# Processing Serial Port routines

## Quick Start Guide

Install processing

<https://processing.org>

Aside installation of Processing IDE (3XX or 4XX) one needs to add G4P library to the Processing IDE, accessed by menu item **Sketch>>Import Library>>Manage Libraries** and search for “**G4P**”, click the library name to select then press install button.

All other dependencies are either default processing libraries or default java libraries.

I will sort out a Git Repository on SharePoint as it will be safer to version control the source, but for the time being simply unzip the project were ever you choose on your HDD.

Launch Processing IDE.

* Open the project from your hard drive.
* Press run button on the IDE and the application should run.

You will be presented with three buttons.

* **Exit** is logical. Closes all comm ports and exits.
* **Open** launches a new window with selection options for the comm ports. If there are no comm ports listed on this screen it means you have no recognizable comm ports on the system.
* **Send** just send hard coded string data out the selected serial port.
* **File Open** Opens a file to save RX data to, file name is from the functions screen. File gets saved to **data** directory of the project.

If you open a comm port, the port will remain open for the whole time the application is running until you exit from the application, or until you close it via code, or by opening the serial port window and pressing close button.

Minimum code to launch.

***serial\_gui new\_serial;***

*void setup()*

*{*

*size(500,300);*

***new\_serial = new serial\_gui(this);***

*}*

*void draw()*

*{*

*}*

But typically, one would launch serial\_gui from a button action.

*import g4p\_controls.G4P;*

*import g4p\_controls.GButton;*

*GButton btn\_open;*

***serial\_gui new\_serial;***

*void setup()*

*{*

*size(500,300);*

*btn\_open = new GButton(this, 10, 260, 65, 30);*

*btn\_open.setText("Open");*

*btn\_open.addEventHandler(this, "btn\_open\_click");*

*}*

*void draw()*

*{*

*}*

*public void btn\_open\_click(GButton source, GEvent event)*

*{*

*if(source == btn\_open && event == GEvent.CLICKED)*

*{*

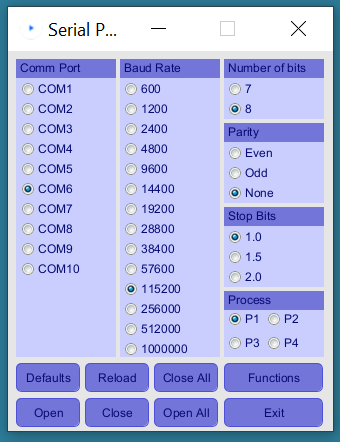
***new\_serial = new serial\_gui(this);***

*}*

*}*

The example project uses the button approach.

## Details of Serial Parameter screen



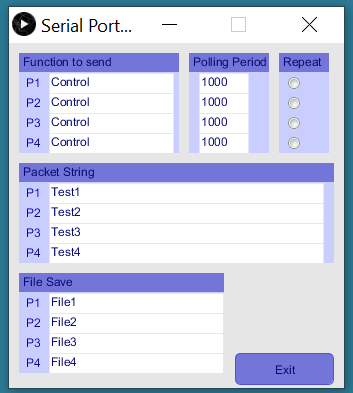
Most of the parameters are logical so won’t go into them.

**Process** panel intent is to allow up to 4 separate comm ports to be under separate software control.

Buttons

* **Defaults** Loads hard coded default parameters for all processes as a starting point.
* **Reload** Re-Loads all the process from the hard drive (if you want to get back to what you had at the start before you started modifying things)
* **Close All** Closes all the ports (Process 1>>4)
* **Open All** Opens all the ports (Process1>>4)
* **Open** Opens the comm port for the specific selected process with parameters as detailed on the screen.
* **Close** the comm port for the specific process selected.
* **Exit** Save the parameters for all processes to comms.dat file and leaves any ports open in their open state for use elsewhere in your code.
* **Functions** as below.

## Details of Function Screen



**Function to send** is the actual JAVA/Processing function one uses to send a specific processes command. Means you can have tailored functions to do specific things.

**Packet String** is the string that actually gets sent ("**Test1>>Test4**"), and it is sent by the **Function to Send**definition (**"Control"**function in this case). Control is the default function for polled events.

**Polling Period**time is milli seconds and Repeat option is the fact that it repeats, aka polling.

**File Save** is used for file name when current process file is opened for “appending”. File class allows other options but only append implemented in launch.pde.

## RX Data Access

To Access received data following is implemented in **launch.pde**

This function calls the function in the “**serial\_ports**.**pde**” class relevant to the opened port.

void serialEvent(Serial p)

{

new\_serial.specific\_process[associated\_process].serialEvent(p);

String inString = new\_serial.specific\_process[associated\_process].strarray\_rx[new\_serial.specific\_process[associated\_process].current\_rx\_pointer];

println("Serial Event :- " + inString + " Port :- " + p);

}

## Comment on RX Data

Current code is setup to wait for a Line Feed “lf” aka “0x10” to fire a Serial port event to receive data.

Based on this

myPort.bufferUntil(byte(10));

If your data stream has some other terminating character you will need to update this. Can be updated in the comms.dat file or programmatically in launch.pde with

new\_serial.specific\_process[associated\_process].setBuffer\_char(byte(10));

or something other than 10. Eg 13 might be valid, aka carriage return.

There is another option, that being wait till X number of chars have been received before firing an event, but that is not yet fully implemented.

## To Be Done

Tidy up abstract.pde

Tidy up file save and implement file close. Although file close is done after each write operation so not high priority.