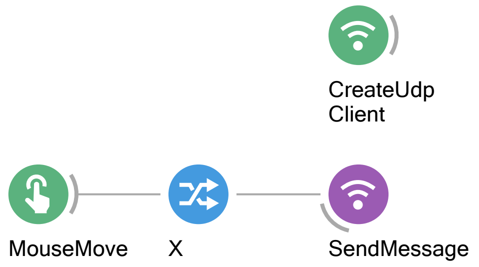
Networking

Bonsai includes support for Open Sound Control (OSC), a flexible networking protocol for low-latency communication between different processes, potentially running in different devices over the network. The next exercises will show you how to leverage these primitives for connecting two Bonsai processes exchanging a variety of data. The final optional exercise shows how to leverage the OSC protocol to interface a Python script with a Bonsai workflow.

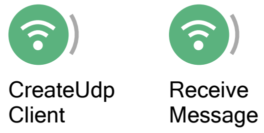
Exercise 1: Peer-to-peer UDP communication

We will start by implementing a direct peer-to-peer communication link between two processes on the same machine. This will allow us to send data between two known nodes in the network, or two independent Bonsai processes.



- Setup the above workflow.
- Set the `Name` property of the `CreateUdpClient` source to `Emitter`.
- Set the `RemotePort` to 2342.
- Set the `Connection` property of the `SendMessage` sink to `Emitter`.

Open a new Bonsai window and setup the following workflow:



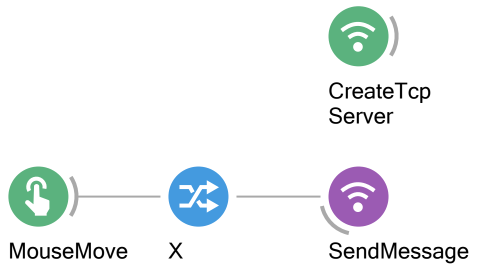
- Set the `Name` property of the `CreateUdpClient` source to `Receiver`.
- Set the `Port` property to 2342.
- Set the `Connection` property of the `ReceiveMessage` source to `Receiver`.
- Run the workflow and visualize the output of the `ReceiveMessage` source. Note the characters displayed in the `TypeTag`. Now change the `TypeTag` property of the `ReceiveMessage` source to `i`. This will make the source interpret the contents of the OSC message as a 32-bit integer. You can string multiple characters to describe complex messages.

complex messages.

- If you have access to two computers over a shared network, you can try to setup one of them to be the `Emitter` and the other to be the `Receiver`. In this case, make sure to set the `RemoteHostName` property of the `Emitter` to match the IP address of the receiver computer.

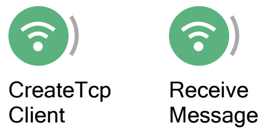
Exercise 2: Client/Server TCP communication

Next we will implement a responsive TCP server with support to accept multiple connections. This will allow us to share data between multiple unknown nodes in the network, where each receiver node just needs to know the IP address of the server and establish a connection to the data stream.



- Setup the above workflow (identical to the previous exercise but using `CreateTcpServer`).
- Set the `Name` property of the `CreateTcpServer` source to `Emitter`.
- Set the `Port` property to 2342.
- Set the `Connection` property of the `SendMessage` sink to `Emitter`.

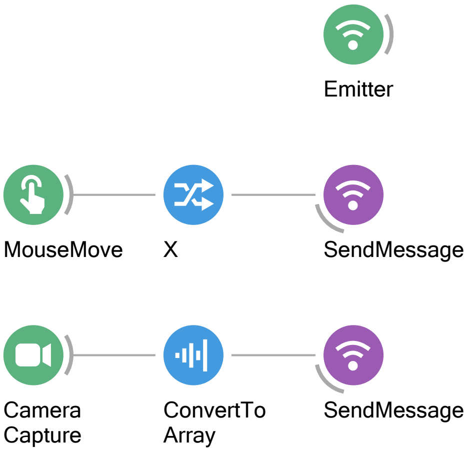
Open a new Bonsai window and setup the following workflow:



- Set the `Name` property of the `CreateTcpClient` source to `Receiver`.
- Set the `Port` property to 2342.
- Set the `Connection` property of the `ReceiveMessage` source to `Receiver`.
- Run the workflow and visualize the output of the `ReceiveMessage` source, optionally setting the `TypeTag` property to `i`. Try opening multiple copies of the receiver workflow and running them simultaneously. Verify that data is streamed to all instances successfully.
- If you have access to two or more computers over a shared network, you can try to set up multiple remote data listeners. In this case, make sure to set the `HostName` property of the `Receiver` node to match the IP address of the receiver computer.

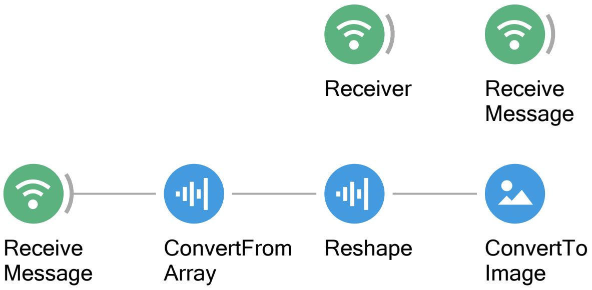
Exercise 3: Streaming image data

It is possible to share multiple data streams of different types simultaneously through a single OSC connection. To do this, we need to specify different OSC addresses for our messages to allow clients to subscribe to the independent streams.



- Start from the previous emitter workflow.
- Set the `Address` property of the `SendMessage` sink to `/cursor`.
- Add a `CameraCapture` source.
- Add a `ConvertToArray` transform to convert the image into an array of bytes.
- Add a new `SendMessage` node with the `Address` property set to `/image`.
- Ensure the `Connection` property of the new node is set to `Emitter`.

Open a new Bonsai window and setup the following workflow:



- Start from the previous receiver workflow.
- Set the `Address` property of the `ReceiveMessage` source to `/cursor`.
- Add a new `ReceiveMessage` source with the `Address` property set to `/image`.
- Run the emitter workflow and the receiver workflow and verify that you can receive both data streams.
- Set the `TypeTag` property on the new `ReceiveMessage` node to `b` for byte array.
- Add a `ConvertFromArray` transform following the `ReceiveMessage` source.
- Add a `Reshape` transform.
- Set the `Channels` property to 3 (color image) and the `Rows` property to 480 (or your camera image height).
- Add a `ConvertToImage` transform to interpret the resulting buffer as an image.
- Run both the emitter and the receiver workflow and verify you can successfully receive and decode both data streams. If you have access to two or more computers over a shared network, you can try to set up multiple remote data listeners, each

over a shared network, you can try to set up multiple remote data listeners, each listening to one or both data streams.

Visual Reactive Programming

A course on Visual Reactive Programming using Bonsai, developed by NeuroGEARS, Ltd.

neuro gears

neurogears

bonsai_rx

This website was prepared and developed for the Sainsbury Wellcome Centre, University College London.

Sainsbury Wellcome Centre