Märklin Digital Interface Commands

Binary Commands for the 6050, 6051 or 6023 Interface

Computers can be easily programmed to talk to the trains and switches through the Märklin Interface. Commands can be sent a number of ways, depending on the type of programming language being used and the interests of the programmer. Data can be sent as an ASCII string in BASIC such as: PRINT #1, CHR$(10), or as a hexadecimal number: PRINT #1,HEX$(10). Or, if assembly language routines are being used, bytes can be addressed as binary or hex notations; 0000 1010 (in binary) or 0A (in hex). The BASIC language command "PRINT #1" directs the output to the serial port on the computer if it is set as port #1. When using this command be sure to end the command with semicolon ";". Further information on this command and others are mentioned later in this chapter.

The Interface is a serial peripheral to the computer. This means it has a baud rate and specific communication parameters. These parameters need to be set in the computer software so the computer can "talk" to the Interface. The communications port needs to be set for:

Baud rate = 2400
Start bits (if requested by computer) = 1
Stop bits = 2
Parity = None
Word size = 8 bits

For the purposes of this web site, all notations will be given in decimal numbers with examples showing the commands written in Microsoft BASIC being sent out serially with ASCII notation. The reason for this is that Microsoft BASIC is the most popular form of the language being used with microcomputers. It was the form included with the early IBM and Radio Shack, and also sold for the Macintosh. It is very close to forms of BASIC being used for the Apple II, Commodore, and Atari.

Commands will be presented for all types of control operations that are possible from the Keyboard and Control 80 units. Each set of commands going out the serial port needs to be separated from other commands that are sent by a fraction of a second. If this is not done, the signals will jam up on the serial line and no action will result. For best results, it is advisable to use a loop command between commands. For example:

```
100 FOR X=1 TO 500
110 NEXT X
```

This short loop should be set into a subroutine in BASIC. Then the subroutine can be called at the end of each set of commands. This pattern will be followed in the examples that follow.

Engine Commands
Engines are controlled by sending a two-byte command to the serial port. The following is an example:

```
10 PRINT #1,CHR$(10);CHR$(1);:GOSUB 100
100 FOR X=1 TO 500
110 NEXT X
120 RETURN
```

In the earlier example, the first byte is the "10" in the command CHR$(10). CHR$ tells the computer that what is to be sent is the ASCII string for the decimal number "10". For those who

understand computer programming the 10 is the same as the byte 0000 1010, or the hex number 0A. The second byte is the "1" in the command CHR$(1).

When controlling locomotives, the two bytes are: 1) the command for speed and function in the first byte, and 2) the loco's address in the second byte. Loco speed can be any one of the 15 possible speed steps available, and the loco's address can be any one of the addresses from 1 to 80. For example: you want loco #26 to travel at a speed of 5. The command would be:

```
10 PRINT #1,CHR$(5);CHR$(26);:GOSUB 100

100 FOR X=1 TO 500
110 NEXT X
120 RETURN
```

Or if loco #2 is to travel at speed 13, the command would be:

```
10 PRINT #1,CHR$(13);CHR(2);:GOSUB 100

100 FOR X=1 TO 500
110 NEXT X
120 RETURN
```

Speed "0" is the same as "STOP", the loco will not move. Reverse is set as speed "15". It will not only reverse the loco, but it will stop it also. That makes the number 14 the fastest possible speed for the loco. The following will send loco #5 moving at the speed 10, reverse it, then start again at speed 4, then stop the loco.

```
10 PRINT #1,CHR$(10);CHR$(5);:GOSUB 100
20 PRINT #1,CHR$(15);CHR$(5);:GOSUB 100
30 PRINT #1,CHR$(4);CHR$(5);:GOSUB 100
40 PRINT #1,CHR$(0);CHR$(5);:GOSUB 100

100 FOR X=1 TO 500
110 NEXT X
120 RETURN
```

Notice in the above example, that each command is separated with the loop to protect the data from being sent too fast. The amount of time between commands can be adjusted down by making the number "500" smaller. Doing so will increase the operating speed of the program, but be careful not to make it so small that the data jams. Newer interfaces such as the one built into the Central Control-i Unit have both RTS and CTS signals and so they do not need the delay loops in each of the statements as seen in this chapter. Later information in this chapter concerns these interface units.

Functions

Functions such as smoke, lights or TELEX couplers can be activated along with the speed command. Simply add 16 to the command, and the function will be active also. For example, take the series of commands given in the last example - if the function needs to be active when the loco begins the first speed (which is 10) and it needs to stay active until the engine reverses in command line 20, the example would look like this:

```
10 PRINT #1,CHR$(26);CHR$(5);:GOSUB 100
20 PRINT #1,CHR$(15);CHR$(5);:GOSUB 100
```

```
30 PRINT #1,CHR$(4);CHR$(5);:GOSUB 100
40 PRINT #1,CHR$(0);CHR$(5);:GOSUB 100

100 FOR X=1 TO 500
110 NEXT X
120 RETURN
```

The only difference is that the speed "10" in line 10, has 16 added to it. The byte "26" tells the loco to turn on the function and travel at speed 10. If the function should be "on" through the entire series of commands, then 16 needs to be added to all of the first bytes. Notice that "stop" becomes 16 and "reverse" becomes 31.

```
10 PRINT #1,CHR$(26);CHR$(5);:GOSUB 100
20 PRINT #1,CHR$(31);CHR$(5);:GOSUB 100
30 PRINT #1,CHR$(20);CHR$(5);:GOSUB 100
40 PRINT #1,CHR$(16);CHR$(5);:GOSUB 100

100 FOR X=1 TO 500
110 NEXT X
120 RETURN
```

Other decoders used in 1 gauge locos have four other functions built into them and they can be called up with the Control 80f Unit, the Control Unit or through the interface. They can be called with the same two-byte system that has been discussed previously, but, the first byte is the signal to turn on the functions. The second remains the address of the loco or the digital car. There is no relationship to speed control with the first byte. It is independent from speed and original function control. Locos will continue with their previous commanded activity, but will employ the new function command. Similar four function decoders are also found in the HO Panorama and Dance cars. Other applications for these decoders will be added to the Märklin line in the future.

The following codes enable the four other functions of the special function cars or accessories. Each combination of the functions can be activated by one of the code numbers. For example, if you wanted the waiter to move up the aisle and also have the table and overhead lights on in the Panorama car, you would choose code 77 or 78. Each of these codes moves the waiter with function #1 and #2, and turns on lights with function #3 and #4. Of course, with the waiter, it is not possible to activate functions #1 and #2 at the same time. The waiter cannot move both up and down the aisle at the same time.

```
code functions on
64 all off
65 #1
66 #2
67 #1,#2
68 #3
69 #3,#1
70 #3,#2
71 #3,#2,#1
72 #4
73 #4,#1
74 #4,#2
75 #4,#2,#1
```

76 #4,#3
77 #4,#3,#1
78 #4,#3,#2
79 #4,#3,#2,#1

Any number not listed will turn off. For example "79" will turn on all functions, and if followed by "71", it will just turn off function #4. The others (#1, #2 and #3) will stay on. Commands will look like the following example which turns on functions #2 and #3 in loco 55.

```
10 PRINT #1,CHR$(70);CHR$(55);:GOSUB 100
100 FOR X=1 TO 500
110 NEXT X
120 RETURN
```

Turnout Commands
Turnout commands also need to be separated from each other with the loop subroutine as described in the previous section. Turnouts, however, when controlled with the 6050 interface need an extra command in the subroutine. The command is:

```
PRINT #1,CHR$(32);
```

The reason why this command is needed has to do with the solenoid in switches and signals. Once the solenoid is activated by a command from the computer, it stays on. If left on, it will burn out. The command PRINT #1,CHR$(32) simply turns the solenoid off. It doesn't need an address, since it will turn off only the last turnout called up. If a number of turnouts are called in sequence, then each command will automatically turn off the turnout called previously, which means that only the last one needs to be turned off. For examples given in this book, the "off" command is given for all switching operations, even if they are followed by other switching commands. It is best to include this "off" command in the timing loop which will look like this for switching operations:

```
100 FOR X=1 TO 500
110 NEXT X
120 PRINT #1,CHR$(32);
130 FOR X=1 TO 500
140 NEXT X
```

Notice that the loop is given twice, once before and again following the "off" command. The first provides a timing pause before the "off" and the second provides a timing pause before the next operation.

Switching is accomplished in a similar fashion to controlling engines. They take a two-byte command, the first one giving the turnout directions, and the second is the turnout address. Addresses are given simply as 1-256. There is no need to worry about Keyboard addresses, just remember to set your accessory decoder addresses correctly. For example, a k83 with the address 1(1) will control turnouts 1, 2, 3 and 4. The next k83 in line would be 1(2) which will control turnouts 5, 6, 7 and 8. Remember that each Keyboard controls 16 turnouts and each accessory decoder controls 4 of those 16 turnouts. The second Keyboard controls accessory decoder units 2(1) through 2(4), and the second set of 16 turnouts (numbers 17 - 32). The accessory decoder unit with the address 2(1) has turnout 17, 18, 19 and 20. And, k83 number 2(4) has turnouts 29, 30, 31 and 32. Only the turnout number needs to be given in the computer program.

Turnoutsettings are "33" to go straight and "34" for the curve or "branch" setting. An example of setting turnout no. 3 to the branch would be:

```
10 PRINT #1,CHR$(34);CHR$(3);:GOSUB 100
100 FOR X=1 TO 500
110 NEXT X
120 PRINT #1,CHR$(32);
130 FOR X=1 TO 500
140 NEXT X
```

Setting the turnout back to the straight position would look like this:

```
10 PRINT #1,CHR$(33);CHR$(3);:GOSUB 100
100 FOR X=1 TO 500
110 NEXT X
120 PRINT #1,CHR$(32);
130 FOR X=1 TO 500
140 NEXT X
```

When connecting uncouplers to the accessory decoders, you will connect one to the red side and one to the green side. This means that when you want the one on the green side to activate, you need to send the "33". To activate the red one, you send "34". When working with uncouplers you will not want to send the "off" command right away. The uncoupler needs to be raised for a long enough period of time to do its work. Try out higher numbers in line 100 of the example above to see how high it needs to be. It may need to go as high as 5000 or even higher. Experiment with it.

Newer interface units such as the 6023 do not need the solenoid off command (CHR$(32)) because the interface automatically sends the off command after switching commands have been sent.

System Commands

The Digital System can be started and stopped from the Control 80 Unit. The computer only needs to send a one-byte command to initiate either action "go" or "stop". "Go" is sent with the number "96" and "stop" uses "97".

The system would be started with this series of commands:

```
10 PRINT #1,CHR$(96);:GOSUB 100

100 FOR X=1 TO 500
110 NEXT X
```

And the system can be stopped with this command:

```
10 PRINT #1,CHR$(97);:GOSUB 100

100 FOR X=1 TO 500
110 NEXT X
```

Track Detection Modules

The s88 Decoders store data sent from the track detectors or the reed switches. Each s88 has room to store two bytes of data (16 separate contacts). In order for these decoders to be read by the computer, they must first be told to "dump" their memories into the computer. Two options

are available for requesting this "dump", depending on how you want the s88 units to feed their data into the computer.

Read one s88 unit - If only one s88 is to be read, you send the code "192" PLUS the number of the unit to be read in. For example: you only have one s88 on the layout - you read it with the command "193" (192 +1). On the other hand, if you had 4 units on the layout and you wanted to read only number 3, then send the code 195 (192+3). The command would be:

```
10 PRINT #1,CHR$(195);.....
```

The command is not finished because more instructions need to be given to the computer so it can accept the incoming data, but that will be covered later.

Read many s88 units - The other option for "dumping" s88 memory is for getting all the s88 units to dump memory up to a specific unit. For example: four s88 units are on the layout and you want information from the first 3 only. The command for this type of memory dump is the number 128 PLUS the number of the last unit to be read. In this case the command would be "131" (128+3). The command would look like:

```
10 PRINT #1,CHR$(131);.....
```

In this case, the first s88 on line would dump its memory, then the second, and finally the third unit connected in the series.

When any of these commands are given, the s88 units will send data back to the computer. The program needs to be alerted so it can receive this data and report it back to the computer operator in some readable fashion. Most BASIC programs require that two communications lines be opened in order to both send and receive data. The opening lines on a program would be:

```
10 OPEN "COM1:2400,N,8,2"FOR OUTPUT AS #1
20 OPEN "COM1:2400,N,8,2"FOR INPUT AS #2
```

The parameters mentioned earlier are seen in these statements. Notice the 2400 Baud, No parity, 8 data bits, 2 stop bits. One line is set for output and numbered #1. That is why all the print statements seen earlier said PRINT #1. They were only sending data on the output line. The other line is for input and is numbered as #2.

If data is expected, then it should be assigned a variable and the computer should be alerted immediately after sending the code to dump memory. The program would look like this:

```
10 OPEN "COM1:2400,N,8,2"FOR OUTPUT AS #1
20 OPEN "COM1:2400,N,8,2"FOR INPUT AS #2
30 PRINT #1,CHR$(193);:a$=INPUT$(2,#2)
```

Line 30 sends the command CHR$(193) out on #1 telling s88 number 1 to dump memory. That memory will come into the variable "a$". It will be assigned as a result of the command a$=INPUT$(2,#2). The first number 2 means that there will be 2 bytes coming in and the #2 means it is on the INPUT format as defined in statement 20. Those two bytes can now be separated and printed on the screen to let you know the status of the s88. This command will do that:

```
30 PRINT #1,CHR$(193);:a$=INPUT$(2,#2)
40 contact=ASC(LEFT$(a$,1)):PRINT contact
50 contact2=ASC(RIGHT$(a$,1)):PRINT contact2
```

A new variable called "contact" will be assigned the leftmost single byte of the two bytes (this is the first one from sockets 1-8 on the s88 unit) and then will be printed to the screen. Line 50 will take the second byte (sockets 9-16 on the s88) and assign it to variable "contact2" and print it.

The data printed to the screen can be confusing unless you understand how binary works. Each of the sockets stand for a binary number in the sequence 1, 2, 4, 8, 16, 32, 64 and 128. They are assigned as follows:

sockets binary number
1 = 128
2 = 64
3 = 32
4 = 16
5 = 8
6 = 4
7 = 2
8 = 1

If only the contact connected to socket #8 was triggered, then the number printed on the screen would be "1". If only #3 was triggered, then it would be "32". If multiple numbers are triggered, then their assigned binary numbers are added together. For instance, contacts #7 and #8 would be reported as a "3" (2+1), while #1 and #8 would be 129 (128+1).

The second side of the s88 is read the same way, only it will be in the second byte of data received. Contact #9 would be the same as #1 on the opposite side of the s88 and socket #16 is the same as #8 (they are both "1"). Programs can be written that will analyze these number totals and tell you the contacts that were active. It is easy to write such a program. If the number is greater than 128 then you know that contact #1 has been triggered. If this was true, then subtract 128 from the number and check it again. If it is now greater than 64, that means contact #2 was triggered. If that's right, subtract 64 and check the next number, etc.

The s88 units can be told to reset its memory after a dump to the computer, or it can remain as it was and continue adding data to the byte. A reset would make each position in memory a "0" again. If you want it to reset, send the single byte 192 without adding any numbers to it. If the reset mode is to be "off", then send the code 128. For example, the following would set the reset mode "on":

```
10 PRINT #1,CHR$(192);
```

Don't forget to also add the time delay loop with these commands, unless your working with the newer interface with both RTS and CTS signals.

ASCII Commands on the 6023 Interface

The binary format used in the 6051 interface is still understood by the interface in the 6023 Central Control-i Unit, plus it has some added features. In the ASCII format, letters and decimal numbers are expected alternately (i.e. L 15 S 5 F 1). The numbers can range from 0 to 256, blank characters are ignored but can be listed for clarity. Only numbers, letters, blank characters and carriage returns (CR) will be accepted by the 6023 interface. Capital letters and small letters are treated equally and are returned just as they are entered in the echo modes. A command will be interpreted only after the reception of the carriage return (CR) and executed if possible.

Commands consist of a maximum of 6 elements (letters or decimal numbers). The string entered into the computer can be as long as you want, but only the first 6 elements will be

executed. Only one command line per entry is possible and linking strings is not possible. Processing the data with the computer is based on the fact that only one line is being sent or received (BASIC: Print #1,"string", Input #2, variable$ or Pascal: WRITELN, READLN). Data being received in the serial buffer is treated as a character string with variable lengths depending on the command that generated the data coming in.

s88 Modules
Up to four s88 modules can be accessed with the 6023 interface. If these are constantly being monitored, their data is stored in the memory of the Central Control-i Unit. When the computer requests data from the s88 units there is an immediate response from the memory of the 6023 rather than waiting for the s88 to respond. Access time is reduced by this function and status of the other contacts is not lost through the inquiry. It is possible to tell the Central Control-i Unit to watch for specific contacts to be activated, either turned on or off, and to report this change. This command only works in the echo mode since the data has to be "echoed" back to the computer.

After logging on only the first s88 module is activated. If others are to be monitored, especially in the Watch and Job modes, they must first be requested (with the D# command in ASCII mode).

Binary Mode
The binary mode of the interface of the Central Control-i Unit still recognizes the binary commands as used in the 6051 interface. The solenoid command was modified, however. When the solenoid (in a switch or signal) is activated, the interface will automatically send the shut off signal after about 200 milliseconds. This prevents accidentally leaving the solenoid on thereby causing the switch unit to run hot and possibly burn out. The time span before sending the off commands can be altered up to 12.5 seconds for older switches and uncouplers which may need the extra time to activate.

The s88 commands to reset (128 and 192) are no longer used. Once data is sent from the s88 to the interface that data is stored in the interface and only changed if the s88 reports new data at that contact. Resetting is not necessary any longer. Also only four s88 units can be monitored, requests for units 5 through 31 will be ignored.

Error reporting commands that are accessible in the ASCII mode (E1 or E2) will not work in binary mode. During an emergency stop, commands going through the Central Control-i Unit to the track will be ignored. Commands to other units (i.e. s88 monitoring) can still be executed.

ASCII MODE Short overview of all ASCII commands

X - invalidate this command string

S {CR} - emergency stop

G {CR} - clear, go

V {CR}- speed control is echoed to serial buffer use echo mode 2

L # S # [F #] {CR} - engine address, speed and function command loco 1…80 speed 1…14 function 0…1

L # D {CR} - direction control

M # R [time] {CR} - magnetic solenoid, time in 0.1 seconds

M # G [time] {CR} - no time = 200 ms default time=0, solenoid is not turned off

M {CR} - magnetic solenoid turned off

D # {CR} - define number of s88 to be active

A # {CR} - read one s88 in "10001111" format echo modes

C # {CR} - read one contact of the s88 use echo modes

R # {CR} - remove s88 memory from storage A, D = 1...4 Contact 1...64

J # W # {CR} - job x to wait for 1 to 12.7 seconds

J # W # P/N {CR} - wait for contact to be positive or negative p=pressed, n=released, use echo modes

K # {CR} - kill job / job number 1...3 contact 1...64

W # {CR} - send all s88 changes to serial buffer. Watch must use echo modes

Q {CR} - ASCII mode off, binary mode on

E # {CR} - echo modes activated 0/1/2 / echo mode 0...2, this invokes ASCII modes