Octagon Cheatsheet

This is a basic cheat sheet for Octagon with all of the basics.

Contents

Commands & Instructions **3**
- Pushers -> 3
- One Liners 3

Program Structure **3**
- Special Tasks 3
- **Actions, Tasks and Code Blocks** **4**
  - Actions 4
  - Tasks 4
  - Code Blocks 4
- **User Tasks** **4**

Variables **4**
- Variable Names 4
- Declaring Variables 4
- Setting Variables 5
- Variable Scoping 5
  - Global 5
  - Local 5
- Task Output Argument Variables 5
- Arrays 5
- Exiting 5

Displaying Information **6**

Repeating Things **6**
- repeat 6
  - Stopping Early 6

Making Decisions **6**
- Tests 6
- any 6
  - any Code Blocks 7
- all 7
  - all code blocks 7
- Using Else 8
- Combining any and all Statements 8
- Ants 8

Maths **9**
- General Math Commands 9
- Trigonometry 9
- Misc Maths Commands 9

Drawing and Graphics **10**
- Text Commands 10
  - ptl Put Text Line 10
  - etl Edit Text Line 10
- Drawing Commands 11
  - pclear Clear the screen 11
  - pp Put pixel 11
  - pvl Put Vertical Line 11
  - phl Put Horizontal Line 11
  - pl Put Line 11
  - pal Put Angular Line 11
  - psq Put Square 12
  - prect Put Rectangle 12
  - pcr Put Circle 12
  - parc Put Arc 12
  - psl Put Slider 12
- Gauges 13
  - pg Put Gauge 13
- **Graphs** **13**
  - plot Plot a Graph 13
  - Plot Command Tags 14
- Buttons 14
  - pb Put Button 14
- Audio 15

play Play Audio File 15
Video 15
play Play MP4 Video File 15
Images 15
play Display an Image File 15

RC Servos 16
Connecting RC servos to JackBord 16
RC Servo Commands 16
svp Set servo position 16
svs Sweep the servo 16
Example RC Servo Program 16

Smart RGB LEDs 17
Connecting Smart LEDs to the JackBord 17
Smart LED Commands 17
sledn Set number of smart LEDs 17
sled Set smart LED 17
Sled Command Colors 18

Buttons 18
Connecting a Button 18
Button Commands 18
btp Button Press 18
btr Button Release 18
lb List Buttons 18
btg Get Buttons Current State 19
lbc List Button Commands 19
btd Set Button Debounce Delay in MS 19
sbce Set brain button debounce delay 19
rstb Reset Buttons 19

JackBord Address Book 19
sabe Save Address Book Entry 19
labe List Address Book Entries 19
rabe Remove an address book entry 19

Sharing Between JackBords 19
Access Control 19
jallow Grant Access 20
jremove Remove Access 20
Listing JackBords With Access 20
Sending Remote Commands 20
src Send Remote Command 20
Running Remote Octagon Tasks 21
Setting Remote Variables 21
Synchronising Data 21
sync Synchronise 21
Synchronising Local Variables 22
Synchronising to Remote Variables 22
ls List Syncs 22
delsync Removing Sync 22

Commands & Instructions

Most of the JackBord commands can also be used in programs as instructions.

Pushers ->

1. Signal the end of something
The pusher means the end of something has been reached.
Eg any /temp > 10 -> heater_off

The pusher indicates the end of the any statements test and the start of the commands it needs to run.

2. Pushing data from one place to another
In this case we are telling Octagon to push some data from one place to another. Eg from a command like add to a variable like /total.

```
add 4 4  -> /total
```

No = in Octagon

In Octagon: temp = get_temp()
is

```
get_temp -> /temp
```

Or to set the value of a variable use:

```
/temp 12.3
```

One Liners

One or more commands or instructions on one line after a -> pusher. eg

```
repeat 10 -> print “Hi 10 times!”
```

or

```
any /temp > 28 -> fan_on
```

Program Structure

The counter program example below illustrates the basic structure of an Octagon program.

```
---
   Counter Program
         Displays the numbers 1 to 10.
---

prog_vars =
   d/count 0
prog_vars.

prog_start =
   print "Program Start"
prog_start.

prog_loop =
   -- main program loop
   print "Count /count"
   inc /count
   any /count >= 10 -> exitprog

prog_loop.

prog_stop =
   print "Bye"
Prog_stop.
```

Special Tasks

All of these tasks are special Octagon tasks each with a specific purpose. Every program must have these tasks and they are not allowed to be empty.

--- Prog Name ---	**Program Name & Description** The program's name must be on the first line, followed by a brief description of what it does.
prog_vars = prog_vars.	This task is where the variables (places where your program stores stuff) are declared. ust appear above it.
prog_start = prog_start.	This task holds the commands which need to run when your program first starts. It runs once only, when your program is run.
prog_loop = prog_loop.	This is where your program will spend the bulk of its time. The code in this task is run in an endless loop until the program stops.
prog_stop = prog_stop.	This task holds all the stuff that needs to be done before the program finally terminates. And lastly the prog_stop task MUST be at the very bottom of your program and any user tasks above it.

Actions, Tasks and Code Blocks

Actions

One, or more lines of code whose execution is conditional upon the result of a test. If it always runs, it is not an action. The action in the example below is shown in bold.

```
any /home_goals < 10 ->
        print “Hurry up we need goals!”
        print “Here are 10 more...”
        sum /home_goals 10 -> /home_goals
        print “I’m not cheating...”
enda.
```

Tasks

A task is a collection of code lines that do a specific thing and can be called from other parts of the program. Eg the get say_hi task below:

```
say_hi =
        Print “Hir from Octagon”
say_hi.
```

Code Blocks

Are one or more lines of code that’s not an action or task. For example, the code inside a repeat statement is a code block. In the examples below, the code blocks are in bold.

```
repeat 5 ->
    print “Dog count is /dogs
    inc /dogs
repeat.
 any /dogs > 100 ->
        print “We have too many dogs”
        print “Get more cats”
        inc /cat
enda.
```

User Tasks

User tasks are tasks created by you. The task can then be called from your program just as you would any other command.

```
do_moonwalk =
        code to make the robot
        do the moonwalk dance
        goes in here.
do_moonwalk.
```

The user task above is called **do_moonwalk** and is called like this:

```
prog_loop =
        do_moonwalk
        do_moonwalk
prog_loop.
```

Variables

Variables are a way of storing information in your program. This information usually changes as the program runs, which is why they are called variables.
Variables can hold integer numbers, floating point numbers, or text.

Variable Names

The variable name must be prefixed by the / character which tells Octagon its a variable.

Declaring Variables

The format is:

d/name value

Variables must be declared before use. Use the **d/** command followed by the variable's name and an initial value to create a new variable.

Here are some examples:

Declaration	**Type**
d/count 0	integer
d/threshold -100	integer
d/height 0.0	floating point
d/threshold 12.3	floating point
d/name “”	empty text
d/name “Cat dog”	text

Setting the initial variable value: tells Octagon the variable type and it sets the initial value.

Setting Variables

Set the value of a variable by typing its name followed by the new value e.g.

/launch_rocket 100
/air_temperature 25.8
/full_name “Fred Dag”

Or set a variable with another variable.
/temp /air_temp

Variable Scoping

Variable scoping refers to where in a program a given variable may be used. We have global and local variables.

Global

A **global** variable is one that may be used anywhere in the program. All variables declared in the **prog_vars** task are global.

Local

A **local** variable is declared inside a task other than the prog_vars task. They only need to be used in the task they were declared in and are not visible to the rest of the program. This is called scoping and refers to the visibility of a programs variables.

Task Output Argument Variables

These are an exception to both rules. They are declared in a task, see the section on tasks, and make the output of the task available elsewhere in the program. But may only be modified from within the task. This enables others to read them, but they cannot change them.

Arrays

To declare an array use this format:
d/array_name[index] default value

/array_name is the arrays name, just like variables.

[index] is the number of entries we wish to store. The index always starts at 1 and must be a positive integer number. Exceeding this value will result in an error.

default value is the same as that of a normal variable and specifies the type of information the array will hold.

Array Examples:

d/car_count[8] 0 integer
d/soup_temps[10] 0.0 float
d/address[10] “” text

These create arrays of the stated data types and sizes, in the [] brackets.

Exiting

These are the various ways in which you can exit a running program.

exitprog
This command tells Octagon that we want to end the program and exit. Octagon will wait until it next reaches the end of the prog_loop task and instead of going back to the top, it will jump to the prog_stop task and run it. Once prog_stop has completed the program will exit.

exitnow
This command tells Octagon we want to end the program on the line the exitnow command is on and jump straight to the prog_stop task. Unlike the exitprog command the lines below the exitnow command will not be run. Once prog_stop has completed the program will exit.

hardstop
This is the most direct way to stop a program. When run, the hardstop command will stop the program dead in it’s tracks no matter where it is. The prog_stop task is NOT run.

Displaying Information

The **print** command is the main method used to display information in Octagon. The format is:

```
print “text to display”
```

The text to display must be in **“”** quotes which can include variables in the text. The maximum length of the text you can use with the print command is about 50 characters. To save space you can leave out the print command and just use the text in “” quotes, so these two lines do the same thing:

```
print “Hi from Octagon”
“Hi from Octagon”
```

Display a Variables Value

You can include a variable in the text to be displayed and Octagon will display the value of the variable.

```
print “Air temp /air_temp deg C”
```

This displays the air temperature in deg C.

Repeating Things

There are two main ways of repeating things in Octagon. The first is the repeat command, which will run its code block a specified number of times. The second method is called recursion, in which a task calls itself a specified number of times from itself. This allows the task to loop and repeat its code as many times as required.

repeat

The repeat command lets you repeat a one liner or a code block a set number of times. Format:

```
repeat reps -> One liner
```

or

```
repeat reps ->
        Code block
repeat.
```

reps is the number of times to run the code.

In this example is a repeat with a code block that displays a rocket countdown timer and then reduces the launch time by 1. It does this 10 times until the launch time reaches 0.

```
repeat 10 ->
        print “Rocket launch in /launch_rocket seconds”
        dec /launch_rocket
 Repeat.
```

Stopping Early

```
repeat 10 ->
        inc /count
        print “count /count”
        skipout
 repeat.
```

Making Decisions

In Octagon there are two main ways in which you can make decisions based upon the outcome of one or more tests. These are the **any** and **all** commands.

Tests

A test compares two values using a condition, to determine if the test is true or false. Tests take the form:

value1 condition value2

Where:

value1 and **value2** are the numeric values being compared or tested.

condition is the means of comparison and may be one of:

```
< <= = >= > !=
```

any

For the any statement to be true any **one** of the tests it contains has to be true. In this case the true action will be run. If none of them are true the false action will be run.

The one liner format of the any statement is:

any test/s → True action

where:

test/s is one or more tests to be completed by the any statement each time it’s run.

True action this is the one liner run if one or more of the tests is true.

Note that because this is a one liner there is only a true action. There is no way of specifying a false action. Also the action commands must be on the same line as the any. If

you want to have more commands in the action you can use the code block version of the any statement.

Examples:
1. Exit when /count reaches 100:
any /count = 100 -> exitprog

2. Exit when /count is < 10 OR
/temp > 5:
any /count < 10 /temp > 5 -> exitprog

3. Turn light on when /light <= 25:
any /light <= 25 -> light_on

any Code Blocks

This version does the same as the one liner version except the true action commands are in a code block on the next line after the any statement. Instead of being all on one line.

The format is:

any test/s →
True code block
enda.

where:
test/s is one or more tests to be done.
True code block runs if any one of the tests is true.
enda. This signals the end of the any statement.

Example:

In this example we are using an any statement to monitor the temperature of a glasshouse. We have a heater some vents and a fan which need to be controlled based upon the measured
/air_temp.
– Too cold
any /air_temp <= 20 ->

close_vents
fan_off
heater_on
enda.

This is the too cold case and when the /air_temp falls below 20C many things need to happen: close the glasshouse vents, turn off the extractor fan and turn on the heater. These are the commands in the code block of the any statement.

all

For the all instruction to be true ALL of the tests must be true. If any one of the tests is false the all statement will be false. The nature of the all instruction implies that there should be more than one test otherwise you could just use any instead.

The one liner format of the all statement is:

all test1 test2 testn -> **True action**

where:
test is two or more tests to be completed by the all statement each time it's run.
True action this is the one liner run if ALL of the tests are true.

Examples:
1. Exit when /count reaches 100 and
/door = 1:
all /count >= 100 /door = 1 -> exitprog

2. Exit when /count reaches < 10 AND
/temp > 5:
all /count < 10 /temp > 5 -> exitprog

3. Turn light on when /light <= 25 AND
/dark is >= 100:
all /light <= 25 /dark >= 100 -> light_on

As you can see for the all statement to be true all of it's tests must be true. If any test is false the all will be false too.

all code blocks

This version does the same as the one liner version except the true action commands are in a code block on the next line after the all statement. Instead of being all on one line.
The format is:

all tests →
True code block
enda.

where:
tests is two or more tests to be completed by the all statement each time it's run.
True code block this code block is run if ALL of the tests are true.
enda. This signals the end of the all statement.

Example
In this example we are controlling a large cutter and need to make sure we only do a cut when the operator presses the

cut button and a safety button, ensuring their hands are away from the blade. We have the variables:

/cut and /hand_ok

These must BOTH be 1 before we can lower the blade to do the cut. The all instruction to do this is shown below.

```
all /cut = 1  /hand_ok = 1 ->
        do_cut
enda.
```

This simple example ensures that the worker cannot cut themselves because both of their hands need to be pressing specific buttons before the cut can be made.

Using Else

By adding an else statement to the any we can specify the true and false actions for the same any. The format is:

```
any test/s ->
        True code block
  else
        False code block
  enda.
```

where:

test/s is one or more tests to be done.

True code block runs if any one of the tests is true.

else this separates the true and false code blocks.

False code block runs if ALL of the tests are false.

enda. This signals the end of the any statement.

Liquid Level Example using else:

This is a version of the previous liquid level example that uses a single any and an else statement:

```
– Low level, turn pump on
any  /liquid_level < 100 ->
        – Low level, pump on
        pump_on
else
        – FULL, turn pump off
        pump_off
enda.
```

The result is less code that does the same job.

Combining any and all Statements

You can use any and all statements together as shown in the example below. When you do this the leading any or all is important. When using any and all's together it helps to think of them as being tests in their own right, each with an overall result of true or false.

For example:

```
any /pin1 = 0 /pin1 = 1 all /temp >= 10 /temp <= 20 ->
```

Order is Important

The order in which the any and all statements occur is important.

Any followed by All

If an **any** is followed by an **all**, like this:

```
any test1 test2  all test3 test4 ->
```

the true action will run if test1or test2 is true or if tests 3 & 4 are true, or if they are all true.

All followed by Any:

If an **all** is followed by an **any**, like this:

```
all test1 test2  any test3 test4 ->
```

the true action will only run if test 1 & 2 are true and at least one of the tests in the any are also true.

To summarise, in the first case the true action can be run if a minimum of one test is true. In the second case at-least 3 of the tests must be true before the true actions will be run.

Ants

The ant statement is used when you have many small tests that need to be done. The format of ant is:

```
ant value -> test -> true one-liner
             test -> true one-liner
             test -> true one-liner
             test -> true one-liner
             test -> true one-liner
                  -> default one-liner
  ant.
```

The value is a variable we are going to test against. This must be a number or text string, but it must also be a variable and not a value like a number or text value. The ant will make its way down the list of tests and as soon as one is true it will run the associated one liner and exit the ant. If none of the tests ends up being true the default one liner at the bottom is run instead and the ant exits.

Example 1:

Talking Thermometer In this example get the /air_temp and use an ant to say the correct phrase based upon the temperature.

```
ant /inside_temp ->
        < 10 -> say "Its very cold"
        < 5 -> say "Its cold"
        = 0 -> say "it's freezing"
        < 10 -> say "It's cool"
```

```
        < 20 -> say “It’s warm”
        < 30 -> say “It’s hot”
        < 40 -> say “It’s boiling”
        > 40 -> say “Sensor is broken”
        -> say “NO Temperature” ant
enda.
```

The say task, that contains the ant, speaks the phrase based upon the value of the /inside_temp variable.

Example 2: Flashing Rainbow

In this example we have some LEDs on port A and an ant statement is used inside a repeat loop to randomly toggle the state of the port A pins

```
repeat /runs_to_do ->
        -- Get a random number
        rand 1 5 -> /rand_no
        "rand no /rand_no run no /runs_done"

        ant /rand_no ->
                = 1 -> tg a1
                = 2 -> tg a2
                = 3 -> tg a3
                = 4 -> tg a4
                 = 5 -> tg a5
                -> "out of range number /rand_no"
         ant.
repeat.
```

Maths

These are the various maths commands in Octagon.

General Math Commands

add nn nn.. -> /result
sub nn nn.. -> /result
mul nn nn.. -> /result
div nn nn.. -> /result
sqrt nnn -> /result
avg nn nn.. -> /result
min nn nn.. -> /result
max nn nn.. -> /result
rand min max -> /result
map min1 max1 min2 max2 value

Trigonometry

sin angle in radians
cos angle in radians
tan angle in radians
deg2rad deg -> /result
rad2deg rad -> /result
abs
tint

Misc Maths Commands

round float decimals
hash
pow base power ->
inc /var step
dec /var step

```
-- Maths Functions Program Example
prog_vars =
d/a 10 -- integer var set to 10
d/b 100 -- integer var set to 100
d/c -3.141 -- float var set to -3.141
d/result 0.0 -- float to hold total
prog_vars.
prog_start =
print "Maths Example Program no 1"
prog_start.
prog_loop =
-- Display system var /pi
print "Pi is /pi"
add /a /b /c -> /result -- Add some numbers
print "adding /a /b /c = /result"
-- Subtract some numbers
sub /a /b /c -> /result
print "subtracting /a /b /c = /result"
-- Multiply some numbers
mul /a /b /c -> /result
print "multiplying /a /b /c = /result"
-- Divide some numbers
div /a /b /c -> /result
print "dividing /a /b /c = /result"
-- Root of pi
sqrt /pi -> /result
print "root of pi is /result"
-- New Values
/a 25|/b 50|/c 100
-- Average of some numbers
ave /a /b /c -> /result
print "ave /a /b /c = /result"
-- Analyse some numbers
min /a /b /c -> /result
print "min of /a /b /c is /result"
max /a /b /c -> /result
```

```
print "max of /a /b /c is /result"
-- Hash of pi
hash /pi -> /result
-- Get a random number between 0 and 1000
rand 0 1000 -> /result
print "Random number /result"
print "Done exiting now!"
exitprog
prog_loop.
prog_stop
Print “Bye from Maths”
Prog_stop
```

Drawing and Graphics

These are the various drawing and graphics commands which are used on the Show page.

Text Commands

These commands let you display text on the Show page and also let you get text input from the user.

ptl Put Text Line

Displays some text at the specified x, y position.

Syntax

ptl x y tags "Text to show"

Arguments

x position to display the text.

y position to display the text.

tags The tags for the command.

"Text to show" The text to be displayed. This MUST be provided and enclosed in "" brackets.

Examples:

1. Display the message “Hello World” at position 100, 100.

ptl 100 100 “Hello World”

2. Display a message at 100,100 in size 45 courier font in the color red.

ptl 100 100 ^id=m1^tf=courier^ts=45^tc=red^ "This is in Courier font size 45"

---

etl Edit Text Line

Displays some text in an input prompt at the specified x, y position. The value can be edited and the new value will be used to update the target variable specified in the ^var= tag. The value of the target variable is updated in real time.

Syntax

etl x y tags "Text to show/edit"

Arguments

x position to display the text.

y position to display the text.

tags The tags for the command.

"Text to show/edit" The text to be edited. This MUST be provided and enclosed in "" brackets.

Examples:

1. Display the name contained on the /name variable and allow the user to edit it. Limit the maximum name length to 20 characters.

etl d d ^pmt=First Name^w=20^max=20^var=name^ "/name"

2. Display the current forward motor speed on channel 150 and show the value, the prompt name is Speed.

etl h j ^pmt=Speed^max=20^var=150^ "/150"

Drawing Commands

These are the graphical drawing commands in Octagon. Use these to draw things like lines, rectangles, circles etc.

pclear Clear the screen

---

pp Put pixel

pp x y tags

x position of the pixel

y position of the pixel

tags The tags for the command

Examples:

1. Put a pixel at 100, 100

pp 100 100

2. Put a red pixel 10px square at 200, 200

pp 200 200 ^lw=20^fc=red^

---

pvl Put Vertical Line

Displays a vertical line starting at x & y on the show page.

pvl x y height tags

x position to display the lines start.

y position to display the line start.

height the height of the line.

tags The tags for the command.

Examples:

1. Create a vertical red line at 150, 150 of height 250.

pvl 150 150 250 ^lc=red^

2. Create a vertical line line starting at 20, 50 of length 200 and width 10 with round ends.

pvl 20 50 200 ^lc=lime^lw=20^lcap=round^

---

phl Put Horizontal Line

Displays a horizontal line starting at x & y on the show page.

phl x y length tags

x position to display the lines start.

y position to display the line start.

length the length of the line.

tags The tags for the command.

Examples:

1. Create a horizontal red line starting at 150, 150 of length 250.

phl 150 150 250 ^lc=red^

2. Create a horizontal lime line starting at 20, 50 of length 200 and width 10 with round ends.

phl 20 50 200 ^lc=lime^lw=20^lcap=round^

---

pl Put Line

Displays a line starting at x,y and ending at x2, y2.

pl x y x2 y2 tags

x position of the lines start.

y position of the lines start.

x2 position of the lines end.

y2 position of the lines end.

tags The tags for the command.

Examples:

1. Create a line starting at 50,50 and ending at 200,200.

pl 50 50 200 200

2. Create an orange line from 150,150 to 600,300, make the line 5 pixels thick.

pl 150 150 600 300 ^lw=5^lc=orange^

---

pal Put Angular Line

Displays a line starting at x & y at the specified angle on the show page. The angle is 0 at the top and increases in a clockwise manner.

pvl x y length angle tags

x position to display the lines start.

y position to display the line start.

length the length of the line.

angle The angle of the line from the vertical rotating clockwise.

tags The tags for the command.

Examples:

1. Create a 90deg red line starting at 150, 150 of height 250.

pal 150 150 250 90 ^lc=red^

2. Create a 45degree line line starting at 20, 50 of length 200 and width 10 with round ends.

pal 20 50 200 90 ^lc=lime^lw=20^lcap=round^

---

psq Put Square

Displays a square at x & y with sides of the specified length.

psq x y length tags

x position of the squares top left corner.

y position of the squares top left corner.

length the length of the squares sides.

tags The tags for the command.

Examples:

1. Create a blue square with sides 50 pixels wide at location 100,100.

psq 100 100 50 ^fc=blue^

2. Create a pink square with rounded corners and a red edge.

psq 50 50 100 ^lw=5^lc=red^fc=pink^rc=15^

---

prect Put Rectangle

Displays a rectangle at x & y with sides of the specified length.

prect x y width height tags

x position of the top left corner.

y position of the top left corner.

width the rectangles width.

height of the rectangle.

tags The tags for the command.

Examples:

1. Create a blue rectangle at 100, 100 that is 500 pixels wide and 100 high.

prect 100 100 500 100 ^fc=blue^

2. Create a pink rectangle with rounded corners and a red edge.

prect 50 50 500 100 ^lw=5^lc=red^fc=pink^rc=15^

---

pcr Put Circle

Displays a circle centered at x,y.

pcr x y radius tags

x position of the circles center.

y position of the circles center.

radius of the circle.

tags The tags for the command.

Examples:

1. Create a circle centered at 200,200 with a radius of 100.

pcr 200 200 100

2. Create an orange circle at 300,300 with a red edge 5 pixels wide.

pcr 300 300 100 ^lw=5^fc=orange^lc=red^

---

parc Put Arc

Displays an arc between a start and end angle centered at x,y.

parc x y radius start end tags

x position of the arcs center.

y position of the arcs center.

radius of the arc.

start the angle in degrees at which the arc will start.

end the angle in degrees at which the arc will end.

tags The tags for the command.

Examples:

1. Create an arc at 200,200 with a start angle of 0 deg and an end angle of 45 deg.

parc 200 200 100 0 45

2. Create an arc at 200,200 with a start angle of 0 deg and an end angle of 180 deg. Set the line width to 10, the line color to orange and the fill color to red. Make the line ends round.

parc 200 200 100 0 180 ^lw=10^lc=orange^fc=red^lcap=round^

---

psl Put Slider

Displays a slider on the show page. Moving the slider updates the target variable when the slider stops.

psl x y width tags "Value"

x position to display the slider.

y position to display the slider.

width the width of the slider.

tags The tags for the command.

"Value" This is the value to set for the slider. It should be a number.

Examples:

1. Create a speed control slider that will update the channel 150, has a range from -100 to 100. Place the slider at 100,100 and make it 200 wide.

psl 100 100 200 ^id=150^var=150^min=-100^max=100^ "/150"

2. Create a slider to set the temperature for a variable called /temp. Make the slider 300 pixels wide and the background color yellow.

psl 100 200 300 ^id=temp^var=temp^fc=yellow^ "/temp"

psl Command Tags

Tag	Description
id	Slide ID Id for the slider. Eg ^id=left^
var	Target Variable This is the variable whose value will be updated by moving the slider. Eg ^var=speed^
min	Slider Minimum Value This the minimum value of the slider. Eg ^min=0^
max	Slider Maximum Value This is the maximum value of the slider. Eg ^max=100^
n	Slider Name The name of the slider, displayed on it. ^n=open^ ^n=close^
h	Slider Height in pixels The height of the slider. ^h=12^

Gauges

A gauge is used to display a value using an analog gauge or dial.

pg Put Gauge

Displays a gauge at the specified x, y point. The value is the measurement to be shown on the gauge.

pg x y value tags

x position to display the gauge.

y position to display the gauge.

value The value to be shown.

tags The tags for the command.

Examples:

1. Display a gauge with a range from -10 to +50 for air temperature and set the value to 26. Display at 50 100.

pg 50 100 26 ^min= -10^max=50^

2. Display a gauge at 10, 150 with a width of 400 pixels. Use the variable /speed to update the gauge.

pg 10 150 /speed ^w=400^n=Car Speed^min=-50^max=50^

Graphs

Use graphs to plot data from a sensor or equation.

plot Plot a Graph

Displays a graph of data on the screen.

plot x y tags

x position to display the graph.

y position to display the graph.

tags The tags for the command.

Examples:

1. Plot some readings from a counter.

We have a variable called /count that goes from 0 to 9. The program below will plot the value of the /count variable.

```
---
        Plot counter program
---
prog_vars =
   d/count 0
prog_vars.

prog_start =
   print "Program Start"
prog_start.

prog_loop =
   -- main program loop
   inc /count
   any /count >= 50 -> exitprog

   -- Plot the count
   plot 0 50 ^id=plot1^v1=/count ^
```

```
prog_loop.
prog_stop =
   print "Bye"
prog_stop.
```

2.0 Plot the /temp value at the point on the x axis set by the /x_pos variable.

```
plot 0 50 ^id=plot1^v1=/count ^
```

Plot Command Tags

Tag	Description
id	Id Eg ^id=name^
n	Plot Name The name displayed on the graph.
w	Graph Width The width of the graph in pixels.
h	Graph Height The height of the graph in pixels.
dsid	Data Set ID The ID of the dataset you wish to plot.
start	Start Record No The number of the record in the dataset you want to start the plot with. Ie the first record to be plotted.
stop	Stop Record No The number of the record in the dataset you want to end the plot with. Ie the last record to be plotted.
xp	The X-axis Plot Point This is the point along the x-axis where the values will be plotted on the Y-axis. It should be incremented by one for each new set of plot values.
v1	Trace 1 Plot Value This is the value to be plotted on y-axis number 1 at the point along the x-axis set by the xp= tag.
v2	Trace 2 Plot Value This is the value to be plotted on y-axis number 2 at the point along the x-axis set by the xp= tag.
v3	Trace 3 Plot Value This is the value to be plotted on y-axis number 3 at the point along the x-axis set by the xp= tag.

Buttons

pb Put Button

Displays a button on the show page. Pressing the button runs specified commands. Separate commands for the press and release events can be set.

pb x y tags

x position to display the button.

y position to display the button.

tags The tags for the command.

Examples:

1. Create a button to turn USER LED 1 on when pressed, display at 100, 100.

```
pb 100 100 ^p=l1 1^n=USER1 ON^
```

2. Create a button to turn USER LED 1 off when pressed, display at 100, 125.

```
pb 100 125 ^p=l1 0^n=USER1 OFF^
```

3. Create a button to turn USER LED 1 ON when pressed, and OFF when released, display at 100, 150.

```
pb 100 150 ^p=l1 1^r=l1 0^n=USER1 Toggle^
```

4. Create a button to run a program task called b_press when it’s pressed.

```
pb 100 150 ^p=b_press^n=Run Task^
b_press =
   “Button Presses”
b_press.
```

Audio

play Play Audio File

Play the specified MP3 audio file in the Show page.

play x y tags "audio_url"

x position

y position

tags The tags for the command.

"**audio_url**" The URL where the MP3 audio file is in quotes.

Notes:

1. The x & y values should be set to 0
2. In Chromebooks the audio will only play after the user has interacted with the page, this is a Google security feature. Simply add a button to the page and click it.

Example:

1. Play an mp3 file:

```
play 0 0 "https://example.com/MP3_audio.mp3"
```

Video

play Play MP4 Video File

Play the specified MP4 video file in the Show page.

play x y tags "video_url"

x position

y position

tags The tags for the command.

"**video_url**" The URL where the MP4 video file is in quotes.

Notes:

1. In Chromebooks the video will only play after the user has interacted with the page, this is a Google security feature. Simply add a button to the page and click it.

Example:

1. Play an mp4 file at 100 100:

```
play 100 100 "https://example.com/video.mp4"
```

Images

play Display an Image File

Display the specified image file in the Show page.

play x y tags "image_url"

x position

y position

tags The tags for the command.

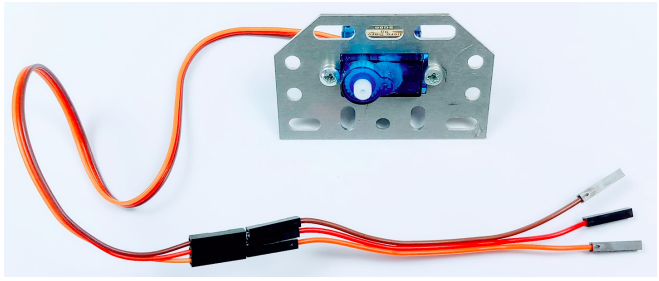
"**image_url**" The URL where the image file is in quotes.

Example:

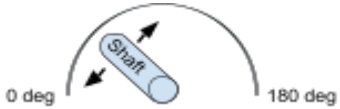
1. Display an image at 200 450:

```
play 200 450 "https://example.com/pic.jpg"
```

RC Servos

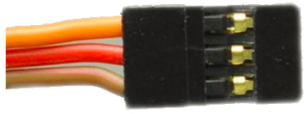


RC servos let you move a shaft through 180 degrees.



Connecting RC servos to JackBord

Orange/Other (signal)
Red wire (+) ~6V
Brown/Black wire (-)



RC servos come with three wires: signal, power and ground. The ground wire is almost always brown or black while the power supply wire is red and in the center. The signal wire is usually orange. **DANGER: DON'T MANUALLY TURN THE SERVOS SHAFT you will damage it.**

JackBord Servo Connections:

Servo	JackBord
Signal (Orange/other)	Port A, B or C
Power (Red)	+5V on POWER port
Ground (Brown/Black)	Grounds on POWER port

The JackBord needs female connections so use some male to female jumpers to connect the servo to the JackBord as shown below.

RC Servo Commands

Use these commands to control RC servos connected to the JackBord.

svp Set servo position

Set the servo on port #~ to position pos between 0 and 180 deg.

svp #~ angle

#~ port pin no eg a1, c4 etc

angle the shaft angle between 0 and 180 degrees

Examples:

1. Set the servo on a2 to 90deg

svp a2 90

2. Set the servo on b4 to 45deg

svp b4 45

svs Sweep the servo

Sweep the servo from 0deg to the sweep angle and back again with delay between steps.

svs #~ angle delay

#~ port pin no eg a1, c4 etc

angle the angle to sweep to from 0 deg

delay the delay in milliseconds between steps

Examples:

1. Make the servo on a1 sweep from 0deg to 90deg and back to 0 with a delay of 50ms for each 1 deg of sweep.

svs a1 90 50

Example RC Servo Program

```
---
        Example RC Servo Program
        This program has an RC servo connected to port pin A1 and will move the servo from 0 to 90 deg. When it reaches 90deg it will reset it back to 0. Each step will be 5 degrees.
---
prog_vars =
  d/count 0
  d/angle 0
prog_vars.

prog_start =
  pclear
  print "Program Start"
prog_start.

prog_loop =
  -- main program loop
  print "Count /count  a /angle"
  inc /count
```

```
    any /count >= 100 -> exitprog

    -- Set the servos angle
    svp a1 /angle

    -- Reset the angle back to 0 when it reaches 90deg
    any /angle < 90 ->
        inc /angle 5
    else
        /angle 0
    enda.
   -- wait 0.5secs
    d500
prog_loop.

prog_stop =
    print "Bye"
Prog_stop.
```

Smart RGB LEDs

Smart RGB LEDs are strips of smart LEDs that let you control each LED in the strip.

Connecting Smart LEDs to the JackBord

The smart LEDs have 3 pins:

LEFT	CENTER	RIGHT
GROUND	DATA	+5V

The DATA pin connects to the JackBords port E1.

JackBord Connections:

Smart LED	**JackBord**
Ground	Ground on POWER port
DATA	Port pin E1
+5V	+5V on POWER port

Smart LED Commands

These commands are for controlling the smart LEDs attached to the JackBord.

sledn Set number of smart LEDs

Sets the number of smart LEDs in the strip connected to the JackBord.

sledn number

number the number of LEDs on the strip

Example:

1. Set the number of smart LEDs connected to 8:

sledn 8

---

sled Set smart LED

Sets the color of the specified smart LED.

sled no color update

no The number of the LED we want to set. **99** = set all

color desired color, see the table

update Missing means update the LEDs now. 0 = Dont update them, instead use the **usled** command later to update them all in one go.

Examples:

1. Set the 2nd LED to red:

sled 2 3

2. Set the 10th LED to cyan:

sled 10 12

Sled Command Colors

The color to set the LED to, from the list of colors below:

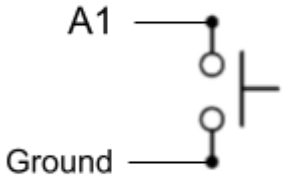
Color	Color Number	Red	Green	Blue
Off	0	0	0	0
Dim white	1	128	128	128
Bright white	2	255	255	255
Red	3	255	0	0
Orange red	4	255	69	0
Orange	5	255	165	0
Gold	6	255	215	0
Yellow	7	255	255	0
Green	8	0	128	0
Lime	9	0	255	0
Light green	10	144	238	144
Blue	11	0	0	255
Cyan	12	0	255	255
Navy	13	0	0	128
Magenta	14	255	0	255
Purple	15	128	0	128

Buttons

Buttons can be attached to the JackBords port A, B & C.

Connecting a Button

One half of the button goes to the respective port A, B or C pin and the other goes to ground.



Button Commands

btp Button Press

Use this command to set what a button will do when its pressed.

btp #~ command/s

#~ port pin no eg a1, c4 etc. Or u1 to u9 for drive page buttons.

command/s one or more commands

Examples:

1. Set a button on port pin A1 to turn the user led L1 on when pressed.

btp a1 l1 1

2. Set Button 3 on the drive page to set a servo on port A3 to 90 degrees.

btp u3 svp a3 90

---

btr Button Release

Use this command to set what a button will do when its released.

btr #~ command/s

#~ port pin no eg a1, c4 etc. Or u1 to u9 for drive page buttons.

command/s one or more commands

Examples:

1. Set a button on port pin A1 to turn the user led L1 off when released.

btr a1 l1 0

---

lb List Buttons

Displays a list of currently set buttons.

lb

Example

```
> lb
< Buttons
No    Chan  Name          Port
1     1     but a1        a1    CMD: l1 1
                              Release l1 0
```

---

btg Get Buttons Current State

Gets the state of a button, 0 = not pressed 1 = pressed.

btg

lbc List Button Commands

Lists the commands needed to program the currently set buttons. Use this to save a set of button commands.

lbc

---

btd Set Button Debounce Delay in MS

Set the debounce delay in MS for buttons.

btd delay

delay The debounce delay in milli seconds.

Example

1. Set the button debounce delay to 100 MS

btd 100

---

sbce Set brain button debounce delay

This sets the button debounce delay for buttons on ports A or B of a block 3 JackBord.

sbce delay

delay The debounce delay in milli seconds.

Example

1. Set the button debounce delay to 100 MS

sbce 100

---

rstb Reset Buttons

Reset all currently set buttons.

rstb

JackBord Address Book

The address book provides a way to store profile ID's of other JackBords you want to interact with.

sabe Save Address Book Entry

This will add a new entry into the address book.

sabe name profile-id

name A name for this entry in the address book

profile-id The profile id of the JackBord.

Example:

1. Add an address book entry for Jill whose profile id is 108f

sabe Jill 108f

---

labe List Address Book Entries

Displays a list of the address book entries.

labe

---

rabe Remove an address book entry

This removes the selected address book entry.

rabe entry no

entry no The Number of the address book entry from the labe command that you wish to remove.

Example:

1. Remove entry number 5:

rabe 5

---

cabe Clear Address Book

Delete all entries from the address book.

Sharing Between JackBords

This section deals with sharing control and information between JackBords. When you adopt a JackBord the profile that you use to adopt it has a profile id and this is used when granting access.

Access Control

Before another JackBord can control yours you have to grant it permission using the **jallow** command. To remove access use the **jremove** command.

---

jallow Grant Access

This command with grant access to your JackBord to another JackBord. Once granted the other JackBord will be able to control yours.

jallow profile-id

profile-id the profile id of the remote JackBord.

Note: You can find the profile id by using the **vs** command. It's listed towards the top.

Examples:

1. Grant a JackBord whose profile id is 108f access to your JackBord.

jallow 108f

2. Grant access to your JackBord to another JackBors whose profile id is 10D3 and add an entry in the address book under Sam.

jallow 10D3 Sam

From now on you can use Sam in any command where a profile-id is expected.

---

jremove Remove Access

This command with remove access to your JackBord it was previously granted.

jremove profile-id

profile-id the profile id of the remote JackBord.

Note: You can find the profile id by using the **vs** command. It's listed towards the top.

Examples:

1. Remove access from a JackBord whose profile id is 108f.

jremove 108f

Listing JackBords With Access

Use the **labe** list address book entries command to view all of the remote JackBords that have access to yours.

Sending Remote Commands

Normally when you run a command on your JackBord it is run locally. The **src** command, which stands for send remote command, lets you send a command to a remote JackBord. In other words that lets you run commands on another JackBord.

Before you can run commands on another JackBord the owner of that JackBord must have granted your JackBord access using the **jallow** command.

---

src Send Remote Command

Send a command to another JackBord.

src profile-id/name command/s

profile-id/name This is the profile ID of the remote JackBord we wish to send the command to. If there is an address book entry for the remote JackBord in your address book you can use that name instead of the profile ID.

command/s One or more commands to run on the remote JackBord. If there is more than one command use the pipe character | between the commands.

Examples:

1. Sam wishes to send commands to Jill's JackBord whose profile id is JI00. Sam's profile id is SA00.

Owner	Profile ID
Sam	SA00
Jill	JI00

First Jill needs to grant Sam access by running the jallow command on her JackBord like this:

jallow SA00

Now Sam can send commands to Jill's JackBord. Sam wants to control an RC servo on port A1 of Jill's JackBord. To set the servo to 90deg Sam enters the following src command on his JackBord:

src JI00 svp a1 90

This command sends the **svp a1 90** servo command to Jills JackBord and it runs it.

2. Sam does not want to have to remember Jills profile ID so he adds it to his address book with this command:

sabe jill JI00

From now on he can use jill instead of her profile id in commands. Thus the src command from the first example could be:

src jill svp a1 90

Running Remote Octagon Tasks

The src command can also be used to call tasks on a currently running program on a remote JackBord. Jill has a program running on her JackBord and one of the tasks is called **turn_light_on**. Sam wants to run that task from his JackBord when he detects that it's getting dark. To do this he would run this command:

src jill turn_light_on

This will cause Jills JackBord to run the task turn_light_on.

Note: For this to work the program containing the task must be running on Jills JackBord.

Setting Remote Variables

The src command can also be used to set the value of variables on a remote JackBord. For example there is a variable on Jill's JackBord called /air_temp and Sam wants to set it to 25 degrees Celsius. To do this Sam runs the command:

src jill /air_temp 25.0

This will set the value of air_temp on Jill's JackBord.

Note: When using the src command to set variables you cannot use a variable as the value to set, so for example this will not work:

src jill /air_temp /temp

This is because the src command will send the string /temp and not it's value.

Synchronising Data

The sync command lets you synchronise two variables or channels on a local JackBord or between JackBords.

sync Synchronise

Synchronise channels or variables locally or between JackBords.

sync source dest profile-id/name

source the channel number or variable name that is to be the data source

dest the channel number or variable name that is to be the destination

profile-id/name (optional) This is the optional profile ID of the remote JackBord we wish to work with.

Note: If you only wish to synchronise locally the profile id is not required.

Examples:

1. Sam has drive motors attached to his JackBord and wants to use a variable resistor connected to pin A1 to control the speed of his robot. The forward speed of the drive is controlled by channel 150, thus the command would be:

sync a1 150

This causes the current value for the variable resistor on A1 to be synchronised to channel 150, thus controlling the speed of the drive motors.

2. Using Sam and Jill from the previous examples:

Owner	Profile ID
Sam	SA00
Jill	JI00

Jill has attached the drive motors to her JackBord and wants to allow Sam to use his JackBord as a remote control for her driving robot.

On Jills JackBord run these commands:

```
jallow SA00 sam
```

On Sams JackBord run these commands:

```
sabe jill JI00
gvr a1 -100 100
sync a1 150 jill
```

These add an address book entry for Jills robot, set the variable resistor on pin A1 to produce a value between -100 and 100. And finally we synchronise the value of pin a1 to channel 150 on Jills JackBord.

Now when Sam turns the variable resistor Jills robot will move.

Synchronising Local Variables

The sync command can be used to synchronise variable values locally. This allows you to have a situation whereby when one variable changes a destination variable will be automatically updated with the new value.
If we have a Raspberry Pi Shake seismometer and we want to use the /rs_EHZ readings and store them in a variable called /quake_EHZ we would type:

sync /rs_EHZ /quake_EHZ

From now on any change in the Raspberry Pi Shake /rs_EHZ variable will be automatically synchronised over to the /quake_EHZ variable.

Synchronising to Remote Variables

In this case we have a local variable and we want to synchronise it over to a variable on a remote JackBord. To do this we use the sync command as normal except we add the address book name or profile ID of the remote JackBord we want to control.

sync /speed /speed jill

The above command will take the local /speed variables value and use it to set the same variable on Jill's JackBord.

Note: The /speed variable must exist on Jill's JackBord and be of the same data type.

ls List Syncs

Lists the active syncs.

ls

---

delsync Removing Sync

Remove a sync.

delsync no

no The number of the sync to remove from the **ls** sync list.