User's Guide

DeviceNet™ Master Emulator

DN-EMU

Pub # 2505003 Revision 1.07 13 April 2006

1. GETTING STARTED

1.1. General Description

The DN-EMU Emulator is an application for the Huron Net Works DeviceGate hardware. It is a general purpose tool for use in the development and debugging of DeviceNet nodes.
The emulator monitors DeviceNet bus traffic, and generates messages for configuring or testing nodes on the bus.

The DeviceGate hardware is a DeviceNet-to-Ethernet interface. The DeviceGate contains a single chip PC with a DOS-like file system in flash memory, plus a web server and TCP/IP stack. The emulator application transfers DeviceNet data to ethernet using HTML and XML via the web server.

A personal computer workstation with an ethernet card and an internet browser is used to connect to the DeviceGate and provides the user interface to the emulator. The DeviceGate is configured with an IP address and this address is entered into the browser to access the emulator.

The emulator application consists of a set of files that reside in flash memory on the DeviceGate. A combination of executable code, Javascript and HTML is used to define the user interface. The user who is familiar with writing HTML and JavaScript will be able to add customized featrues. This manual will give examples of how to do this.

1.2. Requirements

1. A DeviceGate interface with pre-installed emulator files. The files are:

   - hnw.css — A style sheet for the configuration web pages
   - emu.css — A style sheet for the emulator web pages.
   - dmem.exe — The emulator executable
   - autoexec.bat — For autostart of the application
   - chip.ini — Configuration info such as the IP address
   - emu.js — Javascript for basic emulator web pages
   - emu1.htm — Emulator web page
   - emu2.htm — Emulator web page
   - example.htm — Example page to demonstrate customization
   - example.js — Javascript for user customizable features

2. A 10 Mb ethernet network with a twisted pair connection for the DeviceGate. Alternatively, a point-to-point connection can be made using a cross-over ethernet cable.

3. A DeviceNet connection with 24 v DC power. The DeviceGate receives its power from the DeviceNet connection.

4. The CD ROM with DG tools install software and backup copies of the application files.

5. A PC with

   - a 10 Mb ethernet card.
   - An internet browser, either Netscape 6.2 or Internet Exlorer 5 (or later).
   - DeviceGate IP configuration tool.

1.3. Installation

The DeviceGate ethernet interface will need to be assigned an IP address that can be used for access by your browser. This address will be loaded into the DeviceGate using the IP configuration tool provided on the CD-ROM.

Install the configuration tool as a program on your PC by running setup.exe in the DGTools folder of the CD-ROM. This will create an item in your startup menu under Huron Net Works .

If the DeviceGate is connected directly to the PC ethernet adapter with a crossover ethernet cable, the IP address will not interfere with other nodes and can be assigned arbitrarily. In this case one could use a number such as 192.168.1.100.

If the DeviceGate will be on a hub with other nodes, the administrator of that network will need to provide you with an available IP address.

When you are ready to configure the DeviceGate, attach it to the DeviceNet Bus and apply power to the bus. The DeviceGate link LED will flash green twice. When you connect the ethernet cable, the LED will come on solid green and flash when there is traffic on the network.

Next start DeviceGate Tools. Click the Discover button to identify all DeviceGate interfaces on the network. They will be identified by the serial number of their processor. The current network configuration of each DeviceGate will be shown by opening the plus sign (+). Using the mouse, highlight the words ‘Network Configuration’. This makes the ‘Configure’ button available. The Configure button opens a dialog window for entering an IP address, a Net Mask, and a Gateway address.

After entering a new configuration, it is necessary to cycle power to the DeviceGate. Then you can proceed to access the application with your browser (see section 2).

1.4. Screen Layout

The screen consists of a panel of control buttons on the left with a frame to the right for the display of responses and DeviceNet messages. At the bottom are a row of navigation buttons for linking to other pages that can be used for customized commands.

In the result window, the received messages are shown in hexadecimal form with the CAN identifier enclosed in square brackets [ ] and the CAN data field enclosed in angle brackets < > . Where possible a text description of the message is given on the same line.

1.5. User Input

Commands are initiated through the buttons of the control panel and the navigation buttons.

Some command buttons will call a pop-up dialogue window for entry of a function's parameters. On clicking the submit button in a pop-up window, a request is sent to the DeviceGate web server by the user's browser. There may be a slight delay while the commands are processed and the results are returned. If there is no response (for instance if the network is down or the cable was disconnected), the command will time out and the result window will display "no response".

1.6. Printing and Saving Data

The normal methods of saving and printing data with a browser can be used to obtain records of DeviceNet traffic from the message buffer.

Information in the display window (the white area) can be printed by right clicking in the window frame and using your browser's print commands.

To save the data to a file or to add notations before printing, right click in the white area and "select all". Then use ctl-c to copy the data and ctl-v to paste it into an editor such as WordPad.

2. DEVICENET CONFIGURATION

The CAN interface of the DeviceGate must be configured before the emulator can communicate. It may or may not have been previously configured when a new browser session is begun.

The configuration page of the emulator (main.htm) shows the configuration status of the the DeviceGate and if necessary solicits input from the user.

For this reason, bookmarks should not be used to jump to emulator sub-pages since this could bypass configuration.

Note: there is no actual file called main.htm because it is constructed as needed by the emulator.

2.1. CAN Configuration Page

If the DeviceGate CAN interface has not been configured, the base URL will present a form requesting the desired Baud rate, Slave Mac Id, and Master Mac Id. If the DeviceGate has already been configured, the form will be skipped, and configuration status will be presented.

If the DeviceGate was set up with an IP address of 192.168.200.2, for instance, then the Base URL would be:

```
http:// 192.168.200.2/main.htm
```

Alternatively

```
http:// 192.168.200.2/
```

may be used. It is not necessary to include main.htm, since the URL alone will access the base page.

The baud rate can be 125kb, 250kb or 500kb and must match the baud rate of the DeviceNet network.

The Id of the device under test is entered as the Slave Id. The default is 63.

An Id should be chosen for emulator that does not conflict with other devices on the network. A decimal value from 0 to 63 is entered into the Master Id box. The default is 1.

2.2. Status Page

This web page will appear after the above configuration step and also in response to the base URL if the DeviceGate had already been configured.

This page will display the configuration information and present a link to the first page of the emulator.

3. EMULATOR COMMAND BUTTONS

The buttons on an emulator page are used to send page requests and CGI requests to the emulator where they are converted into DeviceNet messages. The responses to these messages are received by the emulator and formatted into XML data which is passed back to the browser. Javascript functions in *emu.js* are used to interperet and display the XML data.

If a slave device is unable to respond there will be a "No Response" message after a timeout period.

The following DeviceNet actions are available as command buttons:

3.1. Configure

This command allows re-configuration from the emulator screen. The configuration button opens a dialog window for entering master id, slave id and baudrate. The Id's are a decimal number from 0 to 63.

The Master Id and the Slave Id will be used by the other commands to construct message Id's. These may be changed at any time by repeating the configuration command.

3.2. Set Filter

The hardware screener in the CAN controller chip in the DeviceGate can be programmed to accept certain messages and ignore other traffic on the network.

The mask and match fields in the pop-up window take hexadecimal arguments as their parameters. The default values are 0xff in all fields, since this leaves the filter open to all messages.

The fields of each parameter corresponds to the ID bits of the incoming CAN identifier as follows:

Left Byte

7	6	5	4	3	2	1	0
ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3

Right Byte

7	6	5	4	3	2	1	0
ID2	ID1	ID0	RTR	*	*	*	*

In constructing a mask place a zero(0) in each position of the eleven bit CAN Identifier field that is significant for determining whether a message is to be accepted. Place a one (1) in each position which is a "don't care".

In constructing the match place a zero(0) in each position of the eleven bit CAN Identifier field that is to be a zero (0). Place a one (1) in each position which is to be a one (1).

Mask & Match Example #1

Suppose that we want to monitor only the **poll** responses from a particular slave node (Node 60). We construct the mask condition as follows:

BIT	10	=	0	Group 1 Message
BITS	9-6	=	1111	Poll Response Message ID
BITS	5-0	=	111100	MAC ID 60

mask	**00**	**1F**
match	**7F**	**80**

All Identifier bits are significant. RTR and the lower unused bits of the mask are set to “don’t care”.

Mask & Match Example #2

Suppose we want to monitor all Group 3 messages. The mask and match commands would be as follows:

mask	**3F**	**FF**
match	**C0**	**00**

Only bits 10 and 9 of the CAN Identifier field are significant, and they must both be one.

3.3. UCMM

The **ucmm** command toggles an internal variable which selects either the Group 2 predefined Explicit Request or the Explicit Request created by the UCMM. After both a UCMM connection (see **open**) and the predefined Explicit (see **allocate**) have been created, this toggle will allow a choice of which method to use.

3.4. Open

The **open** command takes three decimal parameters, and constructs a UCMM Open Explicit Request Message. This message is a group 3 message, with a message ID of six(6), and the Source MAC ID (**masterid)** The three parameters are the requested body format, the group select, and the source message id in that order. No error checking is done on the parameters so many possible combinations will result in error responses or other unexpected conditions. Valid ranges for each of the three parameters is [0..15]. The data field of the message consists of the message header, the service code (0x4B), and the three parameters packed into two bytes. See DeviceNet Specification Vol. I, Rel 2.0, pp. 4-7 to 4-10.

Example

Constructe a UCMM Open, asking for DeviceNet 8/8, on Group 3 with Message ID two(2).

open 0 3 2 Then press Enter

The resulting message might look like

[781]<3F 4B 00 32>

Example

Construct a UCMM Open asking for DeviceNet 16/16, on Group 1 with Message ID 10

open 2 1 10 Then press Enter

The resulting message might look like

[781]<7F 4B 02 1A>

The **open** command sets the value of an internal variable called **ucmm** to the value one(1). It can be toggled between one(1) and zero(0) with the ucmm command.

3.5. Close

The **close** command takes a decimal parameter which is the instance number of the connection to close. It constructs a UCMM Close Connection Request which is a group 3 message, with message ID six(6) and source MAC ID (**masterid**). The data field consists of the message header, the service code (0x4C) and the connection instance number. See the DeviceNet Specification Vol. I, Rel 2.0, pp. 4-17 to 4-19.

Example

Close connection instance #3 on the slave device

close 3 Then press Enter

The resulting message might look like

[781]<3F 4C 03 00>

The internal variable **ucmm** is set to zero.

3.6. Body Format

The **bodyformat** command takes a decimal parameter which may be a in the range zero(0) to three(3). The value is saved in an internal variable; it is then used to construct Explicit Request Messages in any of the acceptable formats. The following table shows the correspondence between the values of **bodyformat** and the format of a corresponding Explicit Request.

Value	Meaning
0	DeviceNet(8/8) Class= 8 bits, Instance= 8 bits
1	DeviceNet(8/16) Class= 8 bits, Instance = 16 bits
2	DeviceNet(16/16) Class= 16 bits, Instance = 16 bits
3	DeviceNet(16/8) Class= 16 bits, Instance = 8 bits

3.7. Allocate

The allocate command is used to establish one or more of the connections in the Predefined Master/Slave Connection Set (DeviceNet Specification Vol. I, Rev 1.3, Chapter 7). The single parameter for the allocate command is a bit mask which is, described in the DeviceNet Specification Vol. I, Rel 2.0, p5-57, and constructed as follows:

7	6	5	4	3	2	1	0
*	NACK	CYC	COS	MLT	STB	POLL	EM

* Reserved (always set to 0)

The following choices will be presented in a dialog window and this will be used construct the allocation choice parameter:

Cyclic/No Ack	CYC + NACK
Cyclic	CYC
Change of State/No Ack	COS + NACK
Change of State	COS
Multicast	MLT
Strobe	STB
Poll	POLL
Explicit	EM

To establish the connections between the Master and the Slave, set the Slave's MAC ID with the configure command to that of the desired Slave device

*** Note:**
It is not possible to allocate just the I/O Connections without the Explicit Messaging Connection. It is possible to allocate all connections and then release the Explicit Messaging Connection.

3.8. Release

The **release** command informs the Slave that it is no longer under the Master's control. The parameter to the release is a bit mask, which follows the same format as for the **allocate**:

7	6	5	4	3	2	1	0
*	*	CYC	COS	MLT	STB	POLL	EM

* Reserved (always set to 0)

The following choices will be presented in a dialog window and this will be used construct the release choice parameter:

Cyclic	CYC
Change of State	COS
Multicast	MLT
Strobe	STB
Poll	POLL
Explicit	EM

This command operates the same as the allocate command, for releasing a previously allocated connection.

3.9. Set EPR

The Set EPR command sets the Expected Packet Rate (EPR) attribute (attribute # 9) of a connection instance(DeviceNet Specification Vol. I, Rev 1.3, page 5-8). This command takes two arguments. The first is the connection instance and the second is the EPR value. For the Predefined Master/Slave connection set, the connection choices are as follows:

Connection Instance Number	Connection Type
1	Explicit Messaging
2	Poll Connection
3	Bit-Strobe Connection
4	COS/Cyclic Connection
5	Multicast Connection

The units of the EPR value are milliseconds.

3.10. Reset

The **reset** command uses **class** and **instance** to build an explicit message whose service code is a reset service (0x05). Not all objects support a reset service, so error responses are to be expected in many cases. Two DeviceNet objects which typically support the reset service are the identity object and the connection object. Resetting the identity object will cause the slave device to execute a DupMac sequence.

3.11. Exit

This command causes termination of the emulator application. Once this command is given, it is necessary to cycle the power on the DeviceGate to restart the application.

This command may be needed if it is desired to connect to the DeviceGate through its FTP server. While the application is running, the web server will have a higher priority and will block the FTP server. The FTP server can be used to upload and download customized html pages and javascript files. (See section 4 on working with the example files).

3.12. Buffer

In addition to the response that is displayed upon execution of a command, all network traffic is recorded in the buffer of the DeviceGate. This data can be displayed using the "Buffer" button.

The buffer is a circular queue in the DeviceGate and contains the last 500 messages. When it is full, the oldest messages will be overwritten. A separate button is provided to clear the buffer.

The display will show the message group number, a timestamp in milliseconds, and the message contents as follows:

The eleven bit CAN identifier, in hexadecimal notation, is enclosed in square brackets. The message body consists of one or more bytes, in hexadecimal notation, enclosed in angle brackets. For example a message with the identifier 0x5E4 and two data bytes (0x55 and 0xAA) would be shown as follows:

[5E4]<55 AA>

3.13. Clear Buffer

This button clears the DeviceGate traffic buffer as described above.

3.14. Get

The Get command is used to send a get attribute single service to the slave device over the explicit messaging connection. In the dialog for this command, the user enters Class in Hex, Instance in decimal and Attribute in decimal.

This command uses the slaveid and masterid that were set during configuration to construct the message. The request will issue a get single attribute command to the Slave device and display the results.

If the result is longer than 8 bytes, the data will be returned using fragmentation protocol. The display will wait until all of the fragments have been received. The data will be displayed as if in one continuous message (that is, the display is not limited to 8 bytes). The details of the actual transaction showing all fragments and frag acks will be recorded in the DeviceGate buffer, which can re read with the monitor command.

3.15. Get Next

The values of class, instance and attribute are javascript variables that record the last values used. The “get next” button is a short cut to get the next attribute of the current class and instance. The values of these variables will be reset to 1,1,1 when the current page is refreshed or when switching to one of the other pages.

3.16. Set

This command allows the user to send “set attribute single” commands by entering Class in hexadecimal, Instance in decimal, Attribute in decimal and Data in hexadecimal. The form provides space for 8 data bytes, however if more than two bytes are entered, the message will be sent using fragmentation. The display window will display the last packet sent and its response. The complete fragmentation & acknowledge sequence can be viewed with the monitor command.

3.17. Bit Strobe

The **strobe** command [Group2, Source Mac Id, MsgId 0] sends one bit of output data to each allocated slave, using the Master's Bit Strobe Command Message. The message length would be eight bytes and each allocated slave on the network is assigned one bit out of the sixty-four, corresponding to its MAC address, to be used as it wishes. Each allocated slave receiving a bit strobe from its master will produce a bit strobe response. Master Emulator also supports the use of the zero length strobe. In this case, just the command with no data is required.

Example #1

Send a strobe bit of 1 to device 60 and zero to the rest.

strobe 0 0 0 0 0 0 0 10 Then press Enter.

To send a zero length strobe to the slaves allocated to the Master Emulator, press Enter without entering data.

3.18. Poll

The poll command is used to send packets of data to the slave device using the Master's Poll command [Group2, Destination Mac Id, MsgId 5]. The Poll command may send up to eight bytes non-fragmented to the destination slave device.

A zero length poll can be sent to the slave device by entering no data.

3.19. MultiCast Poll

The MultiCast command is used to broadcast packets of data to slave devices using the Master's Poll command [Group2, Destination Mac Id, MsgId 1]. The MultiCast specification allows any amount of data to be broadcast. The emulator function in this application allows up to eight bytes non-fragmented.

3.20. Build Message

DeviceNet messages can be built by entering valules into a form to create the eleven bit ID and up to eight bytes of data. The information is entered in hexadecimal. A table of message group ID formats is presented as a guide in the pop-up dialog window. This shows which bits of the ID represent group number, message number and MAC ID, since these vary for each message group.

group 1:	0	Msg Id				MAC Id					
group 2:	1	0	MAC Id						Msg Id		
group 3:	1	1	Msg Id			MAC Id					
group 4:	1	1	1	1	1	Msg Id					
ID bits:	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**
ID (hex):	0-7		0-F			0-F					

4. CUSTOM COMMANDS

The third HTML page of the emulator application illustrates how the user might compose javascript functions and reference them with HTML in order to create customized command buttons.

The files *example.htm* and *example.js* may be downloaded from the DeviceGate and used as templates for additional pages of command buttons. The new pages can be uploaded back to DeviceGate and made accessible by way of the navigation buttons.

Any FTP program may be used to upload and download from the DeviceGate. FTP can be accessed from DOS by typing FTP. Tyoing 'help' will show the available commands.

Type Open followed by the IP address of the DeviceGate. Then for user name and password enter 'ftp'. Now the dir command will list the files on DeviceGate. Put 'filename.ext' is used to upload a file and Get 'filename.ext' will download a file.

There is about 230k of file storage space of which about 60k is used by application files.

It will be necessary to exit from the application before using the FTP connection, and to cycle the power to restart the application. See section 3.11.

4.1. Request/Response

When a button is clicked, a javascript function is called. Most custom commands will use the **doRequest** function in *emu.js*. This provides the means for passing parameters to the emulator and receiving an XML document as the response. Data is extracted by this same function call into a global variable named **response** which can be displayed with the showIt() function.

A simple command function will have the form:

```
function name()
{
     doRequest(url);
     showIt();
}
```

where *name* is your function name and *url* is a string representing the url of the desired emulator webpage (see below).

4.2. Emulator URLs

The DeviceNet data availble via custom commands will be accessed by linking to two different URLs in the emulator. One of these responds with data from the emulator buffer. The other URL accepts cgi parameters as input from the user and responds with the resulting message.

If the DeviceGate had an IP address of 192.168.200.2, for instance, then the Buffer url would be:

```
http:// 192.168.200.2/buf
```

and this is the cgi request url**:**

```
http:// 192.168.200.2/req
```

Emulator requests containing parameters use CGI syntax in which the ampersand (&) seperates the parameters:

```
http:// 192.168.200.2/req?f=9&c=1&i=1&a=7
```

If one of these urls is typed into the navigation bar of your browser, the emulator will respond with XML data. The browser does not know how to interperet the XML without javascript. The **doRequest** javascript function knows how to parse the XML and decode the data.

The **doRequest** function does not require the full url since it is local to the DeviceGate.

Buffer data is obtained as follows:

```
doRequest(“buf”);
```

An emulator request function would be called this way:

```
doRequest(“req?f=9&c=1&i=1&a=7”);
```

4.3. Emulator Request Functions

In this section and the next, the word "function" has two different contexts. This section refers to the various actions that can be accessed through the CGI request web page. The first parameter in the CGI url is the function number.

The section on javascript functions refers to the code that appears in script.js, user.js and any other script file that might be used.

As noted in the preceding section, "req" is the URL of the page that handles CGI requets. The question mark (?) is the start of a CGI parameter list. Each element of the list is seperated by the ampersand (&). The character before the equal sign (=) is an arbitrary parameter name. (The emulator web server doesn't care what characters are used for parameter names. For each function, the order of the parameters determines their meaning.) The number following the equal sign (=) is the parameter value.

The first parameter in the list is the emulator function number. For example, f=9 is a **Get Attribute Single** request.

Each emulator function requires a specific parameter list. For instance, given f=9, class = 1, instance = 1, and attribute = 6 the emulator will request a serial number, the 6th attribute of the identity object, from the slave device.

Here are the emulator functions and their required parameters:

DeviceGate Configuration *		
Function number	1	See section 3.1
Parameter 1	0-63	Master MAC Id (decimal)
Parameter 2	0-63	Slave MAC Id (decimal)
Parameter 3	125, 250, 500	Baud rate (kb)

* This function sets the Mac Id variables in the emulator that are used to build message id's. It also programs the CAN interface with the baud rate.

Set Expected Packet Rate		
Function number	2	See section 3.9
Parameter 1	1-5	Connection instance
Parameter 2 *	0-32767	Milliseconds (decimal)

* If Parameter 2 is omitted, the epr will be set to 0. This will prevent a connection from timing out.

Change MAC Id's *		
Function number	3	See section 3.1
Parameter 1	0-63	Master MAC Id (decimal)
Parameter 2	0-63	Slave MAC Id (decimal)

* This is similar to function 1, except that the CAN interface is not effected.

Open		
Function number	4	See section 3.4
Parameter 1	0-3	Body format
Parameter 2	1-4	Group number
Parameter 3	0-15	Source message ID

Close		
Function number	5	See section 3.5
Parameter 1	1-5	Connection instance

Allocate Connection		
Function number	6	See section 3.7
Parameter 1	1-5	Connection (See section 3.9)
Parameter 2	Selection bits	Choices

Release Connection		
Function number	7	See section 3.8
Parameter 1	1-5	Connection (See section 3.9)
Parameter 2	Selection bits	Choices

Reset		
Function number	8	See section 3.10
Parameter 1	Class	hexadecimal
Parameter 2	Instance	hexadecimal

Get Attribute Single *		
Function number	9	See section 3.14
Parameter 1	Class	hexadecimal
Parameter 2	Instance	hexadecimal

Parameter 3	Attribute	hexadecimal

* If a response to an explicit is sent as a fragmented message the its contents will be displayed as a single string after all fragements have been received and acknowledged. The individual fragemented packets and their Acks can be viewed in the buffer.

Set Attribute Single		
Function number	10	See section 3.16
Parameter 1	Class	hexadecimal
Parameter 2	Instance	hexadecimal
Parameter 3	Attribute	hexadecimal
Parameter 4, 5, etc. *	Value	hexadecimal

* Function 8 will accept up to 80 bytes of data and produce a fragmented explicit message. The displayed response will only show the last packet sent and its response. The individual fragemented packets and their Acks can be viewed in the buffer.

Poll (or Master's Change of State)		
Function number	11	See section 3.18
Parameter 1	Output Data	Up to 8 bytes hexadecimal

Build Message		
Function number	12	See section 3.20
Parameter 1	0-7 (hexadecimal)	ID 1
Parameter 2	0-f (hexadecimal)	ID 2
Parameter 3	0-f (hexadecimal)	ID 3
Parameter 4 - 11	Up to 8 bytes (hexadecimal)	Message Body

Set Mask and Match Filter		
Function number	13	See section 3.2
Parameter 1	Mask1	hexadecimal
Parameter 2	Mask2	hexadecimal
Parameter 3	Match1	hexadecimal
Parameter 4	Match2	hexadecimal

Quit		
Function number	14	See Section 3.11

UCMM		
Function number	15	See Section 3.3

Clear Buffer		
Function number	16	See Section 3.13

Multicast Poll		
Function number	17	See Section 3.19
Parameter 1 thru 8	Data	

Bit Strobe		
Function number	18	See Section 3.17
Parameter 1 thru 8	Data	

Body Format		
Function number	19	See Section 3.6
Parameter 1	0-3	format

The next two functions are available but have no corresponding buttons in the web page display. See section 5.4 for more detail on using functions without the emulator javascript.

Serial Send		
Function number	20	Uses Ext port at 9600 baud (8-odd-1)
Parameter 1 thru 10	ASCII (hex)	Send up to 10 bytes

Example:

```
     http://
192.168.200.2/req?f=20&d=48&d=65&d=6c&d=6c&d=6f
```

This sends the string “Hello” on the serial (EXT) port of the DN-E100. An XML response will be returned to the browser:

```
<?xml version="1.0" ?>
- <HNW>
  - <MSG>
      <QID>000</QID>
      <QLN>05</QLN>
      <QMB>48 65 6c 6c 6f</QMB>
      <RID>000</RID>
      <RLN>00</RLN>
      <RMB />
      <TXT>Serial Message Sent</TXT>
    </MSG>
  </HNW>
```

Serial Recieve		
Function number	21	Uses Ext port at 9600 baud (8-odd-1)
Parameter 1	Length (1 byte hex)	0 to ff (# of bytes to recieve)

Example:

```
http:// 192.168.200.2/req?f=21&L=ff
```

This reads up to 255 characters from the serial port receive buffer. The results are sent back as XML. If the characters “1234567890” had previously been typed on a terminal emulator connected to the serial port then the result would be:

```
<?xml version="1.0" ?>
- <HNW>
  - <MSG>
      <QID>000</QID>
      <QLN>00</QLN>
      <QMB />
      <RID>000</RID>
      <RLN>10</RLN>
      <RMB>31 32 33 34 35 36 37 38 39 30</RMB>
      <TXT>Serial Message Rcvd</TXT>
    </MSG>
  </HNW>
```

4.4. Javascript Functions

Request urls derived from the above tables can be assigned to variables which are then used as follows:

To get the serial number of the slave device:

```
var SerNum     =    "req?f=9&c=1&i=1&a=6";

function getSerNum()
{
     doRequest(SerNum);
     showIt();
}
```

4.5. Multiple Responses

Multiple calls to **doRequest** can be made in a single javascript function, and each will return a response string that is displayed in the results window. If the **showIt** function is used to display the results, only the last one will be visible, since **showIt** closes the window after displaying each response.

To show the results of a series of requests without overwriting previous responses, the window can be left open until all results have been displayed.

For instance, to allocate an explicit connection and an IO connection and then set the IO connection expected packet rate to 500 mSec:

```
var allocate3  =    "req?f=6&c=3";

var setEpr2    =    "req?f=2&c=2&t=500";

function allocateDevice()
{
     doRequest(allocate3);
     window.frames[0].document.open();
     window.frames[0].document.write(response);

     doRequest(setEpr2);
     window.frames[0].document.write(response);
```

```
     window.frames[0].document.close();
}
```

Note: The global variable **response** is a string that contains the results of parsing the XML response data.

4.6. Command buttons and Navigation buttons

Javascript functions can be linked to buttons on the command panel with HTML code as follows:

```
<INPUT        TYPE="button"                VALUE="Serial#"
onclick="getSerNum()">
```

The navigation buttons at the bottom of each page can be used to add pages for new command panels:

```
<INPUT CLASS=NAV TYPE="button" VALUE="Next > " onclick =
"page2()" >
```

where **page2()** is a javascript function as follows:

```
function page2()
{
     window.location = "emu2.htm";
}
```

5. APPLICATION HINTS

5.1. Allocating Connections

To debug various kinds of messages and objects on a slave device it is necessary to establish at least an Explicit Messaging Connection and possibly IO connections. Each connection is associated with an expected packet rate (EPR) which causes it to time out after four (4) times the EPR has elapsed with no message. It is useful to set the explicit connection EPR to zero when doing manual tests, so that it will not time out.

This can be accomplished with the **Allocate** and **Set EPR** command buttons on the first page. This could also be done with a custom command button that does multiple emulator requests to set up one or more connections at the same time.

Here is an example:

```
var allocate3 =    "req?f=6&c=3";    //selects    IO    and
explicit
var setEpr1   =    "req?f=2&c=1"; // Explicit epr = 0
var setEpr2   =    "req?f=2&c=2"; // IO epr = 0

function StartCnxn()
{
     doRequest(allocate3);
     window.frames[0].document.open();
     window.frames[0].document.write(response);

     doRequest(setEpr1);
     window.frames[0].document.write(response);

     doRequest(setEpr2);
     window.frames[0].document.write(response);
     window.frames[0].document.close();
}
```

5.2. Simple Scanner

A simple scanner (MACID = 1) with two slave devices (MACIDs 52 and 54) can be implemented with one command button definition and two javascript functions.

The first function establishes the slave connections and then sets a time interval in milliseconds at which the second function will be executed. The first function is linked to the command button.

The scan function will run automatically at one second intervals until the active html page is refreshed from the browser tool bar or changed with navigation buttons.

```
var slave52 = "req?f=3&m=1&s=52";
var slave54 = "req?f=3&m=1&s=54";
var PollOff =  "req?f=11&d=0";
var PollOn  =  "req?f=11&d=1";


function startScan()
{
     doRequest(slave52);
     doRequest(allocate3);
     doRequest(setEpr1);
     doRequest(setEpr2);

     doRequest(slave54);
     doRequest(allocate3);
     doRequest(setEpr1);
     doRequest(setEpr2);

     setInterval("scan()", 1000);
}


function scan()
{
     doRequest(slave52)
     doRequest(PollOff);
     doRequest(slave54);
     doRequest(PollOn);
}
```

5.3. Help Text

It may be useful to use command buttons to provide instructions or prompts to the user:

```
function help()
{
     var helptext = "<P>This button demonstrates a”;
     helptext += “function for writing help text into”;
     helptext += “ the display window.";
     helptext += "<P>This is another paragraph”;
```

```
      window.frames[0].document.open();
      window.frames[0].document.write(helptext);
      window.frames[0].document.close();
}
```

The example shows that strings written to the display window can contain HTML tags as seen above with the <P> tag seperating paragraphs.

The help function can be linked to a button with this HTML code:

```
<INPUT TYPE="button"  VALUE=" HELP " onclick="help()">
```

5.4. Accessing functions without the emulator javascript

You may wish to access emulator functions with your own programs. Functions can be called directly through a url. As an example, here is the url for the "Build Message" function to send a poll with 2 bytes of data:

http://192.168.1.32/req?f=12&i1=5&i2=f&i3=d&d1=0&d2=1

This calls function number 12 and provides a CAN id of 5fd with two bytes of data, 00 and 01.

The response will be returned in XML:

```
<?xml version="1.0" ?>
- <HNW>
  - <MSG>
      <QID>5fd</QID>
      <QLN>02</QLN>
      <QMB>00 01 </QMB>
      <RID>3ff</RID>
      <RLN>01</RLN>
      <RMB>00</RMB>
      <TXT>IO Message</TXT>
    </MSG>
  </HNW>
```

QID is the query id
QLN is the query length
QMB is the query data

RID is the response id
RLN is the response length
RMB is the response message

TXT identifies the response type

6. APPENDIX: IP ADDRESS SETUP

This appendix describes how to determine an IP address to use for your DeviceGate. First there is an explanation of IP addresses and subnet masks. Then, several possible configurations are considered.

1. Connection through a hub to a network with a DHCP server.
2. Connection through a hub to a network with no DHCP server.
3. Connection to a PC ethernet adapter via cross-over cable with only one ethernet adapter in the PC.
4. Connection to a PC ethernet adapter via cross-over cable when the PC is also connected to a network through a seperate ethernet adapter.

6.1. IP addresses and subnet masks

An IP address consists of four 8-bit fields (octets). An IP address can specify a whole network and it can also specify a node on a specific network.

The DeviceGate will have an IP node address and any PC that communicates with it will have an ethernet adapter with a different IP node address. The two addresses must be on the same network or sub-network. The following will help to determine your network address and the range of possible node addresses for that network.

6.1.1. Reserved Addresses

There are a couple of octet values that are reserved and must not be used as part of a node address.

- An octet must not be set to zero in a node address, since zero is reserved for network addresses. For instance, 192.168.10.23 could be a node on network 192.168.10.0.

- The top address of a network or subnetwork is the value in which all bits are set to one. This is reserved for broadcast messages and should not be used as a node address. If someone sent a message to 192.168.10.255 on the network 192.168.10.0, it would be seen by all nodes on the network.

6.1.2. Masks

Subnet Masks are used to identify the network and node sections of an IP address. To find the network address of an adapter do a bitwise AND with the node address and the subnet mask. For instance:

IP=192.168.10.23	11000000.10101000. 00001010.00010111
Mask=255.255.255.0	11111111.11111111.11111111.00000000
Net=192.168.10.0	11000000.10101000.00001010.00000000

This shows that the right hand octet is the node address portion. Excluding 0 and 255, the node addresses can range from 1 through 254.

In this example it wasn't really necessary to resort to binary calculations because the mask fell neatly on octet boundaries making it possible to simply read the net address from the top three octets. Sometimes it is less obvious.

6.1.3. Subnets

Subnet masks allow subdividing networks into various sizes, not restricted to simple octet boundaries as described above. Take this example:

172.16.0.0 is a network with 65,534 possible nodes (2^16 addressese minus the two reserved values 172.16.0.0 and 172.16.255.255).

This network can be divided into 16 subnets with 254 possible nodes in each by using a subnet mask of 255.255.240.0. 240 is 11110000 binary. This allows 4 bits for specifying 16 different subnets.

Here is an example of identifying the subnet of a node on this network:

IP=172.16.127.1	10101100.00010000.01111111.00000001
Mask=255.255.240.0	11111111.11111111.11110000.00000000
Subnet=172.16.112.0	10101100.00010000.01110000.00000000

Subnet 172.16.112.0 has nodes from 176.16.112.1 through 172.16.127.254. It is one of 16 subnets on the network 176.16.0.0.

6.2. Connection through a hub to a network with a DHCP server

This is perhaps the easiest to set up because you don't have to choose the IP address yourself, but it is not very practical because the DHCP server can re-assign the address at any time. This means that your DeviceGate URL could change without notice and you would have to use DGtools to re-discover it. However, we include the description of this case for completeness.

If your network uses a DHCP server, Network Neighborhood Properties will show that the TCP/IP configuration for your etherlink card is set to obtain an ip address automatically.

Use a straight ethernet cable rather than a cross-over cable between a hub and the DeviceGate. Use the Device Configuration dialog box in DGtools to set DHCP "on" and ignore the other configuration settings. When you power cycle the DeviceGate, the DHCP server should automatically assign an IP address.

Use DGtools to do a "discover". This will show you the address that was assigned to DeviceGate. Now you can use that address as the URL in your browser.

6.3. Connection through a hub to a network with no DHCP server

In this case, your system administrator is coordinating the allocation of available IP addresses. You will have to use a unique address that is not being used by any other node on the network. You will also need to know the correct net mask to use. In the configuration dialog window of DGtools, DHCP is set to "off". Fill in the IP address and the net mask. The gateway field can be ignored since it is not being used.

Cycle the power to DeviceGate and use DGtools to do a "discover". This will show you the new address of the DeviceGate. Use that address as the URL in your browser.

6.4. Connection to a PC ethernet adapter via cross-over cable with only one ethernet adapter in the PC

In this case you need two IP addresses on the same network, one for the ethernet adapter card in the PC and one for the DeviceGate. To find the setup of your adapter card, go to the "Start" menu and use the Run command. Enter "winipcfg". Open the drop down window in "IP Configuration" and select the ethernet adapter. Write down the IP address and the subnet

mask. Refer to the above explanation for how to determine the address of the network. Choose a different IP address in the same network space. Be sure not to choose the reserved values (see above).

Use DGtools to configure the IP address that you have chosen and the same subnet mask as your adapter card. Ignore the gateway address. Leave DHCP set to "off".

Cycle the power to DeviceGate and use DGtools to do a "discover". This will show you the address that was assigned to DeviceGate. Now you can use that address as the URL in your browser.

6.5. Connection to a PC ethernet adapter via cross-over cable when the PC is also connected to a network through a separate ethernet adapter

First you will need to determine the network address of you existing network. Go to the "Start" menu and use the Run command. Enter "winipcfg". Open the drop down window in "IP Configuration" and select the ethernet adapter. Write down the IP address and the subnet mask. Refer to the above explanation for how to determine the network address that you are on.

The Network address of the point-to-point connection (the DeviceGate), will have to be different than the existing network address. Then, just pick two node addresses for the point-to-point network. Remember to avoid the first and last addresses on that network (the ones reserved for the whole network and for broadcast messages). One of these addresses is assigned to the adapter card.