MIDI Utils API Documentation

updated for MIDI Utils 2.0, ©2025 Jeremy Bernstein

This project began as an attempt to rewrite some parts of Cockos' high-level ReaScript MIDI API, correcting some bugs and improving some (at least from my perspective) less than ideal behaviors and restrictions. Once I finished the subset of features I needed for one of my recent projects, I decided to keep going and just do the rest.

With some minor exceptions, this API is drop-in compatible with Cockos' API -- they can be used in parallel if necessary. I've gone to some trouble to ensure that the behaviors, return values, etc. remain consistent (with some documented exceptions).

- `MIDIUtils.MIDI_GetNote`() start and end ppq positions should be consistent with how REAPER itself handles note-on/note-off matching and duration determination.
- `MIDIUtils.MIDI_SetNote`() will no longer arbitrarily truncate or elongate overlapping notes under certain circumstances
- `MIDIUtils.MIDI_GetEvt`() doesn't confuse note-on and note-off messages under certain circumstances
- `MIDIUtils.MIDI_Get` / `Set` / `InsertNote`() support release velocity
- Write operations (set/insert/delete) are by definition unsorted, as they occur in memory. The optional 'unsorted' argument to several API functions has been dropped (will be ignored), and in the case of the note operations, has been replaced with a release velocity argument. Sorting occurs automatically post-commit.
- API 'Write' operations don't write back to REAPER until `MIDIUtils.MIDI_CommitWriteTransaction`() is called.
- API 'Read' operations are based on the data in memory, not the data in the take. If updates are potentially occuring in REAPER 'behind the back' of the API (such as in a defer script), call `MIDIUtils.MIDI_InitializeTake`() every frame, or whenever you need to resync the in-memory data with the actual state of the take in REAPER.
- 'Read' operations don't require a transaction, and will generally trigger a `MIDIUtils.MIDI_InitializeTake`(take) event slurp if the requested take isn't already in memory.

In general, for many projects, dropping down to the low-level `Get` / `SetAllEvts`() API is overkill and, for many developers, a terra incognita of status and data bytes in a 40-year-old serial specification. The high-level API is fairly well-designed and easily applicable to many, if not most MIDI scripting problems. But it's buggy and there doesn't appear to be much dev

interest in solving these problems (some of which have been in the docket since 2016ish) at the present time. So to make a long story short, I took some time to reimplement the very useful API in what Ihope is a more usable form (because fixed).

If the native API worked 100% correctly, there would be no point to this project, of course. My hope is that this replacement becomes obsolete at some point, when Cockos circles around to working on MIDI stuff again in some uncertain future. Until then, I hope this provides a useful interim solution to scripters struggling with the behavior and reliability of the native API.

USAGE:

```
    -- get the package path to MIDIUtils in my repository
  package.path = reaper.GetResourcePath() .. '/Scripts/sockmonkey72
Scripts/MIDI/?.lua'
  local mu = require 'MIDIUtils'

    -- true by default, enabling argument type-checking, turn off for
'production' code
  -- mu.ENFORCE_ARGS = false -- or use mu.MIDIUtils.MIDI_InitializeTake(),
see below

    -- enable overlap correction
  mu.CORRECT_OVERLAPS = true
    -- by default, note-ons take precedence when calculating overlap
correction
    -- turn this on to favor selected notes' note-off instead
  -- mu.CORRECT_OVERLAPS_FAVOR_SELECTION = true

    -- return early if something is missing (currently no dependencies)
  if not mu.CheckDependencies('My Script') then return end

  local take = reaper.MIDIEditor_GetTake(MIDIEditor_GetActive())
  if not take then return end

    -- acquire events from take (can pass true/false as 2nd arg to
enable/disable ENFORCE_ARGS)
    -- you will want to do this once per defer cycle in persistent scripts to
ensure that the
    -- internal state matches the current take state
  mu.MIDI_InitializeTake(take)

    -- inform the library that we'll be writing to this take
  mu.MIDI_OpenWriteTransaction(take)
```

```
  -- insert a note
mu.MIDI_InsertNote(take, true, false, 960, 1920, 0, 64, 64)

  -- insert a CC (using default CC curve)
mu.MIDI_InsertCC(take, true, false, 960, 0xB0, 0, 1, 64)

  -- insert a CC, get new index (using default CC curve)
local _, newidx = mu.MIDI_InsertCC(take, true, false, 1200, 0xB0, 0, 1, 96)

  -- insert another CC (using default CC curve)
mu.MIDI_InsertCC(take, true, false, 1440, 0xB0, 0, 1, 127)

  -- change the CC shape of the 2nd CC to bezier with a 0.66 tension
mu.MIDI_SetCCShape(take, newidx, 5, 0.66)

  -- commit the transaction to the take
  --   by default, this won't reacquire the MIDI events and update the
  --   take data in memory, pass 'true' as a 2nd argument if you want that
mu.MIDI_CommitWriteTransaction(take)

  -- pass 'true' as a 3rd argument to
MIDIUtils.MIDI_CommitWriteTransaction()
  --   to do this automatically
reaper.MarkTrackItemsDirty(
  reaper.GetMediaItemTake_Track(take),
  reaper.GetMediaItemTake_Item(take))
```

---

from ReaPack projects, you can include a copy of MIDIUtils.lua like this

```
@provides MyScriptFolder/MIDIUtils.lua
https://raw.githubusercontent.com/jeremybernstein/ReaScripts/main/MIDI/MIDIUt
ils.lua
```

which will grab the latest version of MIDIUtils.lua and place it in a project-specific folder (the folder will be created if it doesn't already exist). If you include MIDIUtils like this, I would be happy for an acknowledgement of usage in your README.

SetOnError

`MIDIUtils.SetOnError(fn)`

- `SetOnError`: set an optional error callback for xpcall(), otherwise a traceback will be posted to the REAPER console window by default.

Return Values: none

`boolean` rv = `MIDIUtils.CheckDependencies(callerScriptName)`

- `CheckDependencies`: check whether all MIDIUtils dependencies are met, pass the name of your script as an argument, will be used as an identifier in any error message generated.

  Return Values:
  `boolean` rv: `true` if dependencies are met, `false` otherwise

MIDI_InitializeTake

`MIDIUtils.MIDI_InitializeTake`(take, enforceargs = `true`)

- `MIDI_InitializeTake`: gather the events in a `MediaItem_Take` for use with MIDIUtils. In simple usage, this call is optional -- API calls will automatically call `MIDIUtils.MIDI_InitializeTake`() internally if the provided take is not already prepared.

  `NOTE`: if using MIDIUtils in a defer() script, you will want to call this once per defer cycle to ensure that the MIDIUtils internal state is synced with the take state. The optional 'enforceargs' argument can be used to disable API argument type enforcement for efficiency in production code.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER

  Return Values: none

MIDI_CountEvts

`boolean` rv, `number` notecnt, `number` ccevtcnt, `number` textsyxevtcnt = `MIDIUtils.MIDI_CountEvts`(take)

- `MIDI_CountEvts`: provide a count of take events by type.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER

  Return Values:
  `boolean` rv: `true` if successful, `false` otherwise
  `number` notecnt: count of note-on events
  `number` ccevtcnt: count of CC, pitch bend, program change, aftertouch events

`number` textsyxevtcnt: count of meta and system exclusive events (not including bezier curves)

MIDI_OpenWriteTransaction

`MIDIUtils.MIDI_OpenWriteTransaction`(take)

- `MIDI_OpenWriteTransaction`: start a 'write' transaction. MIDIUtils performs all of its MIDI data manipulation in memory. Unlike Cockos' high-level MIDI API, there are no 'immediately sorted' API calls. To make changes to the data, you are required to open a transaction, make all changes, and then commit the transaction, which will write all changes in a single bulk set action.

Arguments:
`MediaItem_Take` take: the `MediaItem_Take` provided by REAPER

Return Values: none

MIDI_CommitWriteTransaction

`boolean` rv =
`MIDIUtils.MIDI_CommitWriteTransaction`(take, refresh, dirty)

- `MIDI_CommitWriteTransaction`: end a 'write' transaction and commit the data to the take.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `boolean` refresh: when true, MIDIUtils will re-initialize the take after commit
  this is useful if you intend to perform further manipulation on the data post-commit
  `boolean` dirty: when true, MIDIUtils will dirty the take post-commit

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise

MIDI_CorrectOverlaps

`boolean` rv =
`MIDIUtils.MIDI_CorrectOverlaps`(take, favorSelection)

- `MIDI_CorrectOverlaps`: manually apply overlap correction to the current take. See also MIDIUtils.CORRECT_OVERLAPS and MIDIUtils.CORRECT_OVERLAPS_FAVOR_SELECTION for automatic overlap correction options.

Arguments:
`MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
`boolean` favorSelection: when true, overlap correction will attempt to leave any selected
notes untouched (the note-on of position unselected note events will
be changed). When `false` (or missing), overlap correction will preserve
note-on positions and move note-offs to prevent MIDI overlaps. [optional]

Return Values:
`boolean` rv: `true` on success, `false` otherwise

MIDI_GetNote

`boolean` rv, `boolean` selected, `boolean` muted, `number` ppqpos, `number` endppqpos, `number` chan, `number` pitch, `number` vel, `number` relvel =
`MIDIUtils.MIDI_GetNote`(take, idx)

- `MIDI_GetNote`: Get MIDI note properties.

Arguments:
`MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
`number` idx: note index (see `MIDIUtils.MIDI_CountEvts` for the number of MIDI notes)

Return Values:
`boolean` rv: `true` on success, `false` otherwise
`boolean` selected: `true` if note is selected
`boolean` muted: `true` if note is muted
`number` ppqpos: note-on PPQ position
`number` endppqpos: note-off PPQ position
`number` chan: MIDI channel (0 - 15)
`number` pitch: MIDI note number (0 - 127)
`number` vel: note-on velocity (0 - 127)
`number` relvel: note-off velocity (0 - 127)

MIDI_SetNote

`boolean` rv =
`MIDIUtils.MIDI_SetNote`(take, idx, selected, muted, ppqpos, endppqpos, chan, pitch, vel, relvel)

- `MIDI_SetNote`: Set MIDI note properties for an existing event.

Arguments:
`MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
`number` idx: note index to modify (see `MIDIUtils.MIDI_CountEvts` for the number of MIDI notes)
`boolean` selected: `true` if note is selected [optional]
`boolean` muted: `true` if note is muted [optional]
`number` ppqpos: note-on PPQ position [optional]
`number` endppqpos: note-off PPQ position [optional]
`number` chan: MIDI channel (0 - 15) [optional]
`number` pitch: MIDI note number (0 - 127) [optional]
`number` vel: note-on velocity (0 - 127) [optional]
`number` relvel: note-off velocity (0 - 127) [optional]

Return Values:
`boolean` rv: `true` on success, `false` otherwise

MIDI_InsertNote

`boolean` rv, idx =
`MIDIUtils.MIDI_InsertNote`(take, selected, muted, ppqpos, endppqpos, chan, pitch, vel, relvel)

- `MIDI_InsertNote`: Create a new MIDI note event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `boolean` selected: `true` if note is selected
  `boolean` muted: `true` if note is muted
  `number` ppqpos: note-on PPQ position
  `number` endppqpos: note-off PPQ position
  `number` chan: MIDI channel (0 - 15)
  `number` pitch: MIDI note number (0 - 127)
  `number` vel: note-on velocity (0 - 127)
  `number` relvel: note-off velocity (0 - 127) [optional, 0 if not provided]

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise
  `number` idx: new note index

MIDI_DeleteNote

`boolean` rv =
`MIDIUtils.MIDI_DeleteNote`(take, idx)

- `MIDI_DeleteNote`: Delete a note event.

Arguments:
`MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
`number` idx: note index to delete (see `MIDIUtils.MIDI_CountEvts` for the number of MIDI notes)
Return Values:
`boolean` rv: `true` on success, `false` otherwise

MIDI_GetCC

`boolean` rv, `boolean` selected, `boolean` muted, `number` ppqpos, `number` chanmsg, `number` chan, `number` msg2, `number` msg3 =
`MIDIUtils.MIDI_GetCC`(take, idx)

- `MIDI_GetCC`: Get CC/channel message properties.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: note index (see `MIDIUtils.MIDI_CountEvts` for the number of MIDI notes)
  Return Values:

  `boolean` rv: `true` on success, `false` otherwise
  `boolean` selected: `true` if event is selected
  `boolean` muted: `true` if event is muted
  `number` ppqpos: event PPQ position
  `number` chanmsg: message type:

  - 0xA0 - poly pressure / aftertouch
  - 0xB0 - continuous controller
  - 0xC0 - program change [* only requires 2 bytes]
  - 0xD0 - channel pressure / aftertouch [* only requires 2 bytes]
  - 0xE0 - pitch bend

  `number` chan: MIDI channel (0 - 15)
  `number` msg2: 2nd message byte (0 - 127)
  `number` msg3: 3rd message byte (0 - 127)

SetCC

`boolean` rv =
`MIDIUtils.MIDI_SetCC`(take, idx, selected, muted, ppqpos, chanmsg, chan, msg2, msg3)

- `MIDI_SetCC`: Set CC/channel message properties for an existing event.

Arguments:
`MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
`number` idx: CC event index to modify (see `MIDIUtils.MIDI_CountEvts` for the number of CC-like events)
`boolean` selected: `true` if event is selected [optional]
`boolean` muted: `true` if event is muted [optional]
`number` ppqpos: event PPQ position [optional]
`number` chanmsg: message type [optional]:

- 0xA0 - poly pressure / aftertouch
- 0xB0 - continuous controller
- 0xC0 - program change [* only requires 2 bytes]
- 0xD0 - channel pressure / aftertouch [* only requires 2 bytes]
- 0xE0 - pitch bend

`number` chan: MIDI channel (0 - 15) [optional]
`number` msg2: 2nd message byte (0 - 127) [optional]
`number` msg3: 3rd message byte (0 - 127) [optional]

Return Values:
`boolean` rv: `true` on success, `false` otherwise

MIDI_InsertCC

`boolean` rv, `number` idx = `MIDIUtils.MIDI_InsertCC`(take, selected, muted, ppqpos, chanmsg, chan, msg2, msg3)

- `MIDI_InsertCC`: Create a new CC/channel message event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `boolean` selected: `true` if event is selected
  `boolean` muted: `true` if event is muted
  `number` ppqpos: PPQ position
  `number` chanmsg: message type:

  - 0xA0 - poly pressure / aftertouch
  - 0xB0 - continuous controller
  - 0xC0 - program change [* only requires 2 bytes]
  - 0xD0 - channel pressure / aftertouch [* only requires 2 bytes]
  - 0xE0 - pitch bend

`number` chan: MIDI channel (0 - 15)
`number` msg2: 2nd message byte (0 - 127)
`number` msg3: 3rd message byte (0 - 127)

Return Values:
`boolean` rv: `true` on success, `false` otherwise
`number` idx: new CC event index

MIDI_DeleteCC

`boolean` rv =
`MIDIUtils.MIDI_DeleteCC`(take, idx)

- `MIDI_DeleteCC`: Delete a CC/channel message event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: CC event index to delete (see `MIDIUtils.MIDI_CountEvts` for the number of CC-like events)

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise

MIDI_GetCCShape

`boolean` rv, `number` shape, `number` beztension =
`MIDIUtils.MIDI_GetCCShape`(take, idx)

- `MIDI_GetCCShape`: Get CC shape and bezier tension.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: CC event index to query

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise
  `number` shape: curve shape

  - 0 - square
  - 1 - linear
  - 2 - slow start/end
  - 3 - fast start
  - 4 - fast end
  - 5 - bezier

  `number` beztension: bezier tension

MIDI_SetCCShape

`boolean` rv =
`MIDIUtils.MIDI_SetCCShape`(take, idx, shape, beztension)

- `MIDI_SetCCShape`: Set CC shape and bezier tension.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: CC event index to query
  `number` shape: curve shape

  - 0 - square
  - 1 - linear
  - 2 - slow start/end
  - 3 - fast start
  - 4 - fast end
  - 5 - bezier

  `number` beztension: bezier tension [required for shape 5 (bezier)]

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise

MIDI_GetTextSysexEvt

`boolean` rv, `boolean` selected, `boolean` muted, `number` ppqpos, `number` type, `string` msg =
`MIDIUtils.MIDI_GetTextSysexEvt`(take, idx)

- `MIDI_GetTextSysexEvt`: Get meta / system exclusive message properties.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (see `MIDIUtils.MIDI_CountEvts` for the number of meta / sysex events)

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise
  `boolean` selected: `true` if event is selected
  `boolean` muted: `true` if event is muted
  `number` ppqpos: event PPQ position
  `number` type: message type

  - -1 - system exclusive

    - 1 - 14: meta text event:
        - 1 - text
        - 2 - copyright notice
        - 3 - track name
        - 4 - instrument name
        - 5 - lyrics
        - 6 - marker
        - 7 - cue point
        - 8 - program name
        - 9 - device name
        - 10 - 14: REAPER-specific, undocumented
    - 15: REAPER notation event

  `string` msg: message payload (no header or footer bytes)

MIDI_SetTextSysexEvt

`boolean` rv = `MIDIUtils.MIDI_SetTextSysexEvt`(take, idx, selected, muted, ppqpos, type, msg)

- `MIDI_SetTextSysexEvt`: Set meta / system exclusive message properties for an existing event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (see `MIDIUtils.MIDI_CountEvts` for the number of meta / sysex events)
  `boolean` selected: `true` if event is selected [optional]
  `boolean` muted: `true` if event is muted [optional]
  `number` ppqpos: event PPQ position [optional]
  `number` type: message type [optional, required if msg is provided]
    - -1 - system exclusive
    - 1 - 14: meta text event:
        - 1 - text
        - 2 - copyright notice
        - 3 - track name
        - 4 - instrument name

        - 5 - lyrics
        - 6 - marker
        - 7 - cue point
        - 8 - program name
        - 9 - device name
        - 10 - 14: REAPER-specific, undocumented
    - 15: REAPER notation event

  `string` msg: message payload (no header or footer bytes) [optional, required if type is provided]

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise

MIDI_InsertTextSysexEvt

`boolean` rv, `number` idx = `MIDIUtils.MIDI_InsertTextSysexEvt`(take, selected, muted, ppqpos, type, bytestr)

- `MIDI_InsertTextSysexEvt`: Create a new meta / system exclusive message event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `boolean` selected: `true` if event is selected
  `boolean` muted: `true` if event is muted
  `number` ppqpos: PPQ position
  `number` type: message type
    - -1 - system exclusive
    - 1 - 14: meta text event:
        - 1 - text
        - 2 - copyright notice
        - 3 - track name
        - 4 - instrument name
        - 5 - lyrics
        - 6 - marker
        - 7 - cue point
        - 8 - program name
        - 9 - device name

        - 10 - 14: REAPER-specific, undocumented
    - 15: REAPER notation event

  `string` msg: message payload (no header or footer bytes)

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise
  `number` idx: new CC event index

MIDI_DeleteTextSysexEvt

`boolean` rv =
  `MIDIUtils.MIDI_DeleteTextSysexEvt`(take, idx)

- `MIDI_DeleteTextSysexEvt`: Delete a meta / system exclusive message event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index to delete (see `MIDIUtils.MIDI_CountEvts` for the number of meta / sysex events)

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise

MIDI_GetEvt

`boolean` rv, `boolean` selected, `boolean` muted, `number` ppqpos, `string` msg =
  `MIDIUtils.MIDI_GetEvt`(take, idx)

- `MIDI_GetEvt`: Get event properties.

Arguments:
`MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
`number` idx: event index (see `MIDIUtils.MIDI_CountEvts` for event count)

Return Values:
`boolean` rv: `true` on success, `false` otherwise
`boolean` selected: `true` if event is selected
`boolean` muted: `true` if event is muted
`number` ppqpos: event PPQ position
`string` msg: complete message payload (incl. status, header and/or footer bytes)

MIDI_SetEvt

`boolean` rv =
  `MIDIUtils.MIDI_SetEvt`(take, idx, selected, muted, ppqpos, msg)

- `MIDI_SetEvt`: Set event properties for an existing event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (see `MIDIUtils.MIDI_CountEvts` for event count)
  `boolean` selected: `true` if event is selected [optional]
  `boolean` muted: `true` if event is muted [optional]
  `number` ppqpos: event PPQ position [optional]
  `string` msg: complete message payload (incl. status, header and/or footer bytes) [optional]

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise

MIDI_InsertEvt

`boolean` rv, `number` idx = `MIDIUtils.MIDI_InsertEvt`(take, selected, muted, ppqpos, bytestr)

- `MIDI_SetEvt`: Insert new event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `boolean` selected: `true` if event is selected
  `boolean` muted: `true` if event is muted
  `number` ppqpos: event PPQ position
  `string` msg: complete message payload (incl. status, header and/or footer bytes)

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise

MIDI_DeleteEvt

`boolean` rv = `MIDIUtils.MIDI_DeleteEvt`(take, idx)

- `MIDI_DeleteEvt`: Delete an event.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (see `MIDIUtils.MIDI_CountEvts` for event count)

  Return Values:
  `boolean` rv: `true` on success, `false` otherwise

MIDI_EnumSelNotes

`number` idx =
`MIDIUtils.MIDI_EnumSelNotes`(take, idx)

- `MIDI_EnumSelNotes`: Returns the index of the next selected MIDI note event after idx (-1 if there are no more selected events).

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (pass -1 to get the first selected note event)

  Return Values:
  `integer` idx: index of next selected MIDI Note event

MIDI_EnumSelCC

`number` idx =
`MIDIUtils.MIDI_EnumSelCC`(take, idx)

- `MIDI_EnumSelCC`: Returns the index of the next selected MIDI CC event after idx (-1 if there are no more selected events).

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (pass -1 to get the first selected CC event)

  Return Values:
  `integer` idx: index of next selected MIDI CC event

MIDI_EnumSelTextSysexEvts

`number` idx =
`MIDIUtils.MIDI_EnumSelTextSysexEvts`(take, idx)

- `MIDI_EnumSelCC`: Returns the index of the next selected MIDI meta / system exclusive event after idx (-1 if there are no more selected events).

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (pass -1 to get the first selected meta / sysex event)

  Return Values:
  `integer` idx: index of next selected MIDI meta / sysex event

MIDI_EnumSelEvts

`number` idx =
`MIDIUtils.MIDI_EnumSelEvts`(take, idx)

- `MIDI_EnumSelCC`: Returns the index of the next selected event after idx (-1 if there are no more selected events).

Arguments:
`MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
`number` idx: event index (pass -1 to get the first selected event)

Return Values:
`integer` idx: index of next selected event of any type (note-on, note-off, CC, meta/sysex)

MIDI_CountAllEvts

`number` allcnt =
`MIDIUtils.MIDI_CountAllEvts`(take)

- `MIDI_CountEvts`: provide a count of all take events.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER

  Return Values:
  `integer` allcnt: count of all events (note-on, note-off, CC, meta/sysex)

MIDI_EnumNotes

`number` idx =
`MIDIUtils.MIDI_EnumNotes`(take, idx)

- `MIDI_EnumNotes`: Returns the index of the next MIDI note event after idx (-1 if there are no more events).

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (pass -1 to get the first note event)

  Return Values:
  `integer` idx: index of next MIDI Note event

MIDI_EnumCC

`number` idx =
`MIDIUtils.MIDI_EnumCC`(take, idx)

- `MIDI_EnumCC`: Returns the index of the next MIDI CC event after idx (-1 if there are no more events).

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (pass -1 to get the first CC event)

Return Values:
`integer` idx: index of next MIDI CC event

MIDI_EnumTextSysexEvts

`number` idx =
`MIDIUtils.MIDI_EnumTextSysexEvts`(take, idx)

- `MIDI_EnumTextSysexEvts`: Returns the index of the next MIDI meta / system exclusive event after idx (-1 if there are no more events).

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (pass -1 to get the first meta/sysex event)

  Return Values:
  `integer` idx: index of next MIDI meta/sysex event

MIDI_EnumEvts

`number` idx =
`MIDIUtils.MIDI_EnumEvts`(take, idx)

- `MIDI_EnumEvts`: Returns the index of the next event after idx (-1 if there are no more events).

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` idx: event index (pass -1 to get the first event)

  Return Values:
  `integer` idx: index of next event of any type (note-on, note-off, CC, meta/sysex)

MIDI_GetCCValueAtTime

`boolean` rv, `number` val, `number` ppqpos, `integer` chanmsg, `integer` chan, `integer` msg2out, `integer` msg3out =
`MIDIUtils.MIDI_GetCCValueAtTime`(take, chanmsg, chan, msg2, time)

- `MIDI_GetCCValueAtTime`: Get the effective CC value at a given time position, applying any curve.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `number` chanmsg: message type:

  - 0xA0 - poly pressure / aftertouch
  - 0xB0 - continuous controller

- 0xC0 - program change [* only requires 2 bytes]
- 0xD0 - channel pressure / aftertouch [* only requires 2 bytes]
- 0xE0 - pitch bend

`number` chan: MIDI channel (0 - 15)
`number` msg2: 2nd message byte (0 - 127)
e.g. the CC# or poly aftertouch note#
[unused for program change, channel pressure and pitch bend types]
`number` time: project time in seconds (as returned by `reaper.GetCursorPosition`() or similar)

Return Values:
`boolean` rv: `true` on success, `false` otherwise
`number` val: the effective CC value at the given time position (floating-point, interpolated)
`number` ppqpos: PPQ position in the take of the requested time
`integer` chanmsg: the message type (generally the same as was passed in)
`integer` chan: MIDI channel (0 - 15) (the same as was passed in)
`integer` msg2out: the 2nd MIDI byte for the controller message
`integer` msg3out: the 3rd MIDI byte for the controller message

MIDI_NoteNumberToNoteName

`string` notename =
`MIDIUtils.MIDI_NoteNumberToNoteName`(notenum, names = { 'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B' })

- `MIDI_NoteNumberToNoteName`: Returns the note name + octave (i.e. A3, D#-1) of the provided MIDI note number. Takes REAPER's MIDI octave name display offset preference into account.

Arguments:
`number` notenum: MIDI note number (0 - 127)
`table` names: table of 12 pitch class names, from C - B, as they should be used for display [optional]

Return Values: note name + octave string for the input value

MIDI_NoteNameToNoteNumber

`integer` notenum =
`MIDI_NoteNameToNoteNumber`(notename, names = { 'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B' })

- `MIDI_NoteNameToNoteNumber`: returns the MIDI note number from a provided note name + octave (i.e. A3, D#-1), taking REAPER's MIDI octave name display offset preference into account.

  Arguments:

  `string` notename: the note name + octave for conversion
  `table` names: table of 12 pitch class names, from C - B, as they should be used for display [optional]

MIDI_GetPPQ

`number` ppq =
`MIDIUtils.MIDI_GetPPQ`(take)

- `MIDI_GetPPQ`: Returns the PPQ of the provided take.
- **Arguments:**
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
- **Return Values:**
  `number` ppq: PPQ value (parts (ticks) per quarter note) of the provided take

MIDI_SelectAll

`MIDIUtils.MIDI_SelectAll`(take, wantsSelect)

- `MIDI_SelectAll`: Select or deselect all MIDI content in the provided take.

  Arguments:
  `MediaItem_Take` take: the `MediaItem_Take` provided by REAPER
  `boolean` wantsSelect: select (true) or deselect (false)

  Return Values: none

---

post

`MIDIUtils.post`(...)

- `post`: Convenience method to post a message (or comma-delimited messages) to the REAPER console.

p

`MIDIUtils.p`(...)

- `p`: Convenience method to post a message (or comma-delimited messages) to the REAPER console.

tprint

`MIDIUtils.tprint`(table)

- `tprint`: Convenience method to print the contents of a Lua table to the REAPER console.

  Arguments:
  `table` table: a Lua table

tableCopy

`MIDIUtils.tableCopy`(table)

- `tableCopy`: makes a deep copy of a Lua table, only applicable for tables *without* table keys.

  Arguments:
  `table` table: a Lua table

  Return Values:
  `table` newTable: a unique copy of the input table, containing unique values which no longer refer to the source table.

MIDI_GetState

`MIDIUtils.MIDI_GetState`()

- `MIDI_GetState`: collects the current state of MIDIUtils, including all events, iteration states, etc. into a table.

  Arguments: none

  Return Values:
  `table` state: a Lua table, should be passed, unmodified to `MIDI_SetState` when the MIDIUtils state is to be restored.

MIDI_SetState

`MIDIUtils.MIDI_SetState`(state)

- `MIDI_SetState`: replaces the current state of MIDIUtils with a state described in the provided table (supplied by `MIDI_GetState`).

  Arguments:
  `table` state: a Lua table acquired through a call to `MIDI_GetState`

  Return Values: none

`MIDIUtils.ENFORCE_ARGS` = `true`

- Flag to enable/disable argument type-checking. When enabled, submitting the wrong argument type(s) to MIDIUtils functions will cause an error, useful for debugging code. Disable in production code, since the type checking adds some minimal overhead. On by default, see also MIDIUtils.InitializeTake(), where this can be set.

`MIDIUtils.CORRECT_OVERLAPS` = `false`

- Flag to enable/disable overlap correction on commit. Off by default, but may be desirable if the option "Correct overlapping notes while editing" is enabled, or for the Inline MIDI Editor (where that option is always active).

`MIDIUtils.CORRECT_OVERLAPS_FAVOR_SELECTION` = `false`

- If `CORRECT_OVERLAPS` is enabled, `CORRECT_OVERLAPS_FAVOR_SELECTION` determines whether the selection's note-off takes precedence over an unselected note-on when performing the overlap correction. Off by default.

`MIDIUtils.CORRECT_OVERLAPS_FAVOR_NOTEON` = `false`

- If `CORRECT_OVERLAPS` and `CORRECT_OVERLAPS_FAVOR_SELECTION` are enabled, `CORRECT_OVERLAPS_FAVOR_NOTEON` determines whether a note collision of selected events favors the note-on or note-off when performing the overlap correction. Off by default.

`MIDIUtils.ALLNOTESOFF_SNAPS_TO_ITEM_END` = `true`

- If `ALLNOTESOFF_SNAPS_TO_ITEM_END` is enabled, the "All Notes Off" event (CC#123, at the end of every item) will be snapped to the end of the item, rather than floating around near the end of the item, or being stuck at the end of the last note, or whatever REAPER decides to do with it. On by default.

`MIDIUtils.CLAMP_MIDI_BYTES` = `false`

- REAPER's default behavior is to AND incoming data byte values with 0x7F, causing values above 127 to wrap around, where 128 = 0, 129 = 1, and so on. When this option is enabled, incoming data bytes will be clipped, such that values below 0 = 0, and values above 127 = 127. Off by default.

`MIDIUtils.CORRECT_EXTENTS` = `false`

- Modify the item extents to accommodate changes to the contents. Off by default.

`MIDIUtils.USE_XPCALL` = `true`

- By default, calls to MIDIUtils are wrapped in `xpcall` to help catch and display errors. This comes with a certain non-zero cost, though. Disable `USE_XPCALL` to use direct calling.

`MIDIUtils.COMMIT_CANSKIP` = `false`

- When enabled, and if MIDIUtils detects that there has been no change to the MIDI events it would send to REAPER at commit time, MIDIUtils won't send the data. Use with care. If this is enabled, the user can use (the otherwise undocumented function) `MIDIUtils.MIDI_ForceNextTransaction()` to force the data to be sent on the next commit.