Logic Pro X Scripter Reference

source: http://noisefirm.com & http://culturedear.wordpress.com/

Scripter – Global attributes and functions

`NeedsTimingInfo:boolean`	Defining `NeedsTimingInfo` as `true` at the global scope enables the `GetTimingInfo()` function
`ResetParameterDefaults:boolean`	Sets UI controls to default values
`HandleMIDI(Event)`	This function is called each time a MIDI event is received by the plug-in, and is required to process incoming MIDI events. If you do not implement this function, events pass through the plug-in unaffected.
`ProcessMIDI()`	This function is called once per “process block,” which is determined by the host’s audio settings (sample rate and buffer size). This function is often used in combination with the`TimingInfo` object to make use of timing information from the host application. To enable the `GetTimingInfo` feature, add`NeedsTimingInfo = true` at the global script level.
`ParameterChanged(integer, real)`	This function is called each time one of the plug-in’s parameters is set to a new value. It is also called once for each parameter when you load a plug-in setting.
`Reset()`	This function is called when the plugin is reset
`Trace(value)`	Prints a message to the console that represents the supplied value of any type
`GetTimingInfo():TimingInfo`	Retrieves a `TimingInfo` object, which contains timing information that describes the state of the host transport and the current musical tempo and meter.
`GetParameter(string):real`	Returns a given parameter’s current value. `GetParameter()` is typically called inside the `HandleMIDI()` or `ProcessMIDI()`functions.

Event – Base class for all events

`send()`	Send the event
`sendAfterMilliseconds(ms:real)`	Send the event after the specified value has elapsed
`sendAtBeat(beat:real)`	Send the event at a specific beat in the host’s timeline
`sendAfterBeats(beats:real)`	Similar to `sendAtBeat()`, but uses the beat value as a delay in beats from the current position.
`trace()`	Prints the event to the plug-in console
`toString()`	Returns a string representation of the event

`channel(integer)`	Sets MIDI channel 1 to 16. Note: `Event.channel` is an event property, rather than a method, so it may be used in expressions such as `(evt.channel == 1)` where `evt` is an instance of`Event`)

Note – Base class for note events

`Note()`	Constructor
`toString()`	Returns a String representation of the Note event.

NoteOn – Represents a note on event

`NoteOn(Event)`	Constructor
`pitch(integer)`	Pitch from 1–127
`velocity(integer)`	Velocity from 0–127. A velocity value of 0 is interpreted as a note off event, not a note on.

NoteOff – Represents a note off event

`NoteOff(Event)`	Constructor
`pitch(integer)`	Pitch from 1–127
`velocity(integer)`	Velocity from 0–127

PolyPressure – Represents a Polyphonic aftertouch event

`PolyPressure(Event)`	Constructor
`pitch(integer)`	Pitch from 1–127
`value(integer)`	Pressure value from 0–127
`toString()`	Returns a String representation of the PolyPressure event.

ControlChange – Represents a ControlChange event

`ControlChange(Event)`	Constructor
`number(integer)`	Controller number from 0–127.
`value(integer)`	Controller value from 0–127.
`toString()`	Returns a String representation of the ControlChange event.

ProgramChange – Represents a ProgramChange event

`ProgramChange(Event)`	Constructor
`number(integer)`	Program change number from 0–127
`toString()`	Returns a String representation of the ProgramChange event.

ChannelPressure – Represents a ChannelPressure event

`ChannelPressure(Event)`	Constructor
`value(integer)`	Aftertouch value from 0–127
`toString()`	Returns a String representation of the ChannelPressure event.

PitchBend – Represents a PitchBend event

`PitchBend(Event)`	Constructor
`value(integer)`	14-bit pitch bend value from -8192–8191. A value of 0 is center.
`toString()`	Returns a String representation of the PitchBend event.

Fader – Represents a Fader event

`Fader(Event)`	Constructor
`value(integer)`	Fader value from 0–127
`toString()`	Returns a String representation of the Fader event.

TimingInfo – Contains timing information that describes the state of the host transport and the current musical tempo and meter

`playing:boolean`	Value is `true` when the host transport is running
`blockStartBeat:real`	Indicates the beat position at the start of the process block
`blockEndBeat:real`	Indicates the beat position at the end of the process block
`blockLength:real`	Indicates the length of the process block in beats.
`tempo:real`	Indicates the host tempo.
`meterNumerator:integer`	Indicates the host meter numerator
`meterDemoninator:integer`	Indicates the host meter denominator.
`cycling:boolean`	Value is `true` when the host transport is cycling

`leftCycleBeat:real`	Indicates the beat position at the start of the cycle range
`rightCycleBeat:real`	Indicates the beat position at the end of the cycle range

MIDI – Contains class-level variables and functions (you don't instantiate MIDI)

`_noteNames:string[]`	Contains names such as C and G# for all 128 MIDI notes
`_ccNames:string[]`	Contains names such as Expression and Sustain for all 128 MIDI controller numbers
`noteNumber(string)`	Returns the MIDI note number for a given note name. For example: C3 or B#2. Flats not permitted.
`noteName(real)`	Returns the name for a given MIDI note number.
`ccName(real)`	Returns the controller name for a given controller number
`allNotesOff()`	Sends the all notes off message on all MIDI channels
`normalizeStatus(real)`	Normalizes a value to the safe range of MIDI status bytes (128–239)
`normalizeChannel(real)`	Normalizes a value to the safe range of MIDI channels (1–16)
`normalizeData(real)`	Normalizes a value to the safe range of MIDI data bytes (0–127)
`_sendEventOnAllChannels(Event)`	Sends a given event to all MIDI channels