

Advanced Control Relationships

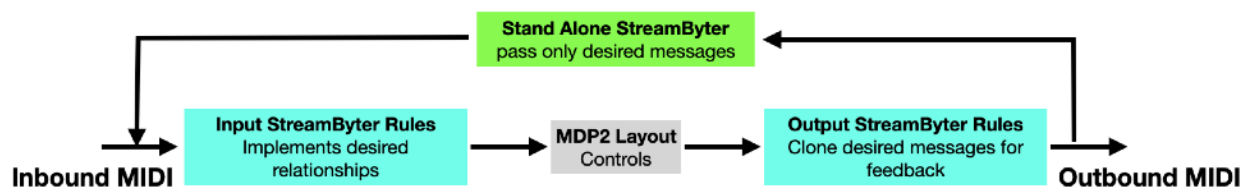
In MIDI Designer Pro 2 Using StreamByte Loopback

Overview

MIDI Designer Pro 2 can implement complex relationships with super control, named ticks, and other built in features. StreamByte allows for advanced processing on either MDP2 input or output. But for complex solutions, we need to simultaneously process current layout control settings, MIDI output and MIDI input, and have results impact either input, output, or layout display.

The solution has been staring us in the face since Dan first implemented StreamByte - use the stand-alone SB to feed back selected messages from the output back to input. With this, the input SB rules can implement rules based on current control settings, output, and inbound MIDI. Any control relationship that can be defined with StreamByte logic can be implemented.

But looping all output MIDI directly back to MDP2 input can generate knarly MIDI feedback. The key is to filter so only desired data gets back to the input.



What follows is my approach to setting up the feedback logic and some sample layouts annotated with logic and StreamByte rules.

This hack works on both iPad and M1 Macs. It seems a little more robust on the Mac, since you don't have to re-wake the stand alone StreamByte after the iPad sleeps.

Feedback Logic

1. We need a message format that can not be confused with any other MIDI message, as the feedback messages will also appear in the outbound MIDI. This typically implies SysEx. Confusion Studio's assigned SysEx ID (00 01 7E) will give low probability of a collision.

2. If a control is only used in the advanced rules, then build it as a SysEx with the header 00 01 7E 1x ..
3. To spy on output MIDI messages, clone those messages in outbound rules.

Stand Alone Streambyter - filter

1. Open in slide-over mode, install rules, then slide out of sight.
2. If the device has gone to sleep, SB may need to be woken by sliding into view
3. The filter is:


```
If M0 == F0 00 01 7E
    # Allow message through if it matches, otherwise block
Else
    Block
End
```

Note - the != version would be shorter, but seems to have a bug for four bytes)

Note - any MIDI router that implements filters can be used in place of SB.

Output Rules - clone desired messages

My opening assumption is that I will only be spying on CC messages, not note on/off, pitch bend, etc. And the simplest conversion is to just drop the message into the SysEx message, as follows:

Output Transformation

```

  Bx  CC  Val
  M00 M01 M02
Becomes
  F0  00  01  7E  0x  CC  Val  F7
  M00 M01 M02 M03 M04 M05 M06 M07

M04 Channel(stripped of leading B_)
M05 CC #
M06 Value
```

Output rule to spy on MIDI Channel 1, CC 01 (B0 01 vv)

```

If M0 == B0 01
  Mat I0 = M0 - B0
  Snd 00 01 7E I0 M1 M2 F7
End
```

Note - this means 00 01 7E 0X ... SysEx messages are reserved for CCs. When building a control dedicated to feedback, use 00 01 7E 1x ... (or higher)

Input Rules - work happens here

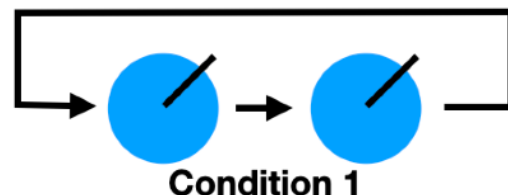
1. With the MIDI feedback, input rules can now make calculations based on layout control settings, output, and MIDI input from other devices. Several examples follow.
2. Most looped messages should be blocked in the inbound rules when processing is complete to help avoid MIDI loops.
3. The soapbox.audeonic.com forum has a wealth of SB example code.

MDP2 Layout - a few notes

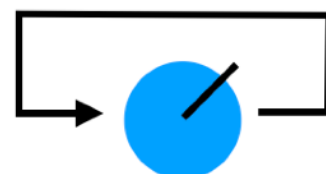
1. If an input rules calculation is intended to go to outbound MIDI without further MDP2 interaction, you need two controls. First (supercontrol) receives the inbound results, the second (subcontrol) sends the message. A MDP2 control only receives or transmits MIDI at any instance. It can do both, just not simultaneously. (This is as designed and as desired.)
2. On occasion, a separate scaler control (with named ticks) may be needed to process inbound data. (See the “Sixteen step memory” example.)
3. If a control is only needed in the inbound rules, build it directly in SysEx to avoid a translation step in outbound SB rules.
4. When values are “stored” in SB variables, you may need to initialize those values to match the layout. Easiest method is a supercontrol that sends all current values, type momentary, force not stepper.

Avoiding Feedback

1. I have avoided locking up any layouts with a feedback loop. If you do, it may require killing the app to recover. This is why this is considered a HACK, not a supported approach (at least for now).
2. The configuration that causes problems is a loop driving a supercontrol, back into the subcontrol that sends the original message. There could be additional controls in this string.
3. Surprisingly, this configuration is not as much a problem, since the same control will not send and receive at the same time. It may be cranky, but usually will not lock up.



Condition 1



Condition 2

4. Another source of unanticipated feedback is losing track of which messages are feeding back into which modules, or which SB variables are being used in calculations.

Summary

1. Using feedback of carefully selected items allows for more complex MDP2 control relationships. But it does require more attention in crafting StreamByte rules. The samples provided are a start at the more complex structures that can be crafted. Each section tells the approach to building the logic and provides the supporting SB rules.
2. In the samples, controls that would normally be hidden in play mode are visible, shown in yellow, with no touch in play mode.

Feedback Message Format

Output Transformation - focused on only control change messages

The output CC

```
Bx  CC  Val
M00 M01 M02
```

Becomes

```
F0  00  01  7E  0x  CC  Val  F7
M00 M01 M02 M03 M04 M05 M06 M07
```

```
M04 Channel(stripped of leading B_)
M05 CC #
M06 Value
```

Feedback Rules

If M0 == F0 00 01 7E

Allow message through if it matches header, otherwise block

Else

Block

End

Switched Control

- When switch is on, secondary control follows primary, otherwise no response

Logic

- Logic - When the switch is off, the SysEx is blocked, when on, it drives the super control that drives the output knob
- The knobs are CCs, the switch and relay are Sysex.
- The switch SysEx is blocked in the input SB to prevent the condition 2 loop, the switch MIDI receive is also off.

Output Rules

Output Rules "Switched Control"

Undeclared variables

i0 = temp storage for stripping leading nibble of message type

If Load

 Define Header F0 00 01 7E

End

If M0 == B0 01 # Primary_00

 Mat i0 = M0 - B0

 Snd Header i0 M1 M2 F7

End

End Output Rules "Switched Control"

Input Rules

Input Rules - "Switched Control"

If Load

 Define Header F0 00 01 7E

 Alias L00 Switch_00

End

If M0 == Header

 If M4 == 10 00 # Switch_00

 Ass Switch_00 = M6

```
        Block # finished with switch
    End

    If M4 == 00 01 # Primary_00
        If Switch_00 == 0
            Block # Block message while flag is off
        End
    End
End

## End Input Rules - "Switched Control" ##
```

Switched Control with memory

- When switch is turned on, secondary control jumps to current value, then follows primary.
- When the switch is off, the primary value is stored, no message is sent to the layout

Logic

- When the switch is turned on, the stored value is sent, and primary value updates are sent through to the layout.
- The knobs are CCs, the switch and relay are Sysex.
- The switch SysEx is blocked in the input SB to prevent the condition 2 loop, the switch MIDI receive is also off.

Note - we could enhance this so the secondary returns to a default value when the switch is turned off

Output Rules

Output Rules "Switched Control with Memory"

If Load

 # Define Header F0 00 01 7E

End

If M0 == B0 09 # Primary_01

 Mat I0 = M0 - B0

 Snd Header I0 M1 M2 F7

End

End Output Rules "Switched Control with Memory"

Input Rules

Input Rules - "Switched Control with Memory"

If Load

 # Define Header F0 00 01 7E

 Alias L04 Switch_01

 Alias L05 Primary_01

End


```
If M0 == Header
  If M4 == 10 03 # Switch_01
    Ass Switch_01 = M6
    If Switch_01 != 0
      Snd Header 10 04 Primary_01 F7
    End
  Block
End

  If M4 == 00 09 # Primary_01
    Ass Primary_01 = M6
    If Switch_01 != 0
      Snd Header 10 04 Primary_01 F7
    End
  Block
End
End

## End Input Rules - "Switched Control with Memory" ##
```

Multiplier

- The output of two controls is multiplied

Logic

- When value A or B is received, it is stored, multiplied by the stored value of the other input, and sent out in SysEx
- Max MIDI value is 127, so A and B are limited to max value 11
- A, B, and AxB are CCs, the relay is SysEx

Output Rules

Output Rules "Multiplier"

If Load

 # Define Header F0 00 01 7E

End

If M0 == B0 04 # A knob

 Mat I0 = M0 - B0

 Snd Header I0 M1 M2 F7

End

If M0 == B0 05 # B knob

 Mat I0 = M0 - B0

 Snd Header I0 M1 M2 F7

End

End Output Rules "Multiplier"

Input Rules

Input Rules "Multiplier"

If Load

 # Define Header F0 00 01 7E

 Alias L01 Value_A

 Alias L02 Value_B

 Alias L03 Value_A*B

End

If M0 == Header

```
If M4 == 00 04 # A
    Ass Value_A = M6
    Mat Value_A*B = Value_A * Value_B
    Snd Header 10 01 Value_A*B F7
    Block
End
If M4 == 00 05 # B
    Ass Value_B = M6
    Mat Value_A*B = Value_A * Value_B
    Snd Header 10 01 Value_A*B F7
    Block
End
End

## End Input Rules "Multiplier" ##
```

Adder

- The output of two controls is added

Logic

- When value C or D is received, it is stored, added by the stored value of the other input, and sent out in SysEx
- Max MIDI value is 127, so A and B are limited to max value 64/63
- C, D and C+D are CCs, the relay is SysEx

Output Rules

Output Rules "Adder"

If Load

 # Define Header F0 00 01 7E

End

If M0 == B0 0A # C knob

 Mat I0 = M0 - B0

 Snd Header I0 M1 M2 F7

End

If M0 == B0 0B # D knob

 Mat I0 = M0 - B0

 Snd Header I0 M1 M2 F7

End

End Output Rules "Adder"

Input Rules

Input Rules "Adder"

If Load

 # Define Header F0 00 01 7E

 Alias L06 Value_C

 Alias L07 Value_D

 Alias L08 Value_C+D

End

If M0 == Header

```
If M4 == 00 0A # C
    Ass Value_C = M06
    Mat Value_C+D = Value_C + Value_D
    Snd Header 10 02 Value_C+D F7
    Block
End
If M4 == 00 0B # D
    Ass Value_D = M06
    Mat Value_C+D = Value_C + Value_D
    Snd Header 10 02 Value_C+D F7
    Block
End
End

## End Input Rules "Adder" ##
```

Memory

- Several control values is entered.
- When the respective load button is pressed, the value is sent to the output.

Logic

- The memory, load, and relay are SysEx
- When a memory is changed, the value is stored
- When the load button (momentary) is pressed, the stored value is sent
- An initialize button is provided to send current memory values
- The Memory and Load SysEx is blocked in the input SB to prevent the condition 2 loop, their MIDI receive is also off.

Output Rules

Output Rules "Memory"

None needed, all controls use defined SysEx

End Output Rules "Memory"

Input Rules

Input Rules "Memory"

If Load

 # Define Header F0 00 01 7E

 Alias L09 Mem_1

 Alias L0A Mem_2

 Alias L0B Mem_3

 Ass Mem_1 = 0

 Ass Mem_2 = 0

 Ass Mem_3 = 0

End

If M0 == Header

 If M4 == 10 05 # Mem_1

 Ass Mem_1 = M6

 Block

 End

 If M4 == 10 06 # Mem_2

```
        Ass Mem_2 = M6
        Block
    End

    If M4 == 10 07 # Mem_3
        Ass Mem_3 = M6
        Block
    End

    If M4 == 10 08 # Load_M1
        If M06 != 0
            Snd Header 10 0B Mem_1 F7
        End
        Block
    End

    If M4 == 10 09 # Load_M2
        If M06 != 0
            Snd Header 10 0B Mem_2 F7
        End
        Block
    End

    If M4 == 10 0A # Load_M3
        If M06 != 0
            Snd Header 10 0B Mem_3 F7
        End
        Block
    End
End

## End Output Rules "Memory" ##
```

16 Step Memory

- The memory array is expanded from 3 to 16 locations by two bytes, but the code is simplified using implied storage locations and indirect referencing.
- This can be used to set up flexible sets of program changes (think multiple set lists), organ combinations, drum kit patterns, etc., - any MIDI data you want to step through and have multiple sets available.
- 16 steps can be expanded up to 128 x 3 bytes, the number of presets is practically unlimited.
- When a prog value is changed, the value is stored
- When the recall button (momentary) is pressed, the stored value is sent
- The recall buttons are grouped, with steppers, to step through the memory locations
- A Preset picker is added to store multiple sets of programs

Logic

- The prog, recall, and relay are SysEx
- Memory locations L40 - L5F are used for storage
- Program button sysex format is (header) 10 10 xx V, where xx is the memory location, and V is the two byte value to store
- Recall button SysEx format is (header) 10 11 xx yy, where xx is the lower memory location, yy is xx+1 (we could calculate $xx + 1$ in the inbound rules, but this saves a step)
- The Memory and Load SysEx is blocked in the input SB to prevent the condition 2 loop, their MIDI receive is also off.
- To initialize, toggle the recall preset button.
- To reset to the parked position (Recall 0), ensure Recall 0 is selected when the preset is stored.
- To drive additional two byte controls, we rescale MIDI output to 0-16383 after the relay to drive other 1000 value two byte controls.

Output Rules

Begin Output Rules "16 Step Memory"

None needed, controls use defined SysEx

End Output Rules "16 Step Memory"

Input Rules

Begin Input Rules - "16 Step Memory"


```
## Undeclared variables
## L40 - L5F - memory storage, two bytes per item
## L40, L41 are memory 1
## ...
## L5E, L5F are memory 16
## L0C - temp for calculating second storage byte

## Memory Store - Header 10 10 Loc MSB LSB
## Loc tells where to store 40, 42, .. 5E

## Memory Recall - Header 10 11 Loc
## Recall tells where to recall 40, 42, .. 5E
## We also need the Location +1, which is calculated

## Memory Send "Header 10 12 MSB LSB

If Load
    # Define Header F0 00 01 7E
End

If M00 == Header
    If M04 == 10 10
        If M06 >= 40 # Location 1
            If M06 <= 5E # Location 16
                Ass LM06 = M07 M08
                Block
            End
        End
    End
End

    If M04 == 10 11
        If M06 >= 40 # Location 1
            If M06 <= 5E # Location 16
                Mat L0C = M06 + 1
                Snd Header 10 12 LM06 LL0C F7
                Block
            End
        End
    End
End

## End Input Rules - "16 Step Memory" ##
```

Ganged Controls

- A primary control adjusts secondary controls, maintaining relative offsets
 - Example - individual track mixer levels with a group super control
- Move secondary controls to set relative values
 - Values are stored when primary control has not been moved for 200 ms
- Move primary control to adjust all, maintaining relative values
 - If they saturate, they will regain offsets as you move away from saturation

Notes

- This was the challenge I was trying to solve that led to developing the feedback control approach.
- Rev 1 is simpler overall. While the SB code is more complex, the layout controls are greatly simplified.

Logic

- When the primary control is moved
 - Timer is reset to zero
 - Recalc flag is reset
 - The secondary control delta values are added to the primary value and transmitted
- When a secondary control is moved > 200 ms after the primary control is moved
 - The primary control is reset to default (40h)
 - Updated delta values are stored for all controls (to account for the reset)
 - Recalc flag is used to ensure this only happens once
 - The moved secondary control delta is updated
- MIDI values are seven bit, SB uses 16 bit values, delta values calculated in twos complement
 - We have to watch wrap-around, since MDP2 only uses the least significant eight bits, (e.g, 0107h would display as 7)
 - Above 8000h, (i.e. negative number), send 0
 - Above 7F, (positive number above max display), send 127

Output Rules

Output Rules "Ganged Controls"

If MT == B0 # Control Change

If M1 == 07 # Volume

If MC <= 7 # Channels 1-8 # Note - this is not MOC

```
        Mat I0 = M0 - B0
        Snd Header I0 M1 M2 F7
    End
End
End
```

End Output Rules "Ganged Controls"

Input Rules R1 - No Enable Switch

```
## Input Rules "Ganged Controls" Rev 1 ##
## Aug 30, 2022
## RedHeronMusic.com
## Updated to remove need for enable control
## & genericize logic for up to 16 channels
```

```
## Variables
## L10 Primary Control Value
## L11 Recalc Complete?
## L12 Pointer to delta value (20-2F) # doubles as out message ID
## L13 Pointer to output value (30-3F)
## L14 Loop Counter
## L15 New Delta
## L20-2F Delta values - Ch 1-16
## L30-3F Output values - Ch 1-16
## T0 time since last primary control movement
## P00 Timer Value
```

```
## It would be cleaner to use aliases for L12 & L13,
## but aliases seem to fail for indirect lookup.
## LL12 works, LDelta_ptr fails
```

```
If Load
    # Define Header F0 00 01 7E
    Alias L10    Primary_02 # Supercontrol
    Alias P00    TimeSince # supercontrol last moved
    Alias L11    Recalc
    Alias L15    NewDelta
    Define DeadTime $200 # dead time in decimal ms
    Define MaxChan 07 # Nr of channels 0-F
    Ass Recalc = 0
End
```

If M00 == Header

```
If M04 == 10 13 # Detected primary control
    Ass Primary_02 = M06
    Ass TimeSince = T0 # dummy read to reset
    Ass TimeSince = 0
    Ass Recalc = 0
    Ass L12 = 20
    Ass L13 = 30
    Ass L14 = 0
    While L14 <= MaxChan # loop to send all eight values
        Mat LL13 = Primary_02 - LL12
        If LL13 > 8000 # negative value, clamp at 0
            Snd Header L12 07 00 F7
        Else
            If LL13 > 7F # clamp at 127
                Snd Header L12 07 7F F7
            Else # actual value
                Snd Header L12 07 LL13 F7
            End
        End
        Mat L12 = L12 + 1
        Mat L13 = L13 + 1
        Mat L14 = L14 + 1
    End
    Block # finished with Primary_02 message
End
```

```
If M04 <= MaxChan
    If M05 == 07 # Detected channel vol cc
        Mat TimeSince = TimeSince + T0
        If TimeSince > DeadTime # Dead time exceeded

            ## recalc deltas for primary reset
            If Recalc == 0
                Mat NewDelta = 40 - Primary_02
                Ass L14 = 0
                Ass L12 = 20
                While L14 <= MaxChan
                    Mat LL12 = LL12 + NewDelta
                    Mat L12 = L12 + 1
                    Mat LL14 = L14 + 1
```

```

                                End
                                Ass Recalc = 1
                            End

                                ## reset master control
                                Snd Header 10 13 40 F7
                                Ass Primary_02 = 40

                                ## calculate and store delta
                                Mat L12 = 20 + M04
                                Mat LL12 = Primary_02 - M06
                            End
                        Block # finished with channel vol cc
                    End
                End
            End
        End

## End Input Rules "Ganged Controls" ##
```