Behringer's Secret ?

Whenever you edit a preset on the BC (BCR-200 or BCF-200) you end up with a controller value that uses **.easypar** .(except for when a Button 'learns' a MIDI message.) That is great for most things like continuous controllers (CC), patch change (PC) , Notes, MIDI Machine Control (MMC) or Non –Registered Parameter Number (NRPN) – GS/XG is just a group of CCs or NRPNs associated with the Roland GS and the Yamaha XG standards.

.tx on the other hand can do all of the **.easypar** functions plus a lot more, but you need a PC (including Mac) to edit this command.

The learn function in the BC can only be used on a button and it will send just that message. When you dump the preset you will see a line something like
.tx $F0 $41 $17 $22 $00 $F7 (this is a made up sysex. The $ means hexadecimal)
This will output the same sysex message that it received every time the button is pressed.
$F0 $41 $17 $22 $00 $F7
$F0 $41 $17 $22 $00 $F7
$F0 $41 $17 $22 $00 $F7
$F0 $41 $17 $22 $00 $F7

Using a PC editor you can put anything after the **.tx** command and that is what is sent. The big thing about **.tx** is you can also attach it to an encoder or fader.
.tx $B0 7 8 is Volume CC on channel 1 value 3. Turning the encoder you will get
$B0 7 8
$B0 7 8
$B0 7 8
$B0 7 8
$B0 7 8
$B0 7 8
Not a great deal of value.

Normally, to send volume change on channel 1 you use **.easypar**
.easypar CC 1 7 0 127 absolute

But to send the value of the encoder you use the word **val** in the **.tx** statement
.tx $B0 7 val is Volume CC on channel 1, where the value (val) is the encoder position.
Turning the encoder you will get
$B0 7 0
$B0 7 1
$B0 7 2
$B0 7 3
$B0 7 4
$B0 7 5
Just like the **.encoder**, but why bother?

Well now you can use sysex…
.tx $F0 $41 $17 val $F7 now you have a sysex message that can vary.
$F0 $41 $17 0 $F7
$F0 $41 $17 1 $F7
$F0 $41 $17 2 $F7
$F0 $41 $17 3 $F7
$F0 $41 $17 4 $F7
$F0 $41 $17 5 $F7
$F0 $41 $17 6 $F7
I have read on many web forums that this is impossible. Try it for yourself and see if it is.

So, the word **val** is used for the 7bit value of the controller.

Multiple Messages

The BC units don't care what numbers you put in the **.tx** message. It will just chuck it out those numbers, so you do have to be a bit careful and you do have to know a bit about MIDI messages.
On the upside, a single **.tx** command can have more than one MIDI message in it.

Let's say the trumpets have different notes to play compared to the saxes so they are placed on a different MIDI channel.
.tx $B0 $07 val $B1 $07 val makes the encoder become a volume control on channel 1 *and* channel 2.
You can have a single controller that will adjust the volume of the saxes and the trumpets at the same time. The BC puts out 2 MIDI messages each time
$B0 7 0
$B1 7 0
$B0 7 1
$B1 7 1
$B0 7 2
$B1 7 2
$B0 7 3
$B1 7 3
$B0 7 4
$B1 7 4

You could also add another message on channel 3 for the trombones and have a single encoder/fader for the whole brass section.

A better idea is to set the volume (CC 7) for each channel to get the balance between instruments and then use CC 43, expression, joined together .
.tx $B0 43 val $B1 43 val $B2 43 val

You can play notes with the BC, but this is really useful for drums.
.tx $99 38 100 $99 38 0 is a very short note with the snare on channel 10
Note-On is $90 + midi channel, but we start the channel count at 0
Drums are channel 10, but Channel 10 is really number 9 therefore $99
It has 2 data bytes following it Pitch (60 is middle C) and velocity.
You can turn off a note by sending Note-On with the same pitch number, but velocity = 0

Put this on a button and you have a drum pad. Put it on an encoder (or fader) and you have a crazy drum roll

You can have up to 125 bytes in the **.tx** message

Resolution

.resolution is not accessible directly on the BC either. It sets the number of times a MIDI message is sent out per revolution of the encoder.
The normal value is 96, so to get from 0 to 127 you have to turn the encoder about 1 and a third times round.
You can adjust this so you have to turn it farther to make it sent out a message or adjust it to be more sensitive.
If you make it a value of 10 means there will only be 10 MIDI messages sent out per revolution, tedious but easy to select a value.
If you make it 2000 you will get to your value very quickly, but it will be very hard to select a particular value.

Behringer thought of this and designed it so the resolution can change depending on what speed you turn the encoder so you can have the best of both worlds.
.resolution 96 96 96 96 is the default value
There are 4 speeds.
The first number if for the slowest turning and the last number is for the fastest turn with the other two inbetween.
Initialising a preset sets an encoder at **.resolution 0 0 0 0**
That is, don't ever put out a value.

If you had a synth that had many banks of patches, set the encoder to
.resolution 10 50 100 500 for a patch select **.tx** message.
As you'll see later you can get the BC to send out bank change and patch change messages.
With this resolution you will jump past large groups of patches if you turn it quickly, but turning it slowly will make it easy to select the correct patch when you are close.
A full range of hundreds of presets all on one control !

You could also try the above **.resolution** with the snare **.tx** message – a bit odd?

minmax

In the **.easypar** command the range is set with the last two numbers.
In **.tx** you need to use another command.
.minmax . Again, this is only available with a PC editor.
.minmax 0 127 is the standard value – turning anticlockwise the **val** number is decreased until it gets to 0.
Swapping the numbers reverses the control.
.minmax 127 0 turning anticlockwise the **val** number is increased until it gets to 127

The values can be up to the maximum 14bit number.

Big Numbers

What happens if you need a number greater than 127?

Pitch bend, for example. If it used 0 to 127 it would only have 63 values for the up pitch bend and 63 values for the down bend.
This is OK if it is only bending 2 semitones but if it is bending an octave there will be noticeable jumps in pitch.
So the pitch wheel uses 2 data bytes for it value.

A byte has 8 bits, bit 0 to bit 7 (0 = cleared or 1 = set).
A MIDI status byte – i.e. what sort of MIDI message it is - always has bit 7 set
A MIDI data byte – i.e. what value - always has bit 7 cleared.
A data byte uses the lower 7 bits to determine its value
0 to 127 ($7F) or 0111 1111 (bit 0 is the rightmost 1, bit 7 is 0)
To get a value greater than 127 we join two data bytes together. The rightmost 7 bits from each byte gives a 14 bit number
0111 1111 and 0111 1111 becomes 0011 1111 1111 1111 the top 2 bits (bit14 and bit15) are ignored
So the 14bit number range is 0 to 16383 ($3FFF) or 0011 1111 1111 1111 (this is just 14 data bits in a 16 bit number – the 2 top (leftmost) bits are always 0)

The two bytes that make up the 14 bit number are what MIDI calls a course adjustment and a fine adjustment.
The byte that does a course adjustment is called the Most Significant Byte (MSB) and the fine adjuster is called the Least Significant Byte (LSB)

The BC units deal in 14 bit values and when only 7bits are required they output just the LSB. So **val** is the LSB of the encoder value
You can set the **.minmax 0 256** and the **val** will start back at 0 after reaching 127, halfway along according to the led dots

The MIDI standard gives the Pitch Wheel its own status byte instead of being a CC
$E0 LSB MSB - pitch wheel on channel 1

So we need a way to access this 14 bit number.
The BC does this with variations of **val.**
val0.6 is the same as **val,** but makes it clear that it is bits 0 to 6 (the lower 7 bits of the 14bit number)

Redundant, but useful in seeing what is really going on.

val7.13 bits 7 to 13 is the MSB (remember we start at bit 0)
So the Pitch Wheel command would be.
.tx $E0 val0.6 val7.13
But you would need to set **.minmax 0 $3FFF**
You could trick this message up with an extra MIDI message to provide a crescendo as you bend up and decrescendo when you bend down.

.tx $E0 val0.6 val7.13 $B0 $07 val

But this wouldn't work as the **val** would jump back to 0 many times before you got to $3FFF. Try this instead

.minmax 0 127
.tx $E0 0 val0.6 $B0 $07 val

and this would get louder as you bent the note up and softer as you bent the note down, but the resolution is now low. **val** is applied to the MSB of the pitch bend.

Just a Nibble

It was once common for more than one of the synths parameters to be packed into a single byte of data. (Common in early synths when memory was expensive) For example the envelope attack could be in bits 0 to 5 and bits 6 and 7 could be the octave of the oscillator.
The BC can output just bit parts of the 14bit value.
The bits are numbered from the right to the left

val0 is the least significant bit of **val**
.tx $99 38 val0 the very quietest snare roll
$99 38 1
$99 38 0
$99 38 1
$99 38 0
etc

val0.3 is the lower 4 bits (nibble) of **val** so this will go through 0 to 15 and then start again (if the **.minmax** max number is greater than 15)
.tx $B0 38 val0.3
$B0 38 1
$B0 38 2
...
$B0 38 14
$B0 38 15
$B0 38 0
$B0 38 1
...
You could just limit **.minmax** but using **val0.3** and looping around can be used in conjunction with **val** or some other word.

If you take a 14 bit number 0010 1111 0110 1010 you can break it up by just taking a group of bits.
Numbering from the right to the left, say bits 4, 5 and 6 this is 110 in our case.
Place that group of bits at the bottom of the MIDI data byte we get 0000 0110 (= 6)

val4.7 bits 4 to 7 In the example this is 0110
Data value has a range of 0 to 16
You can use this bit busting for older synths that, say, had only sixteen presets, but had 16 banks of 16 patches.
The banks were sometimes changed with a sysex message and sometimes with the standard MIDI bank select The example has a made up sysex message to change the bank followed by a patch change message…
.tx $F0 41 17 val4.7 $F7 $C0 val0.3
$ F0 41 17 0 $F7
$C0 0
$ F0 41 17 0 $F7
$C0 1
$ F0 41 17 0 $F7
$C0 2
….
$ F0 41 17 0 $F7
$C0 15
$ F0 41 17 1 $F7
$C0 0
$ F0 41 17 1 $F7
$C0 1
…
This will give you the full range of patches on one control.

val1.7 bits 1 to 7 in the above 14 number this would be 011 0101
This one is a bit odd as the least significant bit is missing, so it is basically a divide by 2.
.tx $F0 val val1.7 $F7
$F0 0   0 $F7
$F0 1   0 $F7
$F0 2   1 $F7
$F0 3   1 $F7
$F0 4   2 $F7
$F0 5   2 $F7
$F0 6   3 $F7
$F0 7   3 $F7
You could use this to fade in a channel half as fast as another channel
.tx $B0 $07 val $B1 $07 val1.7

val8.11 bits 8 to 11 in our example 1111

val12.13 bits 12 and 13 the top (most significant) 2 bits of the 14 bit number.
In the example 10

The useful places for these different bit values are when used together to produce interesting combinations of values.

Checking the data

You can get the BC to generate checksums.
This is needed for editing synths with a sysex message that uses checksums, like most Roland gear.

A checksum is a calculation using part of the data sent in the sysex message. The answer to this calculation is used to verify that the data received is the same as what was sent.

There are a few ways to calculate the checksum and the answer is usually placed just before the end of the sysex message.

The upon receiving the sysex message the synth calculates the checksum itself and if it doesn't agree with the one that was sent, it ignores the whole message.

The BC units haves 3 different ways to calculate the checksum **cks-1 cks-2** and **cks-3**

cks-1 is the one used by Roland for all their synths in my studio
.tx $F0 $41 $10 $00 $6B $12 $1F $00 $20 $49 val cks-1 6 $F7 this adjusts the resonance on the filter (Tone 1) on a FantomX

Although the Fantom allows you to map 4 CCs to many of the parameters, with the BC you can have realtime patch editing of anything that can be controlled by a sysex message while playing live.

The number after the **cks-1** (6) tells the BC where to start the checksum calculation (numbering starts at 0). 6 in this case is $1F

cks-1 adds the numbers from $1F up to and including **val**
eg
let **val** = $1A
$1F + $00 + $20 + $49 + $1A = $A2
$A2 is too big for a data byte so clear bit7
$A2 - $80 = $22
It would be simple if the synth could just add the checksum as well and the message received was correct if the answer is $00
To get that number do this calculation $80 - $22 = $5E
So $5E is the checksum
$1F + $00 + $20 + $49 + $1A + $5E = $100 really $00 as it only looks at the LSB
So for a **val** of $1A the BC sends out
$F0 $41 $10 $00 $6B $12 $1F $00 $20 $49 $1A **$5E** $F7

cks-2 I'm not sure who uses this method, but using the FantomX example above
.tx $F0 $41 $10 $00 $6B $12 $1F $00 $20 $49 val cks-2 6 $F7
The BC sends out
$F0 $41 $10 $00 $6B $12 $1F $00 $20 $49 $1A **$22** $F7
Does this $22 look familiar?
Yes it is the first part of the **cks-1** calculation.
The synth adds everything up to the checksum, but not including the checksum, makes it a 7bit number and it should be the same as the checksum.

cks-3 Again I'm not sure who uses this method, but using the FantomX example above
.tx $F0 $41 $10 $00 $6B $12 $1F $00 $20 $49 val cks-3 6 $F7
The BC sends out
$F0 $41 $10 $00 $6B $12 $1F $00 $20 $49 $1A **$6C** $F7
This checksum calculation is based on exclusive-or (XOR)
XOR means the answer bit is 1 if either bit is 1, but 0 if both are 1 or both are 0
Eg 1010 XOR 0011 = 1001

$1F XOR $00 = $1F
$1F XOR $20 = $3F
$3F XOR $49 = $76
$76 XOR $1A = $6C

Note that bit 7 is always 0 because 0 XOR 0 = 0 so the bit 7 is always cleared

So much for the math.

"Just a jump to the left"

The BC can output different things depending on which direction you are turning the encoder (or fader?) It uses 2 words in the **.tx** statement.
The first word is **ifp** and any numbers that follow it are output when the encoder is turned in the positive direction, (clockwise)
.tx $B0 $07 val ifp $B1 $07 val
This is a volume control for channel 1 but sets the volume on channel 2 when turned clockwise

Tuning clockwise
$B0 $07 1
$B1 $07 1
$B0 $07 2
$B1 $07 2
$B0 $07 3
$B1 $07 3
$B0 $07 4
$B1 $07 4
$B0 $07 5
$B1 $07 5
Now turn anticlockwise
$B0 $07 4
$B0 $07 3
$B0 $07 2
Tuning clockwise again
$B0 $07 3
$B1 $07 3 note that the **val** jumps back
$B0 $07 4
$B1 $07 4
$B0 $07 5
$B1 $07 5

The other word is **ifn** and any numbers that follow it are output when the encoder is turned in the negative direction, (anti-clockwise)
.tx ifp $99 38 100 $99 38 0 ifn $99 36 100 $99 36 0
This is the drum example again, but with a twist. 38 is the snare, but 36 is the kick drum. Turning it clockwise plays the snare and anticlockwise the kick is played.

These words can also be used to emulate a jog/shuttle control (moving the cursor back and forward in a DAW)

"It's all relative"

At the end of **.easypar CC** and **.easypar NRPN** commands there are commands for relative offset. These are used primarily for software mixers according to the manual. Most can be duplicated with **ifn** and **ifp** words
Even so the **.tx** function has 3 relative words.
The first word is **reloffs** <argument> relative offset. This sets the 14 bit number <argument> + 1 on the positive turn (or button on value) or <argument> - 1 on the negative (anticlockwise) turn (or the button off value)
Anything before the **reloffs** word has the normal value. Every **val** after uses this new value.

Be careful with the 14 bit offset number. $1234 looks like it should put $12 in **val7.13** and $34 in **val0.6** but this is not correct
0001 0010 0011 0100 really splits up
x010 0100 in **val7.13** ($24)
x011 0100 in **val0.6** ($34)

.tx $B0 1 val reloffs $0080 $B0 2 val7.13 $B0 3 val0.6
Turn clockwise
$B0 1 $20
$B0 2 $01
$B0 3 $01
$B0 1 $21
$B0 2 $01
$B0 3 $01
$B0 1 $22
$B0 2 $01
$B0 3 $01
Turn anti-clockwise
$B0 1 $21
$B0 2 $00
$B0 3 $7F
$B0 1 $20
$B0 2 $00
$B0 3 $7F

Although I have used a 14bit argument a normal data byte is OK
.tx reloffs $23 $B0 3 val
positive
$B0 3 $24
$B0 3 $24
$B0 3 $24
$B0 3 $24
negative
$B0 3 $22
$B0 3 $22
$B0 3 $22
$B0 3 $22

The next ‘relative’ word is **relsign**.<argument> (can be a 14bit value)
The sign of a number (plus or minus) is usually marked by the most significate bit. So an 8bit number has bit7 cleared if it is a positive number and set if it is a negative number.
So 0011 1010 is +$3A and 1011 1010 is -$3A
Here we use 0100 000 (-1 in a 7 bit number)
You can set where the negative bit is in the argument .
.tx relsign $0040 $B0 7 val
Turn positive
$B0 7 $01
$B0 7 $01
$B0 7 $01
$B0 7 $01
Turn negative
$B0 7 $41
$B0 7 $41
$B0 7 $41
$B0 7 $41
You are not restricted to just one bit. You could use 0111 0000
.tx relsign $0070 $B0 7 val
Turn positive
$B0 7 $01
$B0 7 $01
$B0 7 $01
Turn negative
$B0 7 $71
$B0 7 $71
$B0 7 $71

The last relative word is **rel2s**. Theres are only 2 values $01 for positive and $3FFF for negative turning.
This is just flipping all the bits (2s compliment) when going negative.
.tx rel2s $B0 7 val
Turn positive
$B0 7 $01
$B0 7 $01
$B0 7 $01
Turn negative
$B0 7 $7F
$B0 7 $7F
$B0 7 $7F
If you use **val7.13**
.tx rel2s $B0 7 val7.13
Turn positive
$B0 7 $00
$B0 7 $00
$B0 7 $00
Turn negative
$B0 7 $7F
$B0 7 $7F
$B0 7 $7F

NOTE: for all the ‘relative’ words, when the BC encoder is turned quickly the values are not just the clockwise value and the anti-clockwise value you expect, but can be plus and minus 2 or more depending on how the **.resolution** is set. Very strange and slightly unpredictable. This also happens in the **.easypar** with the **relative** words at the end. The Synthesizer Wiki people think that this is the correct behaviour and have something to do with turning speed, but I think that it may be an error.

“Once again”

A few DAW controllers use single CC messages for moving the cursor through the song. The BC can emulate these controllers.
We have seen that using **.resolution** can make the encoder spit more of these MIDI message out the faster it is turned.
To increase this there is a word **ntimes** that outputs the message following the word more than once if it is turned quickly
.tx $B0 07 02 ntimes $B2 07 val
$B0 07 02
$B2 07 05
$B0 07 02
$B2 07 06
$B0 07 02
$B2 07 07
$B2 07 07
$B2 07 07
$B0 07 02
$B2 07 08
$B2 07 08
$B2 07 08
$B2 07 08
$B2 07 08

increment with .tx

The **.easypar CC 1 7 0 127 increment 1** increments the **val** every time you press the button.
There is a command that is the same as **increment** but for**.tx**, it is **incval** and it is used with the **.mode** statement.

Add a **.tx** message with the sysex message and **val** in it…

```
$button10
.showvalue on
.minmax 1 3          ;You need this line with suitable min and max values
.default 0
.mode incval 1       ;The number is the step size and can be negative to go backwards
.tx $F0 $41 $10 $00 $6B $12 $10 $00 $04 $00 $01 val cks-1 6 $F7
```

```
$F0 $41 $10 $00 $6B $12 $10 $00 $04 $00 $01 1  $6B $F7
$F0 $41 $10 $00 $6B $12 $10 $00 $04 $00 $01 2  $6A $F7
$F0 $41 $10 $00 $6B $12 $10 $00 $04 $00 $01 3  $69 $F7
$F0 $41 $10 $00 $6B $12 $10 $00 $04 $00 $01 1  $6B $F7
$F0 $41 $10 $00 $6B $12 $10 $00 $04 $00 $01 2  $6A $F7
etc
```

These are all the BCR/BCF script words that are associated with .tx that I know of and they came mainly from the German web site Synthesizer Wiki.
Please let me know if there are any errors in the document so I can correct them.

Royce Craven
Sunday, 13 April 2008