JavaScript Reference r2

This document describes in detail the JavaScript Interpreter feature. First of all, see chapter "5.3. JavaScript Interpreter" of the REFERENCE MANUAL.

Sample codes are described inside the frame.

Obsolete (removed) features

`pracq.midiOutCtrl()`	Use **`pracq.midiOut("Ctrl", ...)`** instead.
`pracq.midiOutPlay()`	Use **`pracq.midiOut("Play", ...)`** instead.
`pracq.bpm()`	Use **`pracq.generalProp("BPM", ...)`** instead.
`pracq.stepVel()`	Use **`pracq.stepProp(..., "Velocity", ...)`** instead.
`pracq.getMsCounter()`	Use **`pracq.getInfo("msCounter")`** instead.
`pracq.getTimeStr()`	Use **`pracq.getInfo("yyyy")`**,... instead.
`pracq.toHexStr()`	Use **`pracq.getInfo("toHex", ...)`** instead.
`pracq.isScaleNote()`	Use **`pracq.getInfo("isScaleNote")`** instead.
`pracq.version`	Use **`pracq.getInfo("Version")`** instead.

Event List

`E_STARTUP`

Occurs when PRACQ is launched.
This event will certainly occurs once, so use it to initialize.

`pracq.eventMsg`: 0

`E_TESTRUN`

Occurs when the Test Run button is pressed.
This event is always active regardless of **`pracq.regEvents`**.

`pracq.eventMsg`: Integer value that is inputted into the textbox next to the Test Run button.

`E_TIMER`

Occurs when the timer expires. (See **`pracq.setTimer()`**.)

`pracq.eventMsg`: Timer ID

E_PLAYBACK

Occurs when a playback starts or stops.

pracq.eventMsg: 1 (Start), 0 (Stop)

E_PLAYPOS1, E_PLAYPOS2, E_PLAYPOS3, E_PLAYPOS4

Occurs when the playback position moves.
These are for Track 1 - 4 respectively.
Do not register two or more of these at a time. Multiple registration won't work well.

pracq.eventMsg: Step No. (0 - 127)

E_PLAYPOS

Occurs when the playback position of each track moves. Use **pracq.playPos()** to get a step No. where the playback position is.

pracq.eventMsg: Bits that indicate track No whose playback position moves. For example, if the playback position of track 1 moves, 1(=0b0001) will be set. If the playback positions of track 1 and 3 move at the same time, 5(=0b0101) will be set.

E_FUNCPAD

Occurs when the function pad that is assigned to "JavaScript Event" is pressed or released.

pracq.eventMsg: Function pad No. (1 - 4) (Negative number when released.)

E_MIDIIN_CTRL

Occurs when a MIDI message is received from the controller device that is set as "JavaScript-Controlled" type. (Settings -> Controller -> Grid MIDI Controller)

pracq.eventMsg: Received MIDI message (The first 3 bytes only.)
(Use **pracq.getMsgArray(E_MIDIIN_CTRL)** for a whole message.)

E_MIDIIN_EXT

Occurs when a MIDI message is received from the external keyboard device. (Settings -> Controller -> MIDI In (External MIDI Keyboard))
When this event is registered, the original process for the external keyboard will be disabled.

pracq.eventMsg: Received MIDI message (The first 3 bytes only.)
(Use **pracq.getMsgArray(E_MIDIIN_EXT)** for a whole message.)

E_PCKEY_DN, E_PCKEY_UP

Occurs when the computer keyboard is pressed or released.
In order to use the computer keyboard, the PRACQ main window must be active.
When this event is registered, the original process for the computer keyboard will be disabled.

pracq.eventMsg: Key code of the pressed or released key.
(Use **pracq.getInfo("PCKeyName", pracq.eventMsg)** to convert a key code into a key name.)

PRACQ API (pracq class)

Utilities

pracq.log()

Outputs a string to the console.

```
pracq.log(object1, object2, ...)
```

object1, object2, ...: Objects to output
If the number of objects is 0, the console will be cleared.
If the number of objects is 1, outputs with a new line.
If the number of objects is 2 or more, outputs without a new line.
Return value: Outputted string.

```
pracq.log(); // Clear
pracq.log("Hello world!"); // Hello world! (with a new line)
pracq.log("Hello world!", ""); // Hello world! (without a new line)
pracq.log("The sum is ", 1 + 2, ".\n"); // The sum is 3.
```

pracq.regEvents

Registers events.

pracq.regEvents: Bits associated with events.

```
if(pracq.eventType == E_STARTUP){
    pracq.regEvents = E_PLAYBACK | E_MIDIIN_CTRL;
}
```

pracq.getInfo()

Returns various kinds of information. (See “Sequencer” section too.)

```
pracq.getInfo("msCounter")
```

Returns a milliseconds counter.

Return value: Number of milliseconds since PRACQ is launched.

```
pracq.log(pracq.getInfo("msCounter"); // e.g. 18281
```

pracq.getInfo("toHex", num)

Converts a number to a hexadecimal string.

num: 32-bit integer.

Return value: String representing **num** value in hexadecimal.

```
var str = pracq.getInfo("toHex", 123);
pracq.log(str); // 0x7b
pracq.log(parseInt(str) + 1); // 124
```

pracq.getInfo("PCKeyName", keyCode)

Regarding the computer keyboard, converts a key code into a key name.

keyCode: Key code that is got on the **E_PCKEY_DN** or **E_PCKEY_UP** event.

Return value: Key name

```
if(pracq.eventType == E_PCKEY_DN){
    pracq.log(pracq.getInfo("PCKeyName", pracq.eventMsg), " is pressed.\n");
}
```

pracq.getInfo("Version")

Returns an integer of PRACQ software version.

Return value: Integer (“XXYYZZ” in decimal means “vXX.YY.ZZ”.)

```
pracq.log(pracq.getInfo("Version")); // e.g. 806 (v0.8.6)
```

pracq.getInfo("yyyy")
pracq.getInfo("MM")
pracq.getInfo("dd")
pracq.getInfo("HH")
pracq.getInfo("mm")
pracq.getInfo("ss")
pracq.getInfo("fff")

Returns a string of current time.

Return value: From the top, Year, Month, Day, Hours, Minutes, Seconds and Milliseconds.

```
pracq.log(pracq.getInfo("HH") + ":" + pracq.getInfo("mm"));
```

pracq.showExecTime

Sets whether an execution time will be shown on the console.

pracq.showExecTime: 0 (Not shown), 1 (Shown)

```
if(pracq.eventType == E_STARTUP){
    pracq.showExecTime = 1; // An execution time will be shown.
}
```

pracq.staticInt

Holds a 32-bit integer value beyond the event.

pracq.staticInt: 32-bit integer.

```
if(pracq.eventType == E_STARTUP){
    pracq.staticInt = 123;
} else if(pracq.eventType == E_TESTRUN){
    pracq.log(pracq.staticInt); // 123
}
```

pracq.staticInts()

Expands **pracq.staticInt**.

```
pracq.staticInts(index)
pracq.staticInts(index, value)
```

index: 0 - 127.

value: 32-bit integer to set.

Return value: 32-bit integer to get.

```
if(pracq.eventType == E_STARTUP){
    pracq.staticInts(0, 123);
    pracq.staticInts(1, 456);
} else if(pracq.eventType == E_TESTRUN){
    pracq.log(pracq.staticInts(0) + pracq.staticInts(1)); // 579
}
```

pracq.staticIntArray()

Expands **pracq.staticInts()**.

```
pracq.staticIntArray(index)
pracq.staticIntArray(index, array)
```

index: 0 - 1023.

array: Array that contains integers to set.

Return value: Array that contains integers to get.

```
if(pracq.eventType == E_STARTUP){
    pracq.staticIntArray(0, [60, -1, 62]);
} else if(pracq.eventType == E_TESTRUN){
    var arr = pracq.staticIntArray(0);
    for(var i = 0; i < arr.length; i++){
        pracq.stepNotes(0, i, arr[i]);
    }
```

```
}
```

pracq.setTimer()

Sets a one-shot timer.

```
pracq.setTimer(timerID, milliseconds)
```

timerID: Timer ID (A 32-bit integer more than 0).
milliseconds: Interval in milliseconds (A 32-bit integer more than 0).
If **milliseconds** is 0, the timer that has been started is stopped.
Return value: Set interval in milliseconds.

```
if(pracq.eventType == E_STARTUP){
    pracq.regEvents = E_TIMER;
} else if(pracq.eventType == E_TESTRUN){
    // Press the Test Run button to start the timer.
    pracq.setTimer(1,1000);
} else if(pracq.eventType == E_TIMER){
    // Timer ID 1 has expired.
    pracq.log("Timer ID " + pracq.eventMsg + " has expired.");
}
```

pracq.getMsgArray()

Returns a specific event message as an array.

```
pracq.getMsgArray(event)
```

event: **E_MIDIIN_CTRL** or **E_MIDIIN_EXT**
Return value: Array that contains a event message.

Sequencer

pracq.stepNotes()

Sets/gets notes in the step.
If you want to make the step ON status, use **pracq.stepProp(trackNo, steoNo, "StepStatus", 2)**.

```
pracq.stepNotes(trackNo, stepNo)
pracq.stepNotes(trackNo, stepNo, note1, note2, ...)
pracq.stepNotes(trackNo, stepNo, notes)
```

trackNo: Track No. (0 - 3)
stepNo: Step No. (0 - 127)
note1, note2, ...: Note number (0 - 127) to set (up to **note6**).
To remove all notes in the step, pass -1 to **note1**.

notes: Array that contains up to 6 note numbers (0 - 127) to set.
Return value: Array that contains up to 6 note numbers (0 - 127) to get.

```
pracq.stepNotes(0, 0, [60, 64, 67]);
var notes = pracq.stepNotes(0, 0);
for(var i = 0; i < notes.length; i++){
    notes[i] = notes[i] + 1; // Transpose
}
pracq.stepNotes(0, 1, notes);
pracq.stepNotes(0, 0, -1); // All notes removed.
```

pracq.replaceStepNote()

Replaces note in the step.

```
pracq.replaceStepNote(trackNo, stepNo, oldNote, newNote)
```

trackNo: Track No. (0 - 3)
stepNo: Step No. (0 - 127)
oldNote: A note number (0 - 127) that is to be replaced by **newNote**
newNote: A note number (0 - 127) that replaces **oldNote**
To add **newNote**, pass -1 to **oldNote**.
To remove **oldNote**, pass -1 to **newNote**.

```
pracq.stepNotes(0, 0, [60, 64, 67]);
pracq.replaceStepNote(0, 0, -1, 71); // Add
pracq.replaceStepNote(0, 0, 67, 55); // Replace
var notes = pracq.stepNotes(0, 0);
for(var i = 0; i < notes.length; i++){
    pracq.log(notes[i], " "); // 55 60 64 71 (Automatically sorted)
}
```

pracq.playback()

Starts/stops the playback.

```
pracq.playback()
pracq.playback(isToggle)
```

isToggle: 1 (Toggle between the playing state and the stopped state.)
Return value: 0 (Stopped), 1 (Playing)

```
if(pracq.eventType == E_TESTRUN){
    // The Test Run button works as a play button.
    pracq.log((pracq.playback(1) == 0) ? "Stopped" : "Playing");
}
```

pracq.playPos()

Returns playback position of each track.

```
pracq.playPos()
```

Return value: Array that contains playback positions (0 - 127) of track 1 to 4 in order.

```
if(pracq.eventType == E_STARTUP){
    pracq.regEvents = E_PLAYPOS;
} else if(pracq.eventType == E_PLAYPOS){
    for(var i = 0; i < 4; i++){
        if((pracq.eventMsg & (1 << i)) != 0){
            pracq.log("Tr", i+1, ":", pracq.playPos()[i]+1, " ");
        }
    }
    pracq.log("");
}
```

pracq.getInfo()

Returns various kinds of information. (See “Utilities” section too.)

pracq.getInfo("isScaleNote", note)

Returns whether a specific note is in the current scale.

note: Note number (0 - 127).

Return value: 1 (Scale note), 0 (Non-scale note)

```
for(var i = 60; i < 72; i++){
    pracq.log(pracq.getInfo("isScaleNote", i), ""); // e.g. 101011010101
}
```

pracq.getInfo("ScaleName")

Returns an array of the scale name list.

See the description of **pracq.generalProp("Scale")**.

pracq.getInfo("RootName")

Returns an array of the root name list.

See the description of **pracq.generalProp("Root")**.

pracq.generalProp()

Sets/gets general properties.

pracq.generalProp("BPM")
pracq.generalProp("BPM", value)

Sets/gets a BPM (tempo) value.

value: BPM value to set (1 - 399).

Return value: BPM value to get (1 - 399).

```
var bpm = pracq.generalProp("BPM");
if(bpm < 399){
    pracq.generalProp("BPM", ++bpm);  // Increase the BPM.
}
```

```
pracq.generalProp("Scale")
pracq.generalProp("Scale", number)
```

Sets/gets a scale number.

Use **pracq.getInfo("ScaleName")** to convert a scale number into a scale name.

number: Scale number to set (0 - 25).

Return value: Scale number to get (0 - 25).

```
pracq.generalProp("Scale", 2); // "2" means Minor scale.
pracq.log(pracq.getInfo("ScaleName")[pracq.generalProp("Scale")]); // Minor
```

```
pracq.generalProp("Root")
pracq.generalProp("Root", number)
```

Sets/gets a root number.

Use **pracq.getInfo("RootName")** to convert a root number into a root name.

number: Root number to set (0 - 11).

Return value: Root number to get (0 - 11).

```
pracq.generalProp("Root", 2); // "2" means D.
pracq.log(pracq.getInfo("RootName")[pracq.generalProp("Root")]); // D
```

pracq.trackProp()

Sets/gets track properties.

```
pracq.trackProp(trackNo, "StepTime")
pracq.trackProp(trackNo, "StepTime", number)
```

Sets/gets a step time number.

Step time numbers (0 - 5) mean 1, 1/2, 1/4, 1/8, 1/16, 1/32 in order.

trackNo: Track No. (0 - 3)

number: Step time number to set (0 - 5).

Return value: Step time number to get (0 - 5).

```
pracq.trackProp(0, "StepTime", 2); // "2" means 1/4.
pracq.log(pracq.trackProp(0, "StepTime"));
```

```
pracq.trackProp(trackNo, "Tripret")
pracq.trackProp(trackNo, "Tripret", status)
```

Sets/gets a status of step time tripret.

trackNo: Track No. (0 - 3)

status: Tripret status to set (0 (Off), 1 (On)).

Return value: Tripret status to get (0 (Off), 1 (On)).

```
pracq.trackProp(0, "Tripret", 1); // "1" means On.
pracq.log(pracq.trackProp(0, "Tripret"));
```

```
pracq.trackProp(trackNo, "Blocks")
pracq.trackProp(trackNo, "Blocks", number)
```

Sets/gets a number of blocks.

trackNo: Track No. (0 - 3)

number: Number of blocks to set (1 - 4).

Return value: Number of blocks to get (1 - 4).

```
pracq.trackProp(0, "Blocks", 2);
pracq.log(pracq.trackProp(0, "Blocks"));
```

```
pracq.trackProp(trackNo, "Channel")
pracq.trackProp(trackNo, "Channel", ch)
```

Sets/gets a MIDI channel (starts at zero).

trackNo: Track No. (0 - 3)

ch: MIDI channel to set (0 - 15).

Return value: MIDI channel to get (0 - 15).

```
pracq.trackProp(0, "Channel", 2); // "2" means ch.3.
pracq.log(pracq.trackProp(0, "Channel"));
```

```
pracq.trackProp(trackNo, "Volume")
pracq.trackProp(trackNo, "Volume", vol)
```

Sets/gets a volume.

trackNo: Track No. (0 - 3)

vol: Volume to set (0 - 127).

Return value: Volume to get (0 - 127).

```
pracq.trackProp(0, "Volume", 100);
pracq.log(pracq.trackProp(0, "Volume"));
```

```
pracq.trackProp(trackNo, "ProgramChange")
pracq.trackProp(trackNo, "ProgramChange", pc)
```

Sets/gets a program change.

trackNo: Track No. (0 - 3)

pc: Program change to set (0 - 127).

Return value: Program change to get (0 - 127).

```
pracq.trackProp(0, "ProgramChange", 2);
pracq.log(pracq.trackProp(0, "ProgramChange"));
```

```
pracq.trackProp(trackNo, "SwingValue")
pracq.trackProp(trackNo, "SwingValue", value)
```

Sets/gets a swing value. Note that 48 means 100 [%].

trackNo: Track No. (0 - 3)

value: Swing Value to set (-48 - 48).

Return value: Swing value to get (-48 - 48).

```
pracq.trackProp(0, "SwingValue", 2); // "2" means 4[%].
pracq.log(pracq.trackProp(0, "SwingValue"));
```

pracq.stepProp()

Sets/gets step properties.

```
pracq.stepProp(trackNo, stepNo, "StepStatus")
pracq.stepProp(trackNo, stepNo, "StepStatus", number)
```

Sets/gets a step status number.

Step status numbers (0 - 4) mean OFF, MUTE, ON, TIE, SKIP in order.

trackNo: Track No. (0 - 3)

stepNo: Step No. (0 - 127)

number: Step status number to set (0 - 4).

Return value: Step status number to get (0 - 4).

```
pracq.stepNotes(0, 0, 60);
pracq.stepProp(0, 0, "StepStatus", 2); // "2" means ON.
pracq.log(pracq.stepProp(0, 0, "StepStatus"));
```

```
pracq.stepProp(trackNo, stepNo, "Velocity")
pracq.stepProp(trackNo, stepNo, "Velocity", level)
```

Sets/gets a velocity level.

trackNo: Track No. (0 - 3)

stepNo: Step No. (0 - 127)

level: Velocity level to set (0 - 3).

Return value: Velocity level to get (0 - 3).

```
pracq.stepProp(0, 0, "Velocity", 2);
pracq.log(pracq.stepProp(0, 0, "Velocity"));
```

```
pracq.stepProp(trackNo, stepNo, "SwingResOn")
pracq.stepProp(trackNo, stepNo, "SwingResOn", status)
```

Sets/gets a swing response status (Off/On). “On” means Lengthened or Shortened.

trackNo: Track No. (0 - 3)

stepNo: Step No. (0 - 127)

status: Swing response status to set (0 (Off), 1 (On)).

Return value: Swing response status to get (0 (Off), 1 (On)).

```
pracq.stepProp(0, 0, "SwingResOn", 1); // "1" means ON.
pracq.log(pracq.stepProp(0, 0, "SwingResOn"));
```

```
pracq.stepProp(trackNo, stepNo, "SwingResLengthen")
pracq.stepProp(trackNo, stepNo, "SwingResLengthen", status)
```

Sets/gets a swing response status (Shortened/Lengthened).

trackNo: Track No. (0 - 3)

stepNo: Step No. (0 - 127)
status: Swing response status to set (0 (Shortened), 1 (Lengthened)).
Return value: Swing response status to get (0 (Shortened), 1 (Lengthened)).

```
pracq.stepProp(0, 0, "SwingResLengthen", 1); // "1" means Lengthened.
pracq.log(pracq.stepProp(0, 0, "SwingResLengthen"));
```

pracq.stepProp(trackNo, stepNo, "GateTime")
pracq.stepProp(trackNo, stepNo, "GateTime", level)

Sets/gets a gate time level.
trackNo: Track No. (0 - 3)
stepNo: Step No. (0 - 127)
level: Gate time level (0 - 3) to set.
Return value: Gate time level (0 - 3) to get.

```
pracq.stepProp(0, 0, "GateTime", 2);
pracq.log(pracq.stepProp(0, 0, "GateTime"));
```

pracq.stepProp(trackNo, stepNo, "StepTimeMag")
pracq.stepProp(trackNo, stepNo, "StepTimeMag", number)

Sets/gets a step time maginfication number.
Posible numbers are: 11, 12, 13, 14, 21, 23, 31, 32, 34, 41, 43, 51, 61, 71, 81
The numbers mean: 1, 1/2, 1/3, 1/4, 2, ... in order.
trackNo: Track No. (0 - 3)
stepNo: Step No. (0 - 127)
number: Step time maginfication number (11 - 81) to set.
Return value: Step time maginfication number (11 - 81) to get.

```
pracq.stepProp(0, 0, "StepTimeMag", 21); // "21" means 2.
pracq.log(pracq.stepProp(0, 0, "StepTimeMag"));
```

pracq.trkBlk()

Sets/gets active track No. and block No..

pracq.trkBlk()
pracq.trkBlk(trackNo)
pracq.trkBlk(trackNo, blockNo)

trackNo: Track No. (0 - 3) to set. (If you want to change a block No only, set -1.)
blockNo: Block No. (0 - 3) to set.
Return value: Number that contains a track No. in the ten's place and a block No. in the one's place.

```
var trkBlk = pracq.trkBlk(2);
pracq.log("Track=", parseInt(trkBlk / 10) + 1, ", ");
pracq.log("Block=", trkBlk % 10 + 1, "\n");
```

pracq.playKeyboard()

Play the keyboard of PRACQ.

```
pracq.playKeyboard(keyNo, -1, status)
pracq.playKeyboard(-1, noteNo, status)
```

keyNo: Keyboard pad No. (0 - 27) to play.
noteNo: Note No. (0 - 127) to play.
status: Off(0) or On(1).
Return value: status (0 or 1).

```
// Computer keyboard works as the keyboard pads of PRACQ.
if(pracq.eventType == E_STARTUP){
    pracq.regEvents = E_PCKEY_DN | E_PCKEY_UP;
} else if(pracq.eventType == E_PCKEY_DN){
    if(pracq.getInfo("PCKeyName", pracq.eventMsg) == "A"){
        pracq.playKeyboard(7, -1, 1);
    }
} else if(pracq.eventType == E_PCKEY_UP){
    if(pracq.getInfo("PCKeyName", pracq.eventMsg) == "A"){
        pracq.playKeyboard(7, -1, 0);
    }
}
```

pracq.midiOut()

Sends a MIDI message to the MIDI Out devices.

```
pracq.midiOut("Ctrl", array)
pracq.midiOut("Ctrl", int1, int2, ...)
```

Sends a MIDI message to the controller device that is set as "JavaScript-Controlled" type. (Settings -> Controller -> Grid MIDI Controller)
array: Array that contains 1-byte integers to send.
int1, int2, ...: 1-byte integers to send.
Return value: Sent data size (bytes).

```
// Example of System Exclusive Message
var msg = [0xF0, 0x00, 0x20, 0x29, 0x02, 0x0D, 0x0E, 0x01, 0xF7];
pracq.midiOut("Ctrl", msg);
```

```
pracq.midiOut("Play", array)
pracq.midiOut("Play", int1, int2, ...)
```

Sends a MIDI message to the playback device. (Settings -> Playback -> MIDI Out)
Usage is the same as **pracq.midiOut("Ctrl", ...)**.

JavaScript Standard

(Implemented by JUCE framework.)

String

```
var str1 = "abcde";
pracq.log(str1.substring(1, 3)); // bc
var str2 = "123";
pracq.log(parseInt(str2) + 1); // 124
```

Array

```
var arr = [10, 20, 30];
arr.push(40);
var sum = 0;
for(var i = 0; i < arr.length; i++){
    sum = sum + arr[i];
}
pracq.log(sum); // 100
```

Math

```
pracq.log(Math.PI); // 3.141592653589793
pracq.log(Math.random()); // e.g. 0.2147649195976555
```

Function

```
function func(x) {
    x = x + 1;
    return x;
}
pracq.log(func(2)); // 3
```