461-Debugging Netlinx Programs with Terminal or Telnet messages

This is often referred to as using "**SEND_STRING 0's**" in your code. Strings sent to device 0 (the master), port 1 or port 0, system 0 (the local system) will show up in a terminal or Telnet session once the programmer has typed in '**MSG ON**'<enter> at the prompt.

For example, if this line is in your code:

SEND_STRING 0,'some message'

your terminal will display something like:

(0100636312) some message

when the code executes.

The 13 characters to the left of the message are called the "**time stamp**" or "**tic time**". Also, by default, the master will append a carriage return and line feed to the end of each string.

If the master is **v2.10.81** or higher, there are additional **MSG ON** modes:

1. All messages with tic time. This is the same as '**MSG ON**' with no type.

2. **SEND_STRING 0** only (no tic time in front of string and no CR and LF appended).

3. **SEND_STRING 0** with tic time pre-pended to string. **Example:**

MSG ON 2<enter> sets the message mode to 2.

To turn off these messages type 'MSG OFF'<enter>.

The **SEND_STRING 0** message is obviously completely customizable - whatever is typed in will be what you are going to get.

This can be used as a way to see the value of variables at specific points in the code, by sending that information from that point, instead of stopping execution with a breakpoint. This is especially useful if programmers have to watch non-printable data bytes that only show as blank boxes in Windows programs, the **SEND_STRING 0** code can reformat the data as a string of printable **ASCII characters**.

The attached '**DEV_TO_STRING.axi**' and **debug.axi**' files contain subroutines useful for this type of debugging and other purposes, e.g. **DEV_TO_STRING** and **STRING_TO_DEV** are useful for passing **DEV** information between masters via a virtual device in a master-to-master scenario. Theses subroutines were created as an **AXI** so programmers could use a **#INCLUDE** at the top of the program rather than having to dig through the old programs and find the appropriate code to cut and paste.

The subroutines include, but are not limited to:

1) **FUNCTION CHAR[17] DEV_TO_STRING (DEV dvDEV)** Formatting NetLinx device structures as strings take a little more than the simple **ITOA**(<**device**>) that was done in Axcess. This function takes a **DEV** and returns a

string in the '**<number>:<port>:<system>**' format. **<system>** will contain the actual system number, not 0, if the device is so defined.

2) **FUNCTION CHAR[25] DEVCHAN_TO_STRING (DEVCHAN dcDC)** This function returns a **DEVCHAN** as a string with brackets in the format '**[<number>:<port>:<system>,<chan>]**'. **<system>** will contain the actual system number, not 0, if the device is so defined.

3) **FUNCTION CHAR[25] DEV_CHAN_TO_STRING (DEV dvDEV, INTEGER nChannel)** Same as **DEVCHAN_TO_STRING** but takes a separate **DEV** and **INTEGER** as parameters. Returns '**[<number>:<port>:<system>,<chan>]**'. **<system>** will contain the actual system number, not 0, if the device is so defined.

4) **FUNCTION CHAR[25] DEVLEV_TO_STRING (DEVLEV dlDL)** This function returns a **DEVLEV** as a string in the format '**<number>:<port>:<system>,<lev>**'. **<system>** will contain the actual system number, not 0, if the device is so defined.

5) **FUNCTION CHAR[25] DEV_LEV_TO_STRING (DEV dvDEV, INTEGER nLevel)** Same as **DEVLEV_TO_STRING** but takes a separate **DEV** and **INTEGER** as parameters. Returns '**<number>:<port>:<system>,<lev>**'. **<system>** will contain the actual system number, not 0, if the device is so defined.

6) **FUNCTION printToDebug (dvDEBUG,STR1[],STR2[],nLine,nMode,nFormat) dvDEBUG** is the destination device for the string. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.… **STR1** is a header sent with each line - this is a good place to use **DEV_TO_STRING**. **STR2** is data. Data bytes **< $20** (ASCII control codes) and **> $7E** (~) are formatted as hex and separated by commas. As of **v2.01**, a **space** (**$20**) is no longer added between **STR1** and **STR2**. **nLine** is the line length. This may need to be changed if a terminal with a line length other than the typical 80 characters is being used. If not, just leave it as **0** for the defaults. If **nMode** is 2, **nLine** defaults to 80, else **nLine** defaults to 67 to allow 13 characters for the tic time. **nMode** is the mode, if "**MSG ON 2**" mode is being used to set it to 2; otherwise, use **0** for the defaults.

nFormat is the format desired for **STR2**. 5 formats are supported:

0 – PRINT_UNFORMATTED

1 – PRINT_ASCII_HEX Printable characters are printed with unformatted. Control characters and bytes >= $7F are formatted as 2 digit hexadecimal.

2 – PRINT_HEX All bytes are formatted as 2 digit hexadecimal.

3 – PRINT_DECIMAL All bytes are formatted as 3 digit decimal.

4 – PRINT_ASCII_EXT Control characters less than $20 are formatted as 2 digit hexadecimal. All other bytes are printed without formatting.

7) **FUNCTION printDEVICE_INFO (dvDEBUG,hdr,dvDEV)** This subroutine is designed format the results of a **DEVICE_INFO()** query into readable strings. **dvDEBUG** is the destination device for the strings. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.…

hdr is a header printed with each line of device info.

dvDEV is the device from which to get the info.

8) **FUNCTION printCUSTOM_EVENT (dvDEBUG,hdr,nLine,nMode)**

This subroutine is designed to be called inside of a **CUSTOM_EVENT** response to a G4 button status query. It will format the results of the query into readable strings. **dvDEBUG** is the destination device for the strings. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.…

hdr is a header printed with each line of custom event info.

nLine is the line length. This may need to be changed if a terminal with a line length other than the typical 80 characters is being used. If not, just leave it as **0** for the defaults. If **nMode** is 2, **nLine** defaults to 80, else **nLine** defaults to 67 to allow 13 characters for the tic time.

nMode is the mode, if "**MSG ON 2**" mode is being used to set it to 2; otherwise, use **0** for the defaults.

9) **FUNCTION printMidiDebug (dvDEBUG,STR1[],STR2[],nLine,nMode,nStatus)**

dvDEBUG is the destination device for the string. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.…

STR1 is a header sent with each line - this is a good place to use **DEV_TO_STRING**.

STR2 is data. MIDI channel message data bytes are formatted as 3-digit decimal, separated by commas. MIDI status bytes and data bytes for MIDI system messages are formatted as 2-digit hex, separated by commas. **nLine** is the line length. This may need to be changed if a terminal with a line length other than the typical 80 characters is being used. If not, just leave it as **0** for the defaults. If **nMode** is 2, **nLine** defaults to 80, else **nLine** defaults to 67 to allow 13 characters for the tic time.

nMode is the mode, if "**MSG ON 2**" mode is being used to set it to 2; otherwise, use **0** for the defaults.

nStatus is the current MIDI status byte. If the value is **$F0** or higher it is a MIDI system message and bytes **< $80** will be formatted as hex, otherwise they will formatted as decimal.

10) **FUNCTION STRING_TO_DEV (Txt[], dvDEV)**

Takes ASCII string **Txt[]** of the form '**<dev>**' or '**<dev>:<port>:<sys>**' and stores this as a **DEV**. The dev data in **Txt[]** can be embedded in other non-numeric characters. If **Txt[]** does not contain any numerals, **dvDEV** will default to **0:1:0** (the local master).

11) **Deprecated subroutines**

The subroutines below can still be used for now. They have been rewritten as wrappers for the functions that replaced them.

a) **CALL 'SEND 131 BYTE PACKETS TO MASTER' (sSTRING[]) deprecated**

Replaced by **printToDebug()**

Strings sent to the master will be truncated if they are longer than 131 bytes. This call breaks the strings up if they are longer than 131 bytes. Otherwise, it does not modify the string. Use with master **v2.10.81** or higher "**MSG ON 2**" mode to see only the data in the **sSTRING**. As of **v2.51**, this call no longer parses sString[] destructively, that is, it does not modifiy the array passed in.

b) **CALL 'SEND ASCII/HEX TO DEBUG' (dvDEBUG,STR1[],STR2[],nLine,nMode) deprecated**

Replaced by **printToDebug()**

dvDEBUG is the destination device for the string. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.…

STR1 is a header sent with each line - this is a good place to use **DEV_TO_STRING**.

STR2 is data. Data bytes **< $20** (ASCII control codes) and **> $7E** (~) are formatted as hex and separated by commas. As of **v2.01**, a **space** (**$20**) is no longer added between **STR1** and **STR2**. **nLine** is the line length. This may need to be changed if a terminal with a line length other than the typical 80 characters is being used. If not, just leave it as **0** for the defaults. If **nMode** is 2, **nLine** defaults to 80, else **nLine** defaults to 67 to allow 13 characters for the tic time.

nMode is the mode, if "**MSG ON 2**" mode is being used to set it to 2; otherwise, use **0** for the defaults.

c) **CALL 'SEND DECIMAL TO DEBUG' (dvDEBUG,STR1[],STR2[],nLine,nMode) deprecated**

Replaced by **printToDebug()**

dvDEBUG is the destination device for the string. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.…

STR1 is a header sent with each line - this is a good place to use **DEV_TO_STRING**.

STR2 is data. All data bytes are formatted as 3 digit decimal and separated by commas. As of **v2.01**, a **space** (**$20**) is no longer added between **STR1** and **STR2**.

nLine is the line length. This may need to be changed if a terminal with a line length other than the typical 80 characters is being used. If not, just leave it as **0** for the defaults. If **nMode** is 2, **nLine** defaults to 80, else **nLine** defaults to 67 to allow 13 characters for the tic time.

nMode is the mode, if "**MSG ON 2**" mode is being used to set it to 2; otherwise, use **0** for the defaults.

d) **CALL 'SEND DEVICE_INFO TO DEBUG' (dvDEBUG,dvDEV) deprecated**

Replaced by **printDEVICE_INFO()**

This subroutine is designed format the results of a **DEVICE_INFO()** query into readable strings. **dvDEBUG** is the destination device for the strings. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.…

dvDEV is the device from which to get the info.

e) **CALL 'SEND CUSTOM_EVENT DATA TO DEBUG' (dvDEBUG) deprecated**

Replaced by **printCUSTOM_EVENT()**

This subroutine is designed to be called inside of a **CUSTOM_EVENT** response to a G4 button status query. It will format the results of the query into readable strings. **dvDEBUG** is the destination device for the strings. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.…

f) **CALL 'SEND MIDI TO DEBUG' (dvDEBUG,STR1[],STR2[],nLine,nMode,nStatus) deprecated**

Replaced by **printMidiDebug() dvDEBUG** is the destination device for the string. It could be the local master, a virtual device on the local system (see TN435), an IP device, serial device, etc.…

STR1 is a header sent with each line - this is a good place to use **DEV_TO_STRING**.

STR2 is data. MIDI channel message data bytes are formatted as 3-digit decimal, separated by commas. MIDI status bytes and data bytes for MIDI system messages are formatted as 2-digit hex, seperated by commas.

nLine is the line length. This may need to be changed if a terminal with a line length other than the typical 80 characters is being used. If not, just leave it as **0** for the defaults. If **nMode** is 2, **nLine** defaults to 80, else **nLine** defaults to 67 to allow 13 characters for the tic time.

nMode is the mode, if "**MSG ON 2**" mode is being used to set it to 2; otherwise, use **0** for the defaults.

nStatus is the current MIDI status byte. If the value is **$F0** or higher it is a MIDI system message and bytes **< $80** will be formatted as hex, otherwise they will formatted as decimal.

g) **CALL 'STRING TO DEV' (Txt[], dvDEV) deprecated**

Replaced by **STRING_TO_DEV()**

Takes ASCII string **Txt[]** of the form '**<dev>**' or '**<dev>:<port>:<sys>**' and stores this as a **DEV**. The dev data in **Txt[]** can be embedded in other non-numeric characters. If **Txt[]** does not contain any numerals, **dvDEV** will default to **0:1:0** (the local master).

See the **debug.axi** and **DEV_TO_STRING.axi** source code for additional subroutines not listed here. Also see **TN875** - it includes a device debugging module based on these concepts.

In order to use these subroutines, **debug.axi** must be in the compile path. It is suggested that an "**AXIs**" folder in the **My Documents** folder be created, save **debug.axi** there, then add that path to NetLinx Studio "**Edit/Preferences/Netlinx Compiler Options/Include Files**". Then add the lines:

#INCLUDE 'DEV_TO_STRING.axi'

#INCLUDE 'debug.axi'

to the source code.