MIDI OFF message velocity

If enabled will transmit an accurate MIDI OFF message velocity values unmodified. Otherwise, MIDI OFF message velocity values will be automatically set to zero upon reception.

Virtual Ports ↵

NOTE: These options are currently not available on Android.

Create MIDI input

Controls the creation of a virtual MIDI input port named TouchOSC.

Create MIDI output

Controls the creation of a virtual MIDI output port named TouchOSC.

Preferences - MCP Server

Preferences - MCP Server

Preferences to control TouchOSC's integrated Model Context Protocol server.

The MCP Server runs 48 assistants mapped and with the current document and perform other actions in the application.

For more information on how to integrate the MCP Server into your assistant's tool workflow and other actions are available, please refer to the MCP Server documentation starting from the section titled MCP. We the server dependent on the MCP server you are using.

Can you use the MCP server cascadable at http://127.0.0.1:8079/mcp to work with TouchOSC?

Due to the transient, the server allows local connections only and is available on desktop platforms. This is subject to change in a future release.

- Enabled
- Status
- Port
- Connect



Enabled ↵

Toggles the MCP Server running status on or off.

Note: Set currently, the value of this option will not be saved between application launches and will always default to off on startup.

Status ↵

The current status of the MCP Server.

Displays the server's current state and any error messages related to the server's operation.

Port ↵

The port number the server is listening on for connections.

Changing this value while the server is running will restart the server automatically.

Connect ↵

The URL the server can be reached at when running, plus the commands for adding the MCP Server as a tool for some commonly used AI agents.

Control Reference

In this section we describe all properties and values that are specific to each control type. For a description of common properties and values that all types of controls share, please see the Properties and Values.

All properties and values can also be accessed from within scripts. Please see the scripting API's Properties & Values section for information about the data types and types to use for script access.

- BOX
- BUTTON
- LABEL
- FADER
- TEXT
- ROTARY
- ENCODER
- RADIAL
- XYPAD
- GRAPH
- GROUP
- CANVAS
- MIDI

BOX ↵

A box control is a simple shape with no behavior. It is meant to be affixed to other, for creating decorative elements and backgrounds or to be made visible/invisible using messages and script.

Properties

Name	Description
Shape	The shape of the control.

Values

Name	Type	Description
Shape	Enum	The type of shape. **Rectangle** - A rectangle or square. **Ellipse** - A regular ellipse or circle.

BUTTON ↵

A simple button control with properties to control the press/release behavior. As it uses a PUSH value for its display, it can be also used as a simple LED display.

Properties

Name	Description
Shape	The shape of the control. The type of button. **Momentary** - A push button. **Toggle** - A toggle button, toggle on press. **Toggle Press** - A toggle button, toggle on press.
Type	
Press	Enable the press event for this button.
Release	Enable the release event for this button.
Value Position	If this property is set to on, the button will use the position of the number event to calculate its value. Otherwise it will use the value set by the touch event.

Values

Name	Type	Description
x	Int32	The amount of "pressure" on the button from fully released to fully pressed. The button will be released using the value as signal value for the color of the button fill.

LABEL ↵

A simple non-text display.

Properties

Name	Description
Font	The typeface for rendering the text. **DEFAULT** - The default, scalable width font **MONOSPACED** - A monospaced, fixed width font
Size	The text size.
Line Height	The maximum height of the text.
Alignment	Horizontal and vertical alignment of the text inside the control.
Color	The text color.
Clip	Controls the clipping of the rendered text to the control's frame. **Note** that disabling this property can significantly improve rendering performance in certain cases.

Values

Name	Type	Description
text	STRING	The display text.

TEXT ↵

A multi-line text display.

Properties

Name	Description
Font	The typeface for rendering the text. **DEFAULT** - The default, scalable width font **MONOSPACED** - A monospaced, fixed width font
Size	The text size.
Alignment	Horizontal and vertical alignment of the text inside the control.
Color	The text color.
Clip	Controls the clipping of the rendered text to the control's frame. **Note** that disabling this property can significantly improve rendering performance in certain cases.
Wrap	Controls the word wrapping of the rendered text to the control's frame.

Values

Name	Type	Description
text	STRING	The display text.

FADER ↵

A basic control.

Properties

Name	Description
Control	Show a cursor. **ALWAYS** - Show always **ACTIVE** - Show only if a touch points **INACTIVE** - Show only if no active pointers
Bar	Show a bar. **ALWAYS** - Show always **ACTIVE** - Show only if a touch points **INACTIVE** - Show only if no active pointers
Centered	Render the value bar from the center.
Direction	Type of response to pointer input. **ABSOLUTE** - Value is position **RELATIVE** - Value is relative movement
Factor	Response factor for relative response, in percent of pointer position change.
Grid	Render grid lines and the number of grid lines to render.
Grid Color	The grid line color.

Values

Values

Name	Type	Description
x	FLOAT	The value position.

ZY ▾

A two-dimensional fader control.

Properties

Name	Description
Cursor	Show a cursor. • **ALWAYS** - Show always. • **ACTIVE** - Show only if a fine pointers. • **INACTIVE** - Show only if no active pointers.
Lines	Show value indicator lines. • **ALWAYS** - Show always. • **ACTIVE** - Show only if a fine pointers. • **INACTIVE** - Show only if no active pointers.
Lock X	Lock the x value for pointer input. The value can still be changed through messages and script.
Lock Y	Lock the y value for pointer input. The value can still be changed through messages and script.
Response	Type of response to pointer input • **ABSOLUTE** - Incline touchscreen behavior. • **RELATIVE** - Change relative to pointer position
Factor	Response factor for relative response, in percent of pointer position change.
Grid	Render grid lines over the number of grid lines to isolate.
Grid X Count	The grid line count.
Grid Color	The grid line color.

Values

Name	Type	Description
x	FLOAT	The x value position.
y	FLOAT	The y value position.

RADIAL ▾

A rotary-type control.

Properties

Name	Description
Inverted	Invert the value range.
Centered	Display the value from the center of the value range.
Response	Type of response to pointer input • **ABSOLUTE** - Incline touchscreen behavior. • **RELATIVE** - Change relative to pointer position
Factor	Response factor for relative response, in percent of pointer position change.
Grid	Render grid lines over the number of grid lines to isolate.
Grid Color	The grid line color.

Values

Name	Type	Description
x	FLOAT	The value position.

ENCODER ▾

A circular 3D controller.

Properties

Name	Description
Cursor	Show a cursor. • **ALWAYS** - Show always. • **ACTIVE** - Show only if a fine pointers. • **INACTIVE** - Show only if no active pointers.
Response	Type of response to pointer input • **ABSOLUTE** - Incline touchscreen behavior. • **RELATIVE** - Change relative to pointer position
Factor	Response factor for relative response, in percent of pointer position change.
Grid	Render grid lines over the number of grid lines to isolate.
Grid Color	The grid line color.

Values

Name	Type	Description
x	FLOAT	The value position.
y	FLOAT	The duration of change of the value position.

RADAR ▾

A circular XY control, measuring distance from center and angle.

Properties

Name	Description
Cursor	Show a cursor. • **ALWAYS** - Show always. • **ACTIVE** - Show only if a fine pointers. • **INACTIVE** - Show only if no active pointers.
Lines	Show lines to distance and angle. • **ALWAYS** - Show always. • **ACTIVE** - Show only if a fine pointers. • **INACTIVE** - Show only if no active pointers.
Lock X	Lock the x value for pointer input. The value can still be changed through messages and script.
Lock Y	Lock the y value for pointer input. The value can still be changed through messages and script.
Grid X	Render grid lines for angles and the number of grid lines to isolate.
Grid Color	The grid line color.

Values

Name	Type	Description
x	FLOAT	The distance from the center.
y	FLOAT	The angle. The calculation will take the value of the Orientation property into account.

RADIO ▾

A single-value selection from a number of discrete values.

Properties

Name	Description
Steps	The number of discrete values.
Type	The type of indicator. • **Select** - Highlight only the active value. • **Value** - Highlight all values up to the active one.

Values

Name	Type	Description
x	INTEGER	The selected value.

GROUP ▾

A simple container for child controls. A group will copy the rendering of any contained child controls to its display area.

PAGER ▾

A pager controller for child controls. A pager contains a number of group child controls ("pages") and will display only the active one based on its **page** value. A pager control can be organized to extract the individual page group controls.

Properties

Name	Description
Bar	Display the page tab bar.
Wrap Tabs	The size of the page tab bar.
Double Tap	Require a double tap on the tab bar instead of a single press to change the active page.
Labels	Display the name of each child page's **Label** property on the tab bar.
Tab Size	Font size for labels on the tab bar. Set to zero to use default size and fit the active page label.

Page Properties

Name	Description
Name	The name of the child group control.
Label	The text to display on the parent pager's tab bar for this page.
Background	The background color of the page group control.
Text Color	The colors to use for the parent pager's tab bar button for the page, for both active and inactive states.

Values

Name	Type	Description
page	INTEGER	The active page index.

GRID ▾

A container for two-dimensional, uniform grid of child controls of the same type. A grid control can be organized to extract the individual child controls. The editor will display the child control type's properties in addition to the grid control's own properties.

Properties

Name	Description
Type	The type of child control.
X	The number of columns in the grid.
Y	The number of rows in the grid.
Start	The corner of the grid to start from when creating the child controls.
Direction	The direction in which to proceed when creating the child controls.
Naming	The value to use when assigning the **Name** property of the created child controls.
Automate	Whenever a child control's value changes, mark all other child control values of the same name back to the default value.

Control Surface

Setup Examples

Examples, instructions and recipes on how to set up TouchOSC with other applications and hardware.

We'll be adding more of these in the future, so please check back again at a later time.

Getting Started

- Protocols > MIDI
- Protocols > OSC
- Wireless Lan

How-to's

- Android USB MIDI
- Resolume Wire
- Touch Osc Pro
- Ableton Link
- Logic Pro

Control Surface

Setup - Android USB MIDI

Beginning with the Android 6 (Marshmallow) release, device makers can enable optional MIDI support in the platform.

If your device manufacturer added support for the **USB peripheral mode**, please follow these steps to enable it and make your device appear as a MIDI device in the USB audio devices section, which TouchOSC can then use to send and receive MIDI messages.

1. Turn on Developer Options
2. Enable USB Debugging
3. Select MIDI Hardware

Device Setup ▾

While attached to a USB host, put down from the top of screen and select the entry called **Tap** ... or similar. The exact wording will depend on the customizations applied by your device manufacturer.





From the list of options that opens, select **MIDI** or similar. Again, the exact wording will depend on the customizations applied by your device manufacturer.



Alternatively these options can also be found in the **Android Developer Options** at **Settings > Developer options > Networking > Select USB Configuration**.

TouchOSC Setup ▾

By enabling the **USB peripheral mode** on your Android device, the OS will also create MIDI input and output ports that TouchOSC can use to send and receive MIDI messages via the USB cable connection.

In the TouchOSC MIDI connection preferences select the newly created **Android USB Peripheral Port** (or similar) for both the **Send Port** and **Receive Port** fields.

Quite often, the exact naming of these ports will depend on the customizations applied by your device manufacturer.



Send MIDI messages ▾

We'll use the Pressure pad below to confirm that the MIDI ports are visible to our USB host and that we can receive MIDI messages from TouchOSC.

Pressure is one that we're testing with, and MIDI connections and messages. This is the volume while we are in house for testing our applications.

Launch the Pressure application, switch to the **MIDI** tab page, and check the checkbox next to **Enabled**.

If all went well, you should see a connected MIDI device in the Pressure interface that is named similarly to your device, and messages sent from TouchOSC via the USB connection should be received by Pressure.

As you might have guessed, the exact names of the MIDI ports that your device creates depend on the customizations applied by your device manufacturer.



Control Surface

Setup - Resolume Wire

Resolume Wire is a modular node-based patching environment to create effects, mixers and video generators for Resolume Arena & Avenue.

Wire supports both MIDI and OSC control, but as OSC is the more flexible of the two and can be controlled directly over the network, we'll walk through mapping OSC control from TouchOSC into your Wire project.

- Enable OSC Input
- Configure TouchOSC
- Create OSC nodes
- Print OSC messages

Enable OSC Input ▾

Open the **Wire** preferences, switch to the **OSC** section on the left and enable the **OSC Input**.



Wire is now listening for incoming OSC messages at the **IP Address** and **Sending Port** number displayed here.

Configure TouchOSC ▾

Open **TouchOSC** and make sure that you are connected to the same network as the machine that is running **Wire**.

If you are using the **Bridge** application to send and receive OSC messages over a USB connection (recommended), please note that the **Windows** firewall can prevent this setup from working. Should you run into problems, please disable the **Windows** firewall temporarily and try again.

Open the **MIDI / OSC** connection settings.

Make sure **Connection 1** is enabled and set to type **UDP**.

Use the **Browse** button to search for and select Wire's OSC input on the network and fill in the required values automatically.

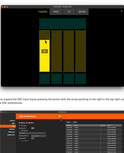
If the Wire instance cannot be found, manually fill in the **IP Address** displayed in Wire's OSC preferences in the **Host** field and the **Sending Port** number in the **Send Port** field.

Close the dialog and open any document configured to send OSC messages, such as the included example layout "1 Fader".

Switch to control mode and move the **Fader** around a few times.



Select the OSC messages from TouchOSC in Minis by using any of the controls in the layout.



In Minis, expand the OSC input by pressing the button with the arrow pointing to the right in the top right corner of the OSC preferences.



If everything is working, you should see the OSC messages sent from TouchOSC to Minis appear in the OSC input log and the button's function is complete.

Create OSC nodes +

Create a new Minis project and double click on the empty project's background to open the node search window.

Enter **OSC** to filter the nodes related to OSC communications.



Select **OSC In** to create an OSC Input node.



Open the node search window again and create a **Read OSC** node.

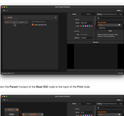


Connect the output of the **OSC In** node to the input of the **Read OSC** node.



In the **Read OSC** node's **Address** field, enter the OSC address that you want to receive.

If you're opened the **Simple Mix** layout in the previous steps, enter /1/fader1.



Open the node search window once again, search for **print** and create a **Print** node.



Connect the **Param** 0 output of the **Read OSC** node to the input of the **Print** node.



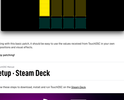
With this setup, the value of the first parameter of OSC messages with OSC address /1/fader1 sent in Minis will be printed to the **Log**.

If you're opened the **Simple Mix** layout in the previous steps, this will be a floating point value ranging from 0.0 to 1.0.

Print OSC messages +

With both the setup in TouchOSC and Minis now complete, let's open the **Log** tab in Minis and confirm that the values are being printed.

The **Log** tab in Minis can be found in the **Log** tab in the queue to the top right of the window.



Send OSC messages from TouchOSC by using the fader control and hold down and you should see the received values printed in the **Log** tab in Minis.



Starting with this basic patch, it should be easy to use the values received from TouchOSC in your own Minis patches and projects.

Happy patching!

5 months ago

Setup - Steam Deck

Follow these steps to download, install and run TouchOSC on the Steam Deck.



- Switch to Desktop Mode
- Download TouchOSC
- Install TouchOSC
- Launch TouchOSC
- Exchange layout
- Starting Mode

Switch to Desktop Mode +

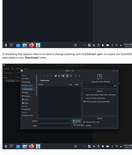
Open the **Steam Menu** using the **Steam Button** and select **Power**.



In the following dialog select **Switch to Desktop**.

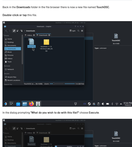


The Steam Deck will now switch to **Desktop Mode**.



Download TouchOSC +

Start the **Firefox web browser** from the task bar at the bottom and navigate to the TouchOSC download section hosted on https://hexler.net/touchosc#download.

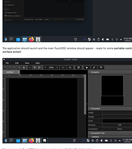


In the **Desktop** section of the downloads, click the button labelled **LINUX**.

A new .sh dialog with all available versions for Linux will appear. Click the button labelled **OK** in the section **x86 1**.

The .sh file will be downloaded to your **Downloads** folder.

Open the **Dolphin** from the task bar at the bottom of the downloads menu in Firefox and select **Downloads** in the list of **Places** on the left.



Extract TouchOSC +

In the file browser **double click or tap the downloaded .sh file** and the archiving tool will launch and display the contents of the .sh file archive.

Select the new top level **TouchOSC** and click the **Extract** button in the top left.

In the dialog that appears, there's no need to change anything, just click **Extract** again to unpack the TouchOSC executable binary **TouchOSC-x86_64**.

Launch TouchOSC +

Back in the **Downloads** folder in the file browser there is now a new file named **TouchOSC**.

Double click on **.sh** file.

In the dialog prompt **"What do you want to do with this file?"** choose **Execute**.

The application should launch and the new TouchOSC window should appear - ready to use **portable control surface** access!

Now you can close the TouchOSC window and close all of Minis's windows as well.

If you are running into any problems following these steps, please do not hesitate to contact me or email us instead!

Gamepad Input +

If you want to use the Master Bus controls on TouchOSC for Someplace Soul's TouchOSC, we have to sure that the Master Bus is enabled in Ableton.

With the Desktop Active Set the Master Bus controls are set thru mouse/keyboard software with the Someplace Active Set we automate with receive regular game controller input.

Launch the Steam client, either from the **Application Launcher** or the Steam shortcut on the desktop.

Add-Steam the **Start Button** to a couple of seconds and a confirmation appears that the action we've been changed.



For more information on how to use game controller input in TouchOSC see Connections: Gamepad

Gaming Mode ↗

Steam client 172.64

Optionally, you might also want to add TouchOSC to the Steam Launcher and use the application in Gaming Mode.

We're assuming that you've completed all the previous steps in the setup.

In the Steam Client use the **Add a Non-Steam Game...** button in the bottom left.



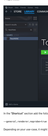
In the Activity that appears, select **Browse...** in the bottom-left.



In the **Find apps to add** dialog, set the **File type** to **All Files**, select the shortcut for TouchOSC Intro Lotro and click **Add Selected Programs**.



Back in the **Add Non-Steam Game** dialog, select **Add Selected Programs**. A few seconds later



And the TouchOSC is now library and both the shortcut is present key in library the settings and in the right and select **Properties...**



In the **Shortcut** section add the following to **LAUNCH OPTIONS**:

-openvr_overlay -overlay_startMinimized

Depending on your use case, it might also be a good idea to add the following as well:

-openvr_no_fs1 -openvr_no_gamepad_ui -openvr_fs



Rename now and want to have a custom icon, you can add URL the exact icon for TouchOSC from [www.](#) in the Shortcut section for "**Icons Resolution**" select "**Ryzen**".



If you are running into any problems following these steps, please do not hesitate to [contact us](#) and we can answer

Setup - Traktor

For the next generation of TouchOSC, Traktor wizard [Andrew Bentley](#) has created the follow up for its legendary top 20 layout.



Log On 2.0 the essential control surface for Native Instruments' [Traktor Pro](#) software and is included with TouchOSC and made it up.

Downloads

Download the detailed manual and Traktor setup file below. The layout itself is included with the application on all platforms.

PDF MANUAL TRAKTOR TSI

Updates

v2.5 - **On Mac OSx**

- Now Quicks syncs, Vista, a TS sends for all four individual Stem Tracks can now be controlled together without having to control transport each track
- Now Quicks syncs, Vista, a TS sends for all four individual Stem Tracks can now be controlled together without having to control transport each track
- Now Quicks syncs, Vista, a TS sends for all four individual Stem Tracks can now be controlled together without having to control transport each track
- All volume level meters on all Decks now display 'warning' colors change when volume level approaches 0dB.

v2.0 - **On Jul 2021**

- Complete redesign of the original way for layout for TouchOSC

Videos

A selection of videos by Andrew demonstrating and explaining the initial workings of the layout.



Watch it on YouTube



Watch it on YouTube



Watch it on YouTube

Setup - Logic Pro

Advanced Setup

Minimum required versions

- Logic Pro X 10.2 or higher

Note: Make sure your computer and all mobile devices are on the same wireless network.

Adding TouchOSC as Control Surface in Logic

- Make sure TS is open
- Start TouchOSC and configure your [MIDI connection](#) as follows:
  Enable a connection and set **Type** to **LAN**
  Check the **Bonjour** button and select the name of the computer running Logic. The **Host** and **Send Port** fields should fill automatically
  Enter a port number in the **Receive Port** field (ex. 9000)
  Set the **Document** option to **Logic/Pad** or **Logic/Touch**, depending on which of these two layouts you plan to use
  Click **Done** to apply the connection settings
- Logic will display the following alert:



- Close the **Edit** window. Logic will add the device as a control surface and open the Control Surfaces Setup window.

Note: If you share the wireless network with other Logic systems, you may want to select the **Don't show again** check box when adding your last TouchOSC device. This will prevent the Setup from appearing for everyone other time you launch Logic. If you do this, you can add additional devices later by choosing **New > Automatic** installation in the Control Surfaces Setup window.

Note: If the AutoFill software forward is enabled, as well will come up when starting for the first OSC message. You can disable this feature in the Control Surfaces Setup window with the OSC Setup.

Note: It is not possible at the time to use customized Layouts or Input OSC commands. However, authorized 3rd party developers can create Logic CS plug-ins for which can use OSC based communication.

Scripting API

Scripting API

TouchOSC's scripting API is based on the [Lua 5.1](#) language and virtual machine with custom additions and modifications.

Scripts can be added to all controls in a document and at the document root level. Each control's script will be executed in its own sandbox.

- Lua Functions
- Global Functions
  - Debugging
  - Messages
- Objects
  - Control
  - Touch Event
  - Tween
  - Timer
  - Animation
- Modules
  - Bit Operations
  - Table Functions
  - Extended Libraries
- Examples

Script - Lua Functions

Script - Lua Functions

The following Lua standard library functions are available.

base ↗

Only the following Lua base library functions are available:

```
assert
error
ipairs
next
pairs
pcall
print
select
tonumber
tostring
type
unpack
xpcall
```

string ↗

All standard Lua string library functions are available.

table ↗

All standard Lua table library functions are available plus the following additions:

```
table.pack(...)
```

Returns a new sequential table created from the elements provided.

```
table.unpack(t)
```

Same as the Lua table library function unpack.

math ↗

All standard Lua math library functions are available plus the following additions:

```
math.clamp(value, min, max)
```

Returns value if min <= value <= max, min if value < min or max if value > max.

os ↗

Lua 5.1.5

Only the following Lua os library functions are available:

```
clock
date
time
```

bit32 ↗

Lua 5.2

Support for bitwise operations has been back-ported from Lua 5.2. All functions are available inside the table bit32.

utf8 ↗

Basic support for UTF-8 encoding has been back-ported from Lua 5.3. All functions are available inside the table utf8.

Script - Global Functions

Script - Global Functions

TouchOSC provides the following global functions.

- Utility Functions
  - getIndex
  - getProperty
  - getTime
  - getMillisecondsSince
  - getMicrosecondsSince
  - getTouchCount
  - getTouchAge
  - getMaximumAge
  - getTouchVelocity
  - setPrecision
- Message Functions
  - OSC Messages
  - sendOSC
  - Complex OSC Messages
  - Complex OSC Messages
- MIDI Functions
  - sendSysex
  - sendMIDI
- Parsing Scripts
  - require

Utility Functions ↗

```
getIndex()
local index = getIndex()
print('Control index: ' .. index)
```

Returns a 0-based integer that specifies the control's index in its parent's children array.

```
getProperty(key)
local name = getProperty('name')
print('Control name: ' .. name)
```

Returns the number of milliseconds since application start.

```
getTime()
local time = getTime()
print('Time: ' .. time)
```

Returns a string with the current local date's year, month, day, hour, minute, second, and millisecond. Optionally, you can specify a format string as the first argument. The format string is the same as the one used by the C function strftime(). If you want to get the number of milliseconds, you can use the format specifier %f.

```
getMillisecondsSince(time)
local time = getTime()
print('Time: ' .. time)
```

Returns the number of milliseconds since application start.

```
local t = getTime()
print(table.unpack(t))
> 10 11 37 893 -- hour minute second millisecond
```

Returns a list with the current local time's `hour` `minute` `second` `millisecond` number values.

```
function hasAccelerometer()

-- example
local b = hasAccelerometer()
print(b)
> true
```

Returns `true` if the host device provides an accelerometer sensor, `false` otherwise.

```
function getAccelerometer()

-- example
local t = getAccelerometer()
print(table.unpack(t))
> -0.925 0.134 4.078 -- x y z
```

Returns a list of three number values that are sampled from the host device's accelerometer sensor. If no accelerometer sensor is available, the values will be all zero.

```
function hasGyroscope()

-- example
local b = hasGyroscope()
print(b)
> true
```

Returns `true` if the host device provides a gyroscope sensor, `false` otherwise.

```
function getGyroscope()

-- example
local t = getGyroscope()
print(table.unpack(t))
> -0.925 0.134 4.078 -- x y z
```

Returns a list of three number values that are sampled from the host device's gyroscope sensor. If no gyroscope sensor is available, the values will be all zero.

```
function getBatteryLevel()

-- example
local n = getBatteryLevel()
print(n)
> 0.84
```

Returns the current battery charge level as a number ranging from `0.0` to `1.0` on devices with an internal battery, `1.0` otherwise.

```
function bytesToInt(number, number, number, number)

-- example
local i = bytesToInt(0x40,0x01,0x00,0x00)
print(i)
> 333
```

Returns a number that is the 32-bit integer representation created from the four byte number parameter values given.

```
function bytesToFloat(number, number, number, number)

-- example
local f = bytesToFloat(0x00,0x00,0x99,0x42)
print(f)
> 76.5
```

Returns a number that is the 32-bit floating point representation created from the four byte number parameter values.

```
function vibrate()

-- example
vibrate()
```

since 1.4.0.230

On devices that support it, a short vibration will be triggered. Has no effect otherwise.

Message Functions ↑

TouchOSC provides the following functions to send MIDI and OSC messages.

MIDI Messages ↑

```
function sendMIDI(table [, table])

-- example
sendMIDI({ 176, 0, 102 })                    -- control change
sendMIDI({ 0xF0, 0x00, 0x01, 0xF7 })  -- system exclusive
```

Send a MIDI message on one or multiple configured connections.

The first argument `table` is a list of byte values that make up the MIDI message.

Starting with version `1.2.1.171` the function will process byte data for multiple messages and continue until either the end of the list or an invalid MIDI message is encountered.

The optional second argument `table` is a list of boolean values that specify which connections to send the message on. If the argument is omitted, the default is to broadcast the message on all configured connections. If the list has fewer elements than the number of connections, the omitted elements default to `false`.

For more usage examples see the Sending MIDI Messages example.

Simple OSC Messages ↑

```
function sendOSC(string [, ... [, table]])

-- example
sendOSC('/1/fader1', 0.5)
sendOSC('/3/xy1', 0.25, 0.75)
sendOSC('/hello', 'world')
```

Send an OSC message on one or multiple configured connections.

`string` is the path of the OSC message to be sent.

The optional argument values `...` will be auto-converted to `boolean`, `float` or `string` OSC types and added to the OSC message as arguments.

Note that argument values are never auto-converted to `integer` OSC types as scripts do not treat floating point and integer numbers as separate types. Use the complex OSC message send function instead.

The optional last argument `table` is a list of boolean values that specify which connections to send the message on. If the argument is omitted, the default is to broadcast the message on all configured connections. If the list has fewer elements than the number of connections, the omitted elements default to `false`.

For more usage examples see the Sending OSC Messages example.

Complex OSC Messages ↑

```
function sendOSC(table [, table])

-- example
sendOSC(
  {
    '/complex',
    {
      { tag = 'i', value = 42 },                    -- int
      { tag = 'f', value = 3.14159 },               -- float
      { tag = 's', value = 'Goodbye Cruel World' } -- string
    }
  }
)
```

Sends an OSC message on one or multiple configured connections.

The first argument `table` is a list that represents the OSC message to be sent, where the first element is the path string of the message, and the second element is a list of argument tables with `tag` and `value` keys for each argument:

```
{
    path,
    {
        { tag = 'argumentTypeTag', value = argumentValue },
        { tag = 'argumentTypeTag', value = argumentValue },
        { tag = 'argumentTypeTag', value = argumentValue },
        ...
    }
}
```

Each argument `value` will be converted to an OSC type according to the type `tag` provided:

Tag	OSC Type	
T	Boolean true	
F	Boolean false	
N	Nil	
I	Infinitum	
i	int32	
h	int64	since 1.1.3.141
f	float32	
d	double	since 1.1.3.141
s	string	
S	symbol	since 1.3.9.226
c	ASCII character	since 1.3.9.226
r	32 bit RGBA color	since 1.3.9.226
m	4 byte MIDI message	since 1.3.9.226
b	blob	

If the `tag` key is omitted, the `value` will be auto-converted the same way as when sending simple OSC messages.

The `T` `F` `N` and `I` types do not need a `value` to be specified.

The `b` OSC blob type expects the `value` to be a list of byte values making up the blob data.

The optional second argument `table` is a list of boolean values that specify which connections to send the message on. If the argument is omitted, the default is to broadcast the message on all configured connections. If the list has fewer elements than the number of connections, the omitted elements default to `false`.

For more usage examples see the Sending OSC Messages example.

Bundled OSC Messages ↑

```
function sendOSCBundle(table [, table])

-- example
sendOSCBundle(
  {
    { '/message1', { { tag = 'i', value = 1337 } } },
    { '/message2', { { tag = 'f', value = 2.71828 } } },
    { '/message3', { { tag = 's', value = 'Hello World' } } }
  }
)
```

since 1.3.7.218

Sends one or more OSC messages as a bundle on one or multiple configured connections.

The first argument `table` is a list of one or more OSC messages to be sent. The messages are in the same format as described in Complex OSC Messages above.

The optional second argument `table` is a list of boolean values that specify which connections to send the message on. If the argument is omitted, the default is to broadcast the message on all configured connections. If the list has fewer elements than the number of connections, the omitted elements default to `false`.

For more usage examples see the Sending OSC Messages example.

Shared Scripts ↑

```
function require(string)

-- example
require('Shared Script 1')
require('Shared Script 2')
```

since 1.5.1.255

The `require` function allows to include a Shared Script by name.

If no Shared Script with a name equal to the parameter `string` can be found, the function will return `false` and log an error.

If the Shared Script is found and returns a value, the function will return this value, or `true` otherwise.

> ⚠ **Shared Script code is compiled as an independent chunk of code and therefore cannot access `local` variables and functions of the control script in which it is included. Anything declared in the global environment of the control script is shared between both.**

NOTE that Shared Script code can not call `require` to include other Shared Script code. This might change in a future update.

NOTE that while the `require` function behaves similarly to Lua's native require function, it is a new implementation native to TouchOSC, and will exhibit differences in behavior.

→ TouchOSC Manual

Script · Objects

TouchOSC defines the following objects to represent its internal and native types.

- Control
- Messages
- Rectangle
- Color
- Vectors

→ TouchOSC Manual

Script · Objects · Control

A control object represents a reference to a single control contained within a TouchOSC document.

Using this reference most of a control's properties and values can be queried and set. From within a control's script, the `self` reference can be used to refer to the control's own fields and functions, the `root` reference (since 1.0.5.109) can be used to refer to the document's root control.

Each control is assigned a unique **ID** on creation that remains unchanged during and between application runs, document save/load and editor network transfer.

All controls live in a document tree hierarchy starting at the document root. All controls except for the root have a reference to a **parent** control, and some control types are containers for **child** controls.

During compilation each script will be checked for the definitions of any of the callback functions listed below, which serve as the main customization points during the various stages of TouchOSC's processing of an application frame.

- Fields
  - ID
  - type
  - index
  - parent
  - children
  - properties
  - values
  - messages
  - pointers
- Functions
  - getValueField
  - setValueField
  - getValueProperty
  - setValueProperty
  - notify
  - findByID
  - findByType
  - findAllByType
  - findByProperty
  - findAllByProperty
  - findByName
  - findAllByName
- Callback Functions
  - init
  - update
  - onValueChanged
  - onPointer
  - onReceiveMIDI
  - onReceiveOSC
  - onReceiveGamepad
  - onReceiveNotify

Fields ↑

```
control.ID

-- example
local myID = self.ID
local parentID = self.parent.ID
print(myID == parentID)
> false
```

A unique ID string, generated when the control is created. It remains unchanged over a control's lifetime, during document load/save and during editor network transmission.

```
control.type

-- example
local myType = self.type
print(myType == ControlType.BUTTON)
> true
```

Control type numeric constant, one of the `ControlType` enumeration values.

```
control.index

-- example
local myIndex = self.index
local parentIndex = self.parent.index
```

The control's position in its parent list of child controls, `1 to n` for regular controls, `0` for the document root.

```
control.parent

-- example
local myParentControl = self.parent
local noParentControl = root.parent -- will be nil, root has no parent
```

A reference to the control's parent `Control` object, or `nil` for the document root.

```
control.children

-- example
self.children.button1.visible = false -- set 'visible' property on child 'button1'
self.children['button1'].visible = false -- same as the previous line
local firstChild = self.children[1] -- first child control
local secondChild = self.children[2] -- second child control
print(#self.children) -- print the number of child controls
```

A list of the control's child `Control` objects. The list can be indexed by control name (a string) or index (a number). Control names are user assignable and **not** unique.

Usage	Description
`control.children.name` `control.children[name]`	Returns the first child control with name `name` or `nil` if none is found. Indexing by name is equivalent to calling `control:findByName(name)`.
`control.children[1 to n]`	Returns the child control at index `1 to n` or `nil` if none is found.
`#control.children`	Returns the number of child controls.

```
control.properties

-- example
self.properties.name = 'new_name'
self.properties['name'] = 'new_name' -- same as the previous line
self.name = 'new_name' -- same as the previous line
self.frame.x = 10
self.color = Color(1,0,0)
self.color.r = 0
print(#self.properties) -- print the number of properties
```

A list of the control's properties. The list can be indexed by property name (a string) or index (a number). Property names are unique.

Usage	Description
`control.properties.name` `control.properties[name]`	Returns the current value of the property with name `name` or `nil` if none is found.
`control.properties[1 to n]`	Returns the current value of the property at index `1 to n` or `nil` if none is found. Since version 1.2.6.185 the list is always ordered alphabetically by property name. Earlier versions do not guarantee any particular order.
`control.properties.keys`	Returns a list of all property names for the control in alphabetical order.
`#control.properties`	Returns the number of properties in the list.

NOTE For convenience, indexing a control reference directly using `control.name` or `control[name]`, where name is not one of the field or function names listed here, will implicitly index the control's property list with `control.properties[name]`.

Therefore `control.color` and `control.properties.color` will refer to the same property value.

See Control Properties and Values for a list of possible properties for each control type.

```
control.values

-- example
self.values.x = 1
self.values['x'] = 1 -- same as the previous line
print(#self.values) -- print the number of values
```

A list of the control's values. The list can be indexed by value name (a string) or index (a number). Value names are unique.

Usage	Description
`control.values.name` `control.values[name]`	Returns the current value of the control value with name `name` or `nil` if none is found.
`control.values[1 to n]`	Returns the current value of the value at index `1 to n` or `nil` if none is found. Since version 1.2.6.185 the list is always ordered alphabetically by value name, with the value named `touch` always at the end of the list. Earlier versions do not guarantee any particular order.
`control.values.keys`	Returns a list of all value names for the control in alphabetical order with the value name `touch` always at the end of the list.
`#control.values`	Returns the number of values in the list.

See Control Properties and Values for a list of possible values for each control type.

```
control.messages

-- example
local midiMessages = self.messages.MIDI -- same as: self.messages[1]
local oscMessages = self.messages.OSC -- same as: self.messages[2]
local localMessages = self.messages.LOCAL -- same as: self.messages[3]
local gamepadMessages = self.messages.GAMEPAD -- same: as self.messages[4]
print(#self.messages)
> 4
```

since 1.2.8.185

A list of the control's messages, containing separate lists for each message type. The list can be indexed by:

- message type name (a string): `MIDI`, `OSC`, `LOCAL`, `GAMEPAD`
- index (a number): `1 - 4`

The messages in each of the lists will be in the same order as they are displayed in the editor UI.

See Script · Objects · Messages for a description of the four message object types.

```
control.pointers

-- example
local pointer = self.pointers[1]
print(pointer.ID,
      pointer.x, pointer.y,
      pointer.state,
      pointer.created, pointer.modified)
> 0 33.285 20.393 1 1836924.838 1836924.867
```

A list containing one table for each pointer currently associated with the control during the current frame with the following table keys per pointer:

Key	Description
`ID`	The numeric ID of the pointer. Constant during the pointers' lifetime.
`x`	The x position of the pointer.
`y`	The y position of the pointer.
`state`	The current state of the pointer, one of the possible values of the `PointerState` enumeration.
`created`	The time the pointer event began, in milliseconds as returned by the `getMillis` global function.
`modified`	The time of the last modification of this pointer, in milliseconds as returned by the `getMillis` global function.

Each pointer progresses through the states in the `PointerState` enumeration during its lifetime:

- After being created the pointer will be in state `PointerState.BEGIN` for one frame.
- During its lifetime the pointer will be either in state `PointerState.ACTIVE` or `PointerState.MOVE` depending on whether the pointer's position has changed since the last frame.
- When the pointer event ends it will be in state `PointerState.END` for one frame and will then be removed from the list of pointers.

See Control Callback Functions for more pointer example use.

Functions ↑

```
function getValueField(string, field)

-- example
self:getValueField('x', ValueField.CURRENT) -- same as: self.values.x
self:getValueField('x', ValueField.DEFAULT)
```

Returns a value field of the control value with name `string`, or `nil` if none is found.

The parameter `field` can be one of the possible values of the `ValueField` enumeration and determines which value is returned:

- `ValueField.CURRENT` - Returns the current value.
- `ValueField.LAST` - Returns the value before the last change.
- `ValueField.DEFAULT` - Returns the default value.

Invoking the function with `field` parameter `ValueField.CURRENT` is equivalent to referencing `control.values[string]`.

See Control Properties and Values for a list of possible values for each control type.

```
// Example: SetPortCommand and SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
```

Save as SetPort.ino

The parameterless Type can be any of the following values of the [SerialType](#) enumeration, and determines which value that will be set:

- SerialType::CURRENT - Set the current value.
- SerialType::DEFAULT - Set the default value.

Invoking the function with Type parameter SerialType::CURRENT is equivalent to calling current_value(unsigned int) – shown [...]

```
// Example: SetPortCommandOverload, SetPortQuery
// SET command
SetPortCommand cmd(1, 5, SerialType::CURRENT);  // Port 1, Channel 5, CURRENT
cmd.execute();
// QUERY command
SetPortQuery query(1, SerialType::CURRENT);     // Port 1, CURRENT
query.execute();
```

Retrieves the value of the property's query of the control value with name cv_ag, or 0 if none is found.

The parameter property_ag is one of the possible values of the [SerialType](#) enumeration with the exception of SerialType::ALL. Profile and Archive return all property values.

See Control Property and Settings for a list of possible values for each control type.

```
// Example: SetPortCommandOverload, QueryQuery, Output
// SET command
SetPortCommand cmd(1, 5, SerialType::CURRENT, 115200);  // Port 1, Channel 5, CURRENT, 115200 baud
cmd.execute();
// QUERY command
SetPortQuery query(1, SerialType::CURRENT);             // Port 1, CURRENT
query.execute();
```

Set the value of the property's query of the control value with name cv_ag.

The parameter property_ag is one of the possible values of the [SerialType](#) enumeration with the exception of SerialType::ALL. Profile and Archive return all property values.

See Control Property and Settings for a list of possible values for each control type.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Creates the value(s) in [Control Settings](#) for a control. Archive or profile values.

The parameter cv_ag may set optional parameter values will be copied to the receiving control's cv_ag control and passed to the [Control Notifications](#) callback function only if that control is configured to receive notifications for the control. Calling the function can error may be offset.

The parameter parameter_value can be of type [Control Settings](#) or [Control Property](#) or [Control Notifications](#).

Please note that because the parameter values have to be copied between cv_ag control's contents and because the parameter values are retrieved, it is advisable not to create the cv_ag by function from inside the [Control Notifications](#) callback.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Retrieves the value(s) or object within the control's cv_ag or cv_ag's [Control Settings](#) and [Control Property](#) values.

The parameter cv_ag may set optional parameter values will be copied to the receiving control's cv_ag control and passed to the [Control Notifications](#) callback function only if that control is configured to receive notifications for the control. Calling the function can error may be offset.

The parameter parameter_value can be of type [Control Settings](#) or [Control Property](#) or [Control Notifications](#).

Please note that because the parameter values have to be copied between cv_ag control's contents and because the parameter values are retrieved, it is advisable not to create the cv_ag by function from inside the [Control Notifications](#) callback.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Returns the first child cv_ag's object control type matches the control's cv_ag's [Control Settings](#) value. The optional type parameter can be one of the [ControlType](#) enumeration values. The optional type parameter determines if the search will only examine and descend the child control's hierarchy defined by its type.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Returns a list of child cv_ag's objects control types matches the control's cv_ag's [Control Settings](#) value. The optional type parameter can be one of the [ControlType](#) enumeration values. The optional type parameter determines if the search will only examine and descend the child control's hierarchy defined by its type.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Returns the first child cv_ag's object whose control value of the property named cv_ag matches the provided cv_ag value or any variant of it that exists. The optional last type parameter determines if the search will only examine and descend the child control's hierarchy defined by its type.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Returns a list of child cv_ag's objects whose control values of the property named cv_ag matches the provided cv_ag value or any variant of it that exists. The optional last type parameter determines if the search will only examine and descend the child control's hierarchy defined by its type.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Dependent to calling findByProperty(cv_ag, cv_ag.type(), cv_ag.value());

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Dependent to calling findAllByProperty("name", string_to_search);

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Callback functions [#](#)

If any of the messaging is not to be run synchronous in a callback function, those callback functions are invoked during the various stages of processing of an application thunks.

When considering a control function for registration as a callback, the parameter declarations are optional and the function will be called regardless of the parameters being omitted or not.

See Control Callback Functions for normal implementations.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Called once when the application transitions from setting mode to control surface mode.

Note that the function is type of void and does not take certain conditions.

- When the application control class is the foreground while being suspended as a mobile device
- When the application control class is [Active Desktop](#) and not receive updates from the server that significantly change the structure of the local document

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Called once per application frame **after** all processing of user input and received messages has completed.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Called **after** processing of user input is complete and an active pointers (mouse cursor or touch input) have been updated. Called once per application frame. Called after all [Control Notifications](#) for pointer and keyboard control activation is responded to in the current app or background.

Will only be invoked if there are any pointers associated with the control during the current frame.

The last values passed as parameters to the callback contains a list of all active pointers that have been selected as the application/screen back according to the control's configuration and do not necessarily include all pointers currently active in the system.

For example, a touchless type control will not be activated by a finger **unless** the control's cv_ag is configured to be selected by the application and passed to the control for processing based on the control's configuration.

To ensure all pointers currently associated with a control do receive the [pointer_up](#) callback, [TBD].

Receiving a true from this callback will exit any further processing TouchGUI would normally do as a result of the change to sending of messages.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Called **after** processing of user input is complete and an active pointers (mouse cursor or touch input) have been updated. Called once per application frame. Called after all [Control Notifications](#) for pointer and keyboard control activation is responded to in the current app or background.

Will only be invoked if there are any pointers associated with the control during the current frame.

The last values passed as parameters to the callback contains a list of all active pointers that have been selected as the application/screen back according to the control's configuration and do not necessarily include all pointers currently active in the system.

For example, a touchless type control will not be activated by a finger **unless** the control's cv_ag is configured to be selected by the application and passed to the control for processing based on the control's configuration.

To ensure all pointers currently associated with a control do receive the [pointer_up](#) callback, [TBD].

Receiving a true from this callback will exit any further processing TouchGUI would normally do as a result of the change to sending of messages.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Called **after** receiving a MIDI message and determining that the control should be a receiver of the message according to the [MIDI Settings](#) and before any further processing of processing of processing of messages by control is received or processing.

Receiving a true from this callback will exit any further processing TouchGUI would normally do for the current control as a result of the message by ceasing any control's processing of messages.

NOTE It is critical that this function's call is efficient without unnecessary delays as all messages received for a control are to be received from that callback, processing of the message will end, it will not be passed along to any other controls as the routing table and any further callbacks will be ignored.

For the format of the message with screen-cross parameters see the [sendMIDI](#) function for complete messages.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Called **after** receiving input from a connected game controller and determining that the control should be a receiver of the message according to the control's [MIDI Settings](#) and before any further processing of processing of processing of messages by control is received or processing.

Receiving a true from this callback will exit any further processing TouchGUI would normally do for the current control as a result of receiving the input for changing a control's value [parameter].

The first parameter 'pad' will be one of the possible values of the [SpecialGroup](#) enumerations.

The second parameter 'pad' will be the numeric value as received by the game controller.

See the [Control Callback Functions](#) section and for an overview of how to handle game controller messages.

```
// Example: SetPortCommand, SetPortQuery
// SET command
SetPortCommand cmd(1, 5);  // Port 1, Channel 5
cmd.execute();
// QUERY command
SetPortQuery query(1);     // Port 1
query.execute();
// Output
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
// Port: 1, Channel: 5, Baud: 115200, Data Bits: 8, Stop Bits: 1, Parity: None
```

Called as a result of the control's [send_to](#) function being called by another control.

The parameters cv_ag and cv_optional_ag as well be copied from the calling control's [cv_ag](#) context to the receiving control's cv_context and passed as parameters to the callback function.

Please note that because the parameter values have to be copied between cv_ag control's contents and because the parameter values are retrieved, it is advisable not to create the cv_ag by function from inside the [Control Notifications](#) callback.

Script - Objects - Messages

Script objects represent the different types of messages associated with a control.

The objects representing the different message types described here can be accessed via a control's [messages](#) field.

- MIDIMessage
  - Policy
    - enabled
    - send
    - receive
    - feedback
    - connections
    - forwarding
    - trigger
    - zero
- MIDIMessage
  - Policy
    - enabled
    - send
    - receive
    - feedback
    - connections
    - forwarding
    - trigger
    - zero
- UserMessage
  - Policy
    - enabled
    - Functions
    - Trigger
- DeviceMessage
  - Policy
    - enabled
    - connections

MIDIMessage [#](#)

An object representing a simple [MIDI message](#) or a control's list of [MIDI messages](#).

Fields [#](#)

The following fields above to dynamically configure some of the core properties that are available in the editor UI.

```
midimessage.enabled
midimessage.enabled([ boolean ]) : boolean
```

Save as MIDIMessage_Enabled.ino

A boolean value. Enable or disable the message.

See [Policy - Messages - MIDI]

```
midimessage.send
midimessage.send([ boolean ]) : boolean
```

Save as MIDIMessage_Send.ino

A boolean value. Enable or disable the sending of the message.

See [Policy - Messages - MIDI]

```
midimessage.receive
midimessage.receive([ boolean ]) : boolean
```

Save as MIDIMessage_Receive.ino

A boolean value. Enable or disable the receiving of the message.

See [Policy - Messages - MIDI]

```
midimessage.feedback
midimessage.feedback([ boolean ]) : boolean
```

Save as MIDIMessage_Feedback.ino

A boolean value. Enable or disable allowing message feedback (sending) immediately after receiving.

See [Policy - Messages - MIDI]

```
midimessage.allow_connections
midimessage.allow_connections([ boolean ]) : boolean
```

Save as MIDIMessage_Allow_Connections.ino

A list of boolean values for each of the connections the message should be sent or received on.

If any boolean flag is set among the list the number is subject to change in the future release and will always be the same number as connections available in the [MIDI connection configuration].

See [Policy - Messages - MIDI]

Functions [#](#)

```
midimessage.trigger()
midimessage.trigger([ boolean ])
```

Save as MIDIMessage_Trigger.ino

Trigger and potentially send the message.

Process the message is the same as a case of no configured trigger) but test will. If both the local and local fields of the message are enabled, the message will be sent on its configured connections.

Note that Sends: flag will cause flag might still prevent the sending of the message.

See [Policy - Messages - MIDI]

```
midimessage.on()
midimessage.on([ boolean ]) : midimessage
midimessage.off() : midimessage
midimessage.send_on() : midimessage
midimessage.send_off() : midimessage
midimessage.receive_on() : midimessage
midimessage.receive_off() : midimessage
```

Return the message state proposed for sending.

Process the message based on its current configuration and return the message. If both the local and local fields of the message are enabled, the message will be sent on its configured connections.

Note that Sends: and local fields will not be considered.

See [Script - Global Functions - MIDI Messages]

OSCMessage [#](#)

An object representing a simple [OSC message](#) or a control's list of [OSC messages](#).

Fields [#](#)

The following fields above to dynamically configure some of the same properties that are available in the editor UI.

```
oscmessage.enabled
oscmessage.enabled([ boolean ]) : boolean
```

Save as OSCMessage_Enabled.ino

A boolean value. Enable or disable the message.

See [Policy - Messages - OSC]

```
oscmessage.send
oscmessage.send([ boolean ]) : boolean
```

Save as OSCMessage_Send.ino

A boolean value. Enable or disable the sending of the message.

See [Policy - Messages - OSC]

```
oscmessage.receive
oscmessage.receive([ boolean ]) : boolean
```

Save as OSCMessage_Receive.ino

A boolean value. Enable or disable the receiving of the message.

See [Policy - Messages - OSC]

```
oscmessage.feedback
oscmessage.feedback([ boolean ]) : boolean
```

Save as OSCMessage_Feedback.ino

A boolean value. Enable or disable allowing message feedback (sending) immediately after receiving.

See Script - Messages - OSC

```
Message receiveFrom:
	| port address |
	port := 57120.
	address := '127.0.0.1'.
	^ OSCMessage newFrom: address port: port
```

since 0.0.50

A boolean value. Enable or disable allowing identical messages to be sent in succession.

See Script - Messages - OSC

```
Message allowDuplicates
	| port address |
	port := 57120.
	address := '127.0.0.1'.
	^ OSCMessage newFrom: address port: port
```

since 0.0.50

A list of boolean values for each of the connections the message should be sent or received on.

The list currently has ten entries, but the number is subject to change in a future release and will always be the same number as the maximum parameter for a [MIDI package](/packages/midi) event.

See Script - Messages - OSC

Functions

```
Message copy
	^ self shallowCopy
```

since 0.0.50

Trigger and potentially send the message.

Process the message in the same way as if one of its configured triggers had been set. If both the keyboard and level triggers are configured, the keyboard trigger will be set.

Note that this does not count the hold until set condition.

See Script - Messages - OSC

```
Message value
	| address |
	address := '/test'.
	^ OSCMessage newFrom: address port: 57120 value: 1 value: 2 value: 3 value: 4 value: 5 value: 6 value: 7 value: 8 value: 9 value: 10
```

since 0.0.50

Return the message data prepared for sending.

Process the message based on its current configuration and return the message data in the format used by the [Sound](/sound) engine to send it.

Note: The send and level fields will not be considered.

See Script - Message Functions - OSC Messages

```
Message value
	| address |
	address := '/test'.
	^ OSCMessage newFrom: address port: 57120 value: 1 value: 2 value: 3 value: 4 value: 5 value: 6 value: 7 value: 8 value: 9 value: 10
```

since 0.0.50

Trigger and potentially send the message.

Process the message based on its current configuration and return the message data in the format used by the [Sound](/sound) engine to send it.

Note that a boolean value is used to determine if the message should be sent.

See Script - Messages - Local

Functions

```
Message copy
	^ self shallowCopy
```

since 0.0.50

Trigger and potentially send the message.

Process the message in the same way as if one of its configured triggers had been set. If the keyboard field of the message is set, it will be sent with the value 1.

See Script - Messages - OSC

```
Message value
	| address |
	address := '/test'.
	^ OSCMessage newFrom: address port: 57120 value: 1 value: 2 value: 3 value: 4 value: 5 value: 6 value: 7 value: 8 value: 9 value: 10
```

since 0.0.50

A list of boolean values for each of the connections the message received on.

The list currently has four entries, but the number is subject to change in a future release and will always be the same number as connections available in the [Sampler](/sampler) [Instrument](/instrument).

See Script - Messages - Received

GamePadMessage

An object representing a single [Gamepad message](/gamepad) in a [GamePad](/gamepad) [Instrument](/instrument).

The following fields allow to dynamically configure some of the same properties that are available in the editor UI.

```
GamePadMessage copy
	^ self shallowCopy
```

since 0.0.50

A boolean value. Enable or disable the message.

See Script - Messages - Gamepad

```
GamePadMessage receiveFrom:
	| port address |
	port := 57120.
	address := '127.0.0.1'.
	^ OSCMessage newFrom: address port: port
```

since 0.0.50

A list of boolean values for each of the connections the message received on.

The list currently has four entries, but the number is subject to change in a future release and will always be the same number as connections available in the [Gamepad](/gamepad) [Instrument](/instrument).

See Script - Messages - Gamepad

Script - Objects - Rectangle

A rectangle object built-in to FoxDOT. Will be returned and can be passed anywhere a rectangle is required.

- Fields
- Constructor Functions
- Functions

Fields

```
rectangle.x
```

The x position of the rectangle.

```
rectangle.y
```

The y position of the rectangle.

```
rectangle.w
```

The width of the rectangle.

```
rectangle.h
```

The height of the rectangle.

Constructor Functions

```
Rectangle(0, 0, 0, 0)
Rectangle(x=0, y=0, w=0, h=0)
Rectangle(x, y, w, h)
```

Returns a new rectangle object with:

1. position and size set to (0,0)
2. position and size copied from another Rectangle object
3. position set to the first pair of numbers and size set to the second two numbers
4. position set to the first pair of numbers and size set to the second two numbers

Functions

```
Rectangle.from_points(x1, y1, x2, y2)
Rectangle.from_points((x1, y1), (x2, y2))
Rectangle.from_points(x1=0, y1=0, x2=0, y2=0)
```

Return a new Rectangle object with the top-left corner at (x1, y1) and the bottom-right corner at (x2, y2)

since 0.0.50

Script - Objects - Color

A new object native to FoxDOT. Will be returned and can be passed anywhere a color is required. Color components are stored as floating point values ranging from 0 to 1.

- Fields
- Constructor Functions
- Class Functions
- Operators

Fields

```
color.r
```

The red component of the color.

```
color.g
```

The green component of the color.

```
color.b
```

The blue component of the color.

```
color.a
```

The alpha component of the color.

Constructor Functions

```
Color(0, 0, 0, 0)
Color(r=0, g=0, b=0, a=0)
Color(r, g, b, a)
Color(color, color, color, color)
```

Returns a new color object with:

1. all components initialized with r
2. all components initialized with the first number
3. all components initialized with number
4. red component initialized with the first number, the g component initialized with the second number
5. red component initialized with the first number, the g component initialized with a
6. red, green and blue initialized with the four numbers provided

Static functions

```
Color.from_hex("#FFFFFF")
Color.from_hex("#FFFFFFFF")
```

Returns a Hexadecimal string representation of the color in the format **#RGBA**.

```
Color.from_hex("#FFFFFFFF")
```

Returns a color object created from the hexadecimal string representation. The string can be in one of the following formats: **#RGB#RGBA#RRGGBB#RRGGBBAA**. Any with the letter **#** are more forgiving in parsing the color from the string.

Operators

```
color + color
color + value
color - color
color - value
color * color
color * value
color / color
color / value
color % color
color % value
```

All operators operate component wise and return a new color object.

Script - Objects - Vectors

FoxDOT provides 2,3 and 4 component vector types as two's two's And objects.

- Fields
- Constructor Functions
- Functions
- Operators

Fields

```
vec2.x
vec2.y
```

The x component of the vector.

```
vec3.y
```

The y component of the vector.

```
vec4.z
```

The z component of the vector.

```
vec4.w
```

The w component of the vector.

Constructor Functions

```
Vec2(0, 0)
Vec2(x=0, y=0)
Vec2(x, y)
Vec2(vec2)
Vec2(vec3)
Vec2(vec4)
Vec2(value, value, value, value)
Vec3(0, 0, 0)
Vec3(x=0, y=0, z=0)
Vec3(x, y, z)
Vec3(vec3)
Vec3(vec4)
Vec3(value, value, value, value)
Vec4(0, 0, 0, 0)
Vec4(x=0, y=0, z=0, w=0)
Vec4(x, y, z, w)
Vec4(vec4)
Vec4(value, value, value, value)
```

Returns a new vector object with:

1. all components initialized with 0 & 0
2. all components initialized with the first vector object
3. all components initialized with a value
4. each component initialized with the numbers provided

Functions

```
Vec2(0, 0).length()
Vec3(0, 0, 0).length()
Vec4(0, 0, 0, 0).length()
```

Returns the length of the vector.

```
Vec2(0, 0).normalize()
Vec3(0, 0, 0).normalize()
Vec4(0, 0, 0, 0).normalize()
```

Returns a new vector that is the normalized vector

Operators

```
vec2 + vec2
vec2 + value
vec2 - vec2
vec2 - value
vec2 * vec2
vec2 * value
vec2 / vec2
vec2 / value
vec2 % vec2
vec2 % value
```

All operators operate component wise and return a new vector object.

Script - Enumerations

- Align
- Mouse
- CursorType
- CursorState
- CursorShape
- Font
- MidiMessageType
- MidiMessageStatus
- Orientation
- ButtonState
- ModifierKey
- ControlType
- RateType
- Screen
- Stage
- MouseButton
- VirtualKeyCode
- WindowType

Align

Possible values for a text object's horizontal alignment.

- Align.LEFT
- Align.CENTER
- Align.RIGHT

Align

Possible values for a text object's vertical alignment.

- Align.TOP
- Align.CENTER
- Align.BOTTOM

ButtonType

Possible values for a control object's button type property.

- ButtonType.PUSHBUTTON
- ButtonType.TOGGLEBUTTON
- ButtonType.TRISTATE

ControlType

Possible values for a control object's type field.

- ControlType.BUTTON
- ControlType.BUTTONBOX
- ControlType.LABEL
- ControlType.SLIDER
- ControlType.TEXT
- ControlType.TOOLTIP
- ControlType.CHECKBOX
- ControlType.GROUP
- ControlType.LISTBOX
- ControlType.SPIN
- ControlType.RADIO

CursorDisplay

- CursorDisplay.SHOW
- CursorDisplay.HIDE
- CursorDisplay.DEFAULT

Font

- Font.DEFAULT
- Font.SANS
- Font.SERIF
- Font.MONOSPACE

Possible values for a2 control: object's have property

- font_default
- font_monospace

ComposedShape +

Possible values for a2 control: object's have property

- ComposedShape.STYLE_SOFT_1
- ComposedShape.STYLE_SOFT_2
- ComposedShape.STYLE_SOFT_3
- ComposedShape.STYLE_SHARP_1
- ComposedShape.TRIGGER_LEFT
- ComposedShape.TRIGGER_RIGHT
- ComposedShape.SHAPE_LEFT
- ComposedShape.SHAPE_RIGHT
- ComposedShape.SHAPE_LEFT_RIGHT
- ComposedShape.SHAPE_ALL
- ComposedShape.SHAPE_NONE
- ComposedShape.SHAPE_1
- ComposedShape.SHAPE_2
- ComposedShape.SHAPE_3
- ComposedShape.SHAPE_4
- ComposedShape.SHAPE_ALL_SOFT
- ComposedShape.SHAPE_ALL_SHARP
- ComposedShape.SHAPE_SOFT
- ComposedShape.SHAPE_SHARP
- ComposedShape.SHAPE_MIXED
- ComposedShape.SHAPE_ALL_MIXED
- ComposedShape.SHAPE_MIX_SOFT
- ComposedShape.SHAPE_MIX_SHARP

MIDIMessageType

Possible values for a2 control: object's have property

- MIDIMessageType.NOTE_OFF
- MIDIMessageType.NOTE_ON
- MIDIMessageType.POLY_PRESSURE
- MIDIMessageType.CONTROL_CHANGE
- MIDIMessageType.PROGRAM_CHANGE
- MIDIMessageType.CHANNEL_PRESSURE
- MIDIMessageType.PITCH_BEND
- MIDIMessageType.SYSTEM_EXCLUSIVE
- MIDIMessageType.MIDI_TIME_CODE
- MIDIMessageType.SONG_POSITION
- MIDIMessageType.SONG_SELECT
- MIDIMessageType.TUNE_REQUEST
- MIDIMessageType.EOX
- MIDIMessageType.TIMING_CLOCK
- MIDIMessageType.START
- MIDIMessageType.CONTINUE
- MIDIMessageType.STOP
- MIDIMessageType.ACTIVE_SENSING
- MIDIMessageType.SYSTEM_RESET

Orientation +

Possible values for a2 control: object's a control can property

- Orientation.HORIZ
- Orientation.VERT
- Orientation.HORIZ
- Orientation.VERT

OutlineStyle +

Possible values for a2 control: object's set's control's property

- Set's outline to TRUE
- Set's outline to COMMON
- Set's outline to FRAME

PointerPriority +

Possible values for a2 control: object's have property

- PointerPriority.MOUSE
- PointerPriority.MOUSE

PointerState +

Possible values for a2 control: object's mouse property

- PointerState.MOUSE
- PointerState.ACTIVE
- PointerState.MOUSE
- PointerState.MOUSE

RadioType +

Possible values for a2 control: object's radio type property

- RadioType.RADIO
- RadioType.RADIO

Response +

Possible values for a2 control: object's response property

- Response.IMMEDIATE
- Response.RELEASE

Shape +

Possible values for a2 control: object's shape property

- Shape.RECTANGLE
- Shape.CIRCLE
- Shape.OVAL
- Shape.DIAMOND
- Shape.TRIANGLE
- Shape.ROUNDRECT

ValueField +

- Set.self.value.CURRENT
- Set.self.value.ARRAY
- Set.self.value.ARRAY

ValueProperty +

- Set.self.type to_TYPE
- Set.self.type to_CLASS
- Set.self.type to_CLASS_DEFAULT_CURRENT
- Set.self.type to_DEFAULT_CELL

ValueType +

- Set.self.type to_BOOLEAN
- Set.self.type to_INTEGER
- Set.self.type to_STRING
- Set.self.type to_FIGURE

Script - Constants

Colors

- Colors.x_clear
- Colors.x_black
- Colors.x_white
- Colors.x_red
- Colors.x_green
- Colors.x_blue
- Colors.x_cyan
- Colors.x_yellow
- Colors.x_magenta
- Colors.x_gray
- Colors.x_darkgray
- Colors.x_lightgray
- Colors.x_darkcyan
- Colors.x_darkmagenta
- Colors.x_darkyellow

Common +

Properties and values that are common to all controls, independent of their type. Not all control types will utilize the value of these properties.

Properties

Name	Description	Since
name	A user editable string.	
tag	A user editable string.	10.3.7fb
frame	A Rectangle object.	
owner	A number object.	
visible	A boolean value.	
mouseLocal	A boolean value.	
backgrounded	A boolean value.	
textFont	A boolean value.	
textSize	A boolean value.	
textStyle	A boolean value.	
textAlign	A boolean value.	
textHeight	A boolean value.	
enabled	A boolean value.	
hilited	A boolean value.	
autoHilite	A boolean value.	
hiliteMode	One of the possible values of the hiliteMode enumeration.	
cursor	One of the possible values of the Cursor enumeration.	
script	A string value. The control's script source code.	

Values

Name	Description
frame	A boolean value. true if any portions are associated with the control is in the current frame. false if no portions are in the current frame. If the associated part is partially in a control and extends into otherwise frame it will be true.

BDE +

Properties

Name	Description
shape	One of the possible values of the Shape enumeration

BUTTON +

Properties

Name	Type
shape	One of the possible values of the Shape enumeration.
buttonType	One of the possible values of the ButtonType enumeration.
press	A boolean value.
release	A boolean value.
autoRepCount	A boolean value.

Values

Name	Description
x	A floating point value ranging from 0.0 to 1.0.

LABEL +

Properties

Name	Type	Since
font	One of the possible values of the font enumeration.	10.3.7fb
textSize	An integer value.	
textLength	An integer value.	
textAlign	One of the possible values of the Align enumeration.	
textAlignV	One of the possible values of the VAlign enumeration.	
textStyle	Added: 050411	
textFit	A boolean value.	12.2.01
textWrap	A boolean value.	12.2.01

Values

Name	Description
text	A string value.

TEXT +

Properties

Name	Type	Since
font	One of the possible values of the font enumeration.	10.3.7fb
textSize	An integer value.	
textStyle	One of the possible values of the Style enumeration.	10.3.7fb
textAlign	One of the possible values of the Align enumeration.	12.2.01
textScript	Added: 050411	
textFit	A boolean value.	12.2.01
textWrap	A boolean value.	12.2.01

Values

Name	Description
text	A string value.

FADER +

Properties

Name	Type	Since
vertical	A boolean value.	
currentTracking	One of the possible values of the TrackingMode enumeration.	
fast	A boolean value.	
deltaTracking	One of the possible values of the TrackingMode enumeration.	
centered	A boolean value.	
response	One of the possible values of the Response enumeration.	
responseFactor	An integer value ranging from 1 to 100.	
grid	A boolean value.	
gridOffset	An integer value.	
gridValue	Added: 050411	12.0.5fd

Values

Name	Description
x	A floating point value ranging from 0.0 to 1.0.

XY +

Properties

Name	Type	Since
vertical	A boolean value.	
currentTracking	One of the possible values of the TrackingMode enumeration.	
fast	A boolean value.	
deltaTracking	One of the possible values of the TrackingMode enumeration.	
centered	A boolean value.	
response	One of the possible values of the Response enumeration.	
responseFactor	An integer value ranging from 1 to 100.	
gridX	A boolean value.	
gridXOffset	An integer value.	
gridXValue	Added: 050411	12.0.5fd

Values

Name	Description
x	A floating point value ranging from 0.0 to 1.0.
y	A floating point value ranging from 0.0 to 1.0.

RADIAL +

Properties

Name	Type	Since
invertVal	A boolean value.	
centered	A boolean value.	
response	One of the possible values of the Response enumeration.	
responseFactor	An integer value ranging from 1 to 100.	
grid	A boolean value.	
gridOffset	An integer value.	
gridValue	Added: 050411	12.0.5fd

Values

Name	Description
x	A floating point value ranging from 0.0 to 1.0.

ENCODER +

Properties

Name	Type	Since
invertVal	A boolean value.	
currentTracking	One of the possible values of the TrackingMode enumeration.	
response	One of the possible values of the Response enumeration.	
responseFactor	An integer value ranging from 1 to 100.	
grid	A boolean value.	
gridOffset	An integer value.	
gridValue	Added: 050411	12.0.5fd

Values

Name	Description
x	A floating point value ranging from 0.0 to 1.0.
y	A floating point value ranging from 0.0 to 1.0.

RADAR +

Properties

Name	Type	Since
invertVal	A boolean value.	
currentTracking	One of the possible values of the TrackingMode enumeration.	
fixedAz	A boolean value.	
fixedAzimuth	One of the possible values of the FixedAzimuth enumeration.	
fastAz	A boolean value.	
fastAzX	A boolean value.	
azGrid	A boolean value.	
azOff	A boolean value.	
azOffsetX	A boolean value.	
azOffsetY	An integer value.	
azMirror	Added: 050411	12.0.5fd

Values

Name	Description
x	A floating point value ranging from 0.0 to 1.0.
y	A floating point value ranging from 0.0 to 1.0.

RADIO +

Properties

Name	Type
integer	An integer value.
radioType	One of the possible values of the RadioType enumeration.

Values

Name	Description
x	An integer value ranging from 0 to the value of the integer property minus one.

PAGER +

Properties

Name	Type
`tabbar`	A boolean value.
`tabbarSize`	An integer value ranging from `10` to `300`.
`tabbarDoubleTap`	A boolean value.
`tabLabels`	A boolean value.
`textSizeOff`	An integer value.
`textSizeOn`	An integer value.

Page Properties

Name	Type
`tabLabel`	A string value.
`tabColorOff`	A `Color` object.
`tabColorOn`	A `Color` object.
`textColorOff`	A `Color` object.
`textColorOn`	A `Color` object.

Values

Name	Description
`page`	An integer value ranging from `0` to the number of pages minus one.

Script - Examples

- Control Callback Functions
- Sending MIDI Messages
- Sending OSC Messages
- Control "Double-tap"
- Send Periodic Message
- Send Accelerometer Data
- Snap Fader to Grid
- Reset Control Value

Control Callback Functions ↑

The following script demonstrates all possible callback handlers during TouchOSC's processing of an application frame and all associated events.

As the root level of a document will always be called first if it defines any of the following callback functions (except for the `onReceiveNotify` callback), when putting started with the scripting API, we recommend setting this script at the root level, in order to see all possible events being handled and printed to the log view.

```
function init()
  print('init')
end

function update()
  print('update')
end

function onPointer(pointers)
  print('onPointer')
  for i=1,#pointers do
    local pointer = pointers[i]
    print('\t', pointer.id, pointer.x, pointer.y, pointer.state, pointer.created, pointer.mod
  end
end

function onValueChanged(key)
  print('onValueChanged')
  print('\t', key, '=', self.values[key])
end

function onReceiveOSC(message, connections)
  print('onReceiveOSC')
  print('\t message     = ', table.unpack(message))
  print('\t connections = ', table.unpack(connections))
end

function onReceiveMIDI(message, connections)
  print('onReceiveMIDI')
  local pack = message[1]
  local arguments = message[2]
  print('\t pack      = ', pack)
  for i=1,#arguments do
    print('\t arguments = ', arguments[i].tag, arguments[i].value)
  end
  print('\t connections = ', table.unpack(connections))
end

function onReceiveGamepad(input, value, connections)
  print('onReceiveGamepad')
  print('\t input       = ', input) -- one of the GamepadInput enumeration values
  print('\t value       = ', value)
  print('\t connections = ', table.unpack(connections))
end

function onReceiveNotify(key, value)
  print('onReceiveNotify')
  print('\t key   = ', key)
  print('\t value = ', value)
end
```

Sending MIDI Messages ↑

Send MIDI messages on one or multiple connections.

For more information see the MIDI Messages script documentation.

```
-- control change, controller 0, channel 1
-- send on all configured connections
sendMIDI({ 0xb0, 0x00, 0x00 })
sendMIDI({ MIDIMessageType.CONTROLCHANGE, 0, 0x0 })

-- control change, controller 0, channel 2
-- send on all configured connections
sendMIDI({ 0xb1, 0x00, 0x7f })
sendMIDI({ MIDIMessageType.CONTROLCHANGE + 1, 0, 0x7 })

-- control change, controller 2, channel 4
-- send on all configured connections
sendMIDI({ 0xb3, 0x02, 0x40 })
sendMIDI({ MIDIMessageType.CONTROLCHANGE + 3, 2, 0x4 })

-- send only on connections 1 and 2
sendMIDI({ MIDIMessageType.NOTE_ON, 12, 32 }, { true, true, })
sendMIDI({ MIDIMessageType.NOTE_OFF, 12, 0 }, { true, true, })

-- send only on connections 1 and 3
sendMIDI({ MIDIMessageType.NOTE_ON, 12, 32 }, { true, false, true })
sendMIDI({ MIDIMessageType.NOTE_OFF, 12, 0 }, { true, false, true })

-- send only on connection 2 and 3
sendMIDI({ MIDIMessageType.NOTE_ON, 14, 32 }, { true, false, false, true })
sendMIDI({ MIDIMessageType.NOTE_OFF, 14, 0 }, { true, false, false, true })

-- send system exclusive
sendMIDI({ 0xf0, 0x00, 0x01, 0xf7 })
sendMIDI({ MIDIMessageType.SYSTEMEXCLUSIVE, 0x00, 0x01, 0xF7 })
```

Sending OSC Messages ↑

Send OSC messages on one or multiple connections.

Messages can either be sent using a simple format, where TouchOSC will auto-convert parameter types, or using a complex format, where each parameter type can be specified using OSC protocol type-tags.

For more information see the Simple OSC Messages and Complex OSC Messages script documentation.

```
-- ------------------------------------
-- Send simple OSC messages
--
-- arguments are auto-converted to
-- boolean, float or string (not integer!)
-- ------------------------------------

-- send on all configured connections
sendOSC('/ping')
sendOSC('/ping', 'pong')
sendOSC('/on',true)
sendOSC('/2/fader1', 0.5)
sendOSC('/1/multixy/1', 0.25, 0.75)
sendOSC('/Greeting/Hello', 'Hello',1, true, 'World')

-- send only on connections 1 and 2
sendOSC('/2/fader1', 0.5, { true, true, })

-- send only on connections 1 and 3
sendOSC('/2/xy1', 0.25, 0.75, { true, false, true })

-- send only on connections 1 and 5
sendOSC('/Greeting/Hello', 'Hello',1, true, 'World', { true, false, false, false, true })

-- ------------------------------------
-- Send complex OSC messages
-- with argument type tags
-- ------------------------------------

sendOSC(
  -- message
  {
    -- path
    '/complex',

    -- argument list
    {
      { tag = 'f', value = 0.5 },                   -- true
      { tag = 'T', value = true },                  -- false
      { tag = 'N', },                               -- nil
      { tag = 'I', },                               -- infinitum
      { tag = 's', value = 42 },                    -- string
      { tag = 'i', value = 1337 },                  -- int64
      { tag = 'd', value = 3.14159 },               -- float32
      { tag = 'd', value = 3.14159265359 },         -- double
      { tag = 's', value = 'Goodbye Cruel World' }, -- string
      { tag = 'b', value = { 0x00, 0x00, 0x10, 0xFF } } -- blob
    },
    -- connections
    {
      true,
      true, -- 2
      true, -- 3
      true, -- 4
      true, -- 5
      true, -- 6
      true, -- 7
      true, -- 8
      true, -- 9
      true  -- 10
    }
)
```

Control "Double-tap" ↑

Detect a "double-tap" on a control, with a certain maximum time passing between the taps.

```
local delay = 200 -- the maximum elapsed time between taps
local last = 0

function onValueChanged()
  if(not self.values.touch) then
    local now = getMillis()
    if(now - last < delay) then
      print('double tap!')
      last = 0
    else
      last = now
    end
  end
end
```

Send a Periodic Message ↑

Repeatedly send an OSC message, in this example once every second.

```
local delay = 1000 -- every 1000ms = 1s
local last = 0

function update()
  local now = getMillis()
  if(now - last > delay) then
    last = now
    sendOSC('/ping')
  end
end
```

Send Accelerometer Sensor Data ↑

Read data from the host device's accelerometer sensor (if available) and send as OSC message.

For more information see the documentation for the `getAccelerometer` utility function.

```
function update()
  local values = getAccelerometer()
  sendOSC('/accxyz', table.unpack(values))
end
```

Snap Fader to Grid ↑

Snap a fader's values to the configured grid line interval. Works for any control with a value of type FLOAT and a grid steps property.

```
function onValueChanged(key)
  if(key == 'touch') then
    local steps = self.gridSteps - 1;
    self.values[key] =
      math.floor(steps * self.values[key] + .5) / steps
  end
end
```

Reset Control Value ↑

Resets a control's value to its default value.

```
function resetValueToDefault(control, valueName)
  control.values[valueName] = control:getValueField(valueName, ValueField.DEFAULT);
end

-- example uses

resetValueToDefault(faderControl, 'x')
resetValueToDefault(pagerControl, 'page')
```


Copyright © 2026 Hexler Limited. All rights reserved. v1.0.2155