Banner Engineering Corp

DXM Controller API

7/1/2019
186221 Rev H

Contents

Overview

The table below gives a functional overview of the communications options with the DXM controller. The two physical media are Ethernet or Cellular. The two options for communications methods are Device-Initiated (DXM Controller to Host) or Host-initiated (Host to DXM Controller) communications.

	Device-Initiated Communications (Mobile Originated Data)	**Host-Initiated Commutations (Mobile Terminated Communications)**
	The DXM Controller creates the data to send to the host, the host can return information within the acknowledgement packets to control the device. Data is typically sent on a cyclical schedule.	A host system sends messages directly to the DXM controller. This requires a private network connection (VPN) when using Cellular or when using Ethernet the host path will require the firewalls to be managed.
Ethernet	The DXM Controller will require the ability to send GET and POST messages to a host system. Some network firewall settings will block outgoing POST messages. Most HTTP GET packets will be allowed, these are typical of browser based messages. Control messages to the DXM device will be sent in the acknowledgement packet from GET messages. The host system can only send control messages to the DXM Controller at the time the device sends a message to the host. This is typically on a cyclical time schedule.	A host system will need to have direct access to the IP address of the DXM controller. From a remote host the network firewall settings will need to allow access to the DXM controller. A host on the same internal network as the DXM controller should be able to access the device using the API functions. Immediate control of the DXM controller is possible.
Cellular	The DXM Controller will send GET and POST packets using a standard dynamic cellular connection. Control messages to the DXM device will be sent in the acknowledgement packet from HTTP GET messages. The host system can only send control messages to the DXM Controller at the time the device sends a message to the host. This is typically on a cyclical time schedule. The host system can send an email/text message to the DXM device to force the device to 'call home'. This allows for immediate interaction with the DXM Controller.	A private network (VPN) must be established to communicate directly with the DXM controller using API commands. This allows the host and the DXM controller to communicate with each other using the respective IP addresses. Immediate control of the DXM controller is possible.

Device-Initiated Protocol

This section describes the device-initiated protocol used by the DXM Controller. The HTTP GET request is used as the basic register push method. The acknowledgement messages to the GET requests may also contain messages to the controller to indicate other actions (output updates, for example). At a minimum, the acknowledgement to a GET request must contain the site ID string. HTTP POST requests are used when log data is sent, typically larger data sets.

Outputs are updated on a cyclical schedule; that is, the next time interval the device sends a GET request the acknowledgement message contains the updates. For updating outputs immediately, an email is sent to the phone number (if cellular connected). The device immediately corresponds with the webserver. If Ethernet connected, a response can be created by sending a GET request to 'triggerpush.htm', port 80 on the devices' internal webserver. Only an acknowledgement is sent in return, there is no HTML content.

The push string of the GET request is made up from input parameters defined in the table below.

Input Parameters

id=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx	A 32-character string used as a site identifier. This unique string is generated by the web site and stored in the XML configuration file to be used by the client device initiating the data push.
regX=Y	Defines the register number (X) and value for the register, (Y). Data reported in a cyclical basis is reported in this format. Register values are 32-bit values; signed or unsigned based on the configuration XML. Registers greater than 1000 are in IEEE 754 floating point format.
eventX=Y	Defines a threshold rule that has changed state. The Threshold rule is indicated by X and the time/date and value of the Threshold rules is Y. The "Y" format is <Date><Time>, A, B. Date format is MM/DD/YYYY - Time format is HH:MM:SS –' A' is the Threshold rule number - 'B' is "T" or "F" indicating whether the rule was evaluated as true or false. The space between the date and time will be URL encoded as "%20" Additionally, if the client device is so configured (verbose mode) additional information will be included in the format: <date><time>,A,B,<name>,C,D,E,F Where <name> is the name of the threshold rule, C is the register number tested, D is the data in that register, E describes the test type (GT, LT, etc.) and F is the test threshold value the register value is compared against.
flags=XXXXX	A bit map indicating the present status of threshold events. If there is no threshold rules defined, this tag is omitted. Otherwise, the string of digits is equal in length to the number of rules, with 0 indicating the rule is inactive and 1 indicating the rule is active. This status is reported as part of every periodic log along with the register data (if any registers have been selected for reporting)
pkt=yyyymmddhhmmss	Defines the packet time stamp. Year, month,day,hour,min,sec (UTC time)

GET Request

The GET request is the main method for pushing data at a periodic rate from the client device to the web server. In the acknowledgment of the GET request, further operations can be defined for the client device. The acknowledgment data from the GET request must always include the site ID string. See Messages to the Remote device for more information. For example, the acknowledgment might include a new XML configuration file or tell the device to reset. There may be multiple GET requests if the amount of register information exceeds 100 registers.

Event-driven pushes are created by Threshold Rules defined in the device. They create immediate data for the webserver to consume. Below is an example of a Cyclical Push and an Event-driven push.

Example: Cyclical Push

From device to webserver:

```
GET /push.aspx?id=505678ed-4ee4-4034-b62c-
8bad456fae55&pkt=20131205153418&flags=000110000100100&reg1=31032&reg3=48&reg4=0&reg5=4&re
g6=0&reg7=4&reg8=0&reg10=0&reg15=0&reg23=0&reg24=0&reg32=0&reg36=0&reg37=0&reg52=0&reg53
=0&reg61=0&reg80=65535&reg81=65535&reg82=65535&reg83=65535&reg84=65535&reg85=65535&reg9
```

0=6868&reg91=59648&reg92=51272&reg93=960&reg94=524&reg95=39136&reg96=2&reg97=22328&reg98=0&reg99=51213

The basic acknowledgement from the web server to the DXM controller is in HTML format.

<html><head><title>HTTP Push Ack</title></head><body>id=505678ed-4ee4-4034-b62c-8bad456fae55</body></html>

Example: Event-driven Push

GET /push.aspx?id=e89b06c4-b921-46aa-b7e1-7a329be3e16c&pkt=20140404152316&lat=45.019531,0&lon=-93.480309,0&flags=1&event0=20140404152316,T,ErrorCondition,1,21,gt,0.000000

The GET command is followed by the website page to push to: "/push.aspx". This is the default for the Sensonix website. After the "?" delimiter the site id is defined.

The ampersand (&) is used as the delimiter for input parameters.

Also, part of the GET input parameters include the initial parameters. The first GET message after a device reset/reboot includes input parameters that show information about the client device.

lat	GPS latitude reading, no GPS entry will be shown as 'NOGPS'
lon	GPS longitude reading
mod	Device model number
ser	Device serial number
fwv	Firmware version
ipaddr	Ethernet IP address assigned to the device or Cellular IP address assigned from the network
maddr	Ethernet MAC address or cellular MAC address
xmlgid	XML GUI ID – uniquely identifies the XML file on the device. This can be compared to the XML file loaded on the web server for consistency
csq	Cellular signal quality, 1-30 logarithmic scale followed by the BER (bit error rate). A 99 indicates no data.
cnum	Cellular phone number: xxxyyyzzzz
meid	The cellular Mobile Equipment IDentifier is a globally unique number identifying a physical piece of CDMA mobile equipment.

Example: Initial Push

GET /push.aspx?id=505678ed-4ee4-4034-b62c-8bad456fae55&pkt=20131206142431**&lat=NOGPS&lon=NOGPS&mod=000000&ser=000000&fwv=v0.0.98&ipaddr=10.10.82.165&maddr=M:00:23:d9:ff:00:10**&flags=000110000100100&reg1=31028&reg3=48&reg4=0&reg5=4&reg6=0&reg7=4&reg8=0&reg10=0&reg15=0&reg23=0&reg24=0&reg32=0&reg36=0&reg37=0&reg52=0&reg53=0&reg61=0&reg80=65535&reg81=65535&reg82=65535&reg83=65535&reg84=65535&reg85=65535&reg90=6868&reg91=58176&reg92=51271&reg93=56896&reg94=528&reg95=6992&reg96=2&reg97=11359&reg98=0&reg99=61213

Example: Initial Push (cellular)

id=b06c4395-ff3c-4019-a7f8-f3dd90bcec1e&pkt=20150227204324**&lat=0.000000,0&lon=0.000000,0&csq=15,99&cnum=7342047391&meid=A1000032BD4942&mod=000000&ser=000000&fwv=v0.02.85&ipaddr=192.168.0.1&maddr=M:00:2**

```
3:d9:ff:00:6d&reg1=959&reg2=500&reg3=1540&reg4=903&reg5=502&reg6=1542&reg7=992&reg8=494&re
g9=1530&reg10=970&reg11=498&reg12=1538
```

POST Request

Use POST requests for file transfers or when there is stored data that needs to be sent to the web server. Stored data is created when the logging rate is faster than the push rate to the web server, or if the connection to the web server is not available.

The POST requests are formatted in a XML style using start and end tags. The acknowledgment message for the POST request is just like the GET request: the webserver must return the id string in HTML format.

```
<html><head><title>HTTP Push Ack</title></head><body>id=505678ed-4ee4-4034-b62c-
8bad456fae55</body></html>
```

<httplog>	Outer most tag defining entire data structure
<id>	Similar to the GET input parameter ‘id’ – defines the 32-character site id created by the website and stored in the XML configuration file on the device.
<st>	Send time stamp of the file, YYYYMMDDHHMMSS format.
<log>	XML tag for register data, including pkt (packet time stamp), flags (threshold rule status) and reg (register number & data)
<miss>	XML tag for register data that has missed the first attempt at transmitting to the web server.

Device POST request:

```
<httplog>
        <id>505678ed-4ee4-4034-b62c-8bad456fae55</id>
        <st>20131205153420</st>
<log>pkt=20131205152519&flags=000110000100100&reg1=31028&reg3=48&reg4=0&reg5=4&reg6=0&reg
7=4&reg8=0&reg10=0&reg15=0&reg23=0&reg24=0&reg32=0&reg36=0&reg37=0&reg52=0&reg53=0&reg61
=0&reg80=65535&reg81=65535&reg83=65535&reg8</log>
<log>pkt=20131205152618&flags=000110000100100&reg1=31030&reg3=48&reg4=0&reg5=4&reg6=0&reg
7=4&reg8=0&reg10=0&reg15=0&reg23=0&reg24=0&reg32=0&reg36=0&reg37=0&reg52=0&reg53=0&reg61
=0&reg80=65535&reg81=65535&reg83=65535&reg8</log>
</httplog>
```

Acknowledgment from web server in HTML format:

```
 <html><head><title>HTTP Push Ack</title></head><body>id=505678ed-4ee4-4034-b62c-
8bad456fae55</body></html>
```

POST example:

This example shows two time-stamped register groups encapsulated in the <log> tag. Each group is a result of a user defined polling time.

The 'id' string calls out the site ID string, 'st' calls out the send time stamp (not the log time of the registers). The acknowledgment from the web server is the just the 'id' string.

Device POST request:

```
<httplog>
        <id>505678ed-4ee4-4034-b62c-8bad456fae55</id>
        <st>20131205153420</st>
<log>pkt=20131205152519&flags=000110000100100&reg1=31028&reg3=48&reg4=0&reg5=4&reg6=0&reg
7=4&reg8=0&reg10=0&reg15=0&reg23=0&reg24=0&reg32=0&reg36=0&reg37=0&reg52=0&reg53=0&reg61
=0&reg80=65535&reg81=65535&reg82=65535&reg83=65535&reg8</log>
<log>pkt=20131205152618&flags=000110000100100&reg1=31030&reg3=48&reg4=0&reg5=4&reg6=0&reg
7=4&reg8=0&reg10=0&reg15=0&reg23=0&reg24=0&reg32=0&reg36=0&reg37=0&reg52=0&reg53=0&reg61
=0&reg80=65535&reg81=65535&reg82=65535&reg83=65535&reg8</log>
</httplog>
```

Authentication

A username and password will be sent when logging into the webserver if the user defines the DXM Controller to use authentication. See the DXM Configuration Tool documentation to set up the authentication.

The username and password authentication use the facilities of the HTTP protocol. The credentials need to be defined in the DXM Controller and at the web server. The username and password are not stored in the XML configuration file but in the DXM Controller's flash memory. The XML configuration does need to specify the requirement to send authentication.

Example:

```
Authorization: Basic TXJ1ZTpUZW1wcGFzczE=\r\n
```

Messages to the Remote Device

tod	At the end of the acknowledgement message from a GET request, a "time of day" can be included from the webserver to update the device time. The date is in month-day-year format. Example: tod=12-12-2013-19:20:30
configure	In the acknowledgement from a GET request the 'configure' message indicates to the device that a XML configuration file is waiting to be downloaded. The full file path must not exceed 50 characters. For example, this path and filename is 25 characters long: configure=/RSP/274870B/WLConfig.xml
regX=Y	Defines the register number (X) and value for the register, (Y). Data sent to the unit will be immediately updated. Example: reg5=2
dif	Down Load File. In the acknowledgement from a GET request, the 'dlf' message indicates to the device that a file is waiting to be downloaded. The full path must not exceed 50 characters. For example - dlf=/RSP/D009EE9/MyFile.sb
clear	Erases the saved HTTP log files. Typically, not required for normal operation. Example: clear=1
reboot	This message will reboot the DXM controller. Example: reboot=x
pwd1	Changes the root password for API access to the file system. Example: pwd1="MyPassword1"
pwd2	Changes the LCD display password for access to the LCD menu items.

	Example: pwd2="1234"
dxmfw	Upload firmware file for boot loading the processor board. Uploading firmware files require the SD card for storage. Example: dxmfw=/RSP/D009EE9/MyFile.bin
iofw	Upload firmware file for boot loading the I/O board. Uploading firmware files require the SD card for storage. Example: Iofw=/RSP/D009EE9/MyFile.bin
ismfw	Upload firmware file for boot loading the ISM radio. Uploading firmware files require the SD card for storage. Example: ismfw=/RSP/D009EE9/MyFile.bin
lcdfw	Upload firmware file for boot loading the LCD display board. Uploading firmware files require the SD card for storage. Example: lcdfw=/RSP/D009EE9/MyFile.bin

Messages can be sent to the remote device using the HTTP acknowledgement message for the normal GET request. After the remote device decodes the acknowledgement message, the device sends another GET request to start requested operation, if needed.

A POST message acknowledgment should only contain the site id string. Server control messages should be put in the GET message acknowledgment.

HTTP acknowledgement messages should be a single line; no newline characters in HTML format. See the examples below for the proper format.

A standard HTTP acknowledgment message shows the site id with no further action required. (No new line characters)

<html><head><title>HTTP Push Ack</title></head><body>id=1bd7ed04-aae8-42f4-a618-56a2b8d03ad2</body></html>

This example shows an acknowledgement message to update the time on the device. (UTC time) (HTML formatting removed for clarity)

id=1bd7ed04-aae8-42f4-a618-56a2b8d03ad2**&tod=12-12-2013-19:20:30**

Acknowledgement message content from the web server indicating to get a new XML configuration file. The full file path must not exceed 50 characters. For example this path is 45 characters long: (HTML formatting removed for clarity)

id=1bd7ed04-aae8-42f4-a618-56a2b8d03ad2**&configure=/Xml/5d66f92c-8658-4a9b-8887-fb7ab669aa0e.xml**

The follow-up GET request from the device that starts the XML configuration file transfer.

GET /Xml/5d66f92c-8658-4a9b-8887-fb7ab669aa0e.xml

The HTTP GET request in the following example pushes the cyclical data up to the webserver. The acknowledgement to the GET request contains updates to two registers: 6 and 9. Both the GET message and the acknowledgement start with the site ID string. (HTML formatting in the acknowledgement message removed for clarity)

GET /push.aspx?id=b9f368bf-f205-4be0-b602-3ecd09bf426c&pkt=20140418202059&reg1=1885&reg2=493&reg3=2181&reg4=485&reg5=1&reg6=0&reg8=0&reg9=0&reg11=0&reg12=0&reg13=0&reg14=0&reg15=0&reg16=0&reg17=3&reg18=5492&reg19=3&reg20=64404

```
id=b9f368bf-f205-4be0-b602-3ecd09bf426c&reg6=1&reg9=1
```

To update output registers immediately, the cellular method is an email/text message to the phone number of the device. When an email is sent to the device (blank contents), the device sends cyclical data to the webserver, and the webserver includes the register updates in the HTTP acknowledgement.

Example:

Webserver sends an email to the appropriate phone number of the device:

```
To: 6127239817@vtext.com
Subject: Push
Body: Push
```

Register update from the web:

```
id=604f0807-a067-437a-a26b-064a319240da&reg14=1
```

If Ethernet connected, create a response by sending a GET request to ‘triggerpush.htm’, port 80 on the devices’ internal webserver. Only an acknowledgement is sent in return, there is no HTML content.

DXM Controller File Transfer from Webserver

DXM Controllers defined to have device-initiated communications require interactions with the DXM Controller to be initiated in the acknowledgment of a GET message. (The DXM Controller is the client, and the website is the server). File transfers from a webserver to a DXM Controller start with the same mechanism; the webserver indicates to the DXM Controller that there is a file waiting through the acknowledgment from the GET message.

Configuration File Transfer from Webserver

Updating a configuration file to the DXM Controller is queued by the webserver waiting for the next DXM Controller push. After a push message is received by the webserver, the file transfer request is sent using the acknowledgment message. The process steps are shown below. The configuration file is stored in the DXM Controller’s local data flash. If multiple file operations are required, only one should be done at a time.

Standard cyclical push message from the DXM Controller to the webserver:

```
GET /push.aspx?id=0c2a907a-2014-42b6-820d-a7115af8d4f5&pkt=20170302144340&reg0=-1
```

Webserver acknowledgement to the GET request includes a request to get a new configuration file. The HTML format acknowledge includes key word configure: *configure= ‘path’/WLConfig.xml*. Configuration file loads using HTTP do not require the file name to be WLConfig.xml. Configuration file loads using API require the file name to be WLConfig.xml.

```
<html><head><title>HTTP Push Ack</title></head><body>id=0c2a907a-2014-42b6-820d-
a7115af8d4f5&configure=/RSP/D009EE9/WLConfig.xml</body></html>
```

After the DXM Controller decodes the request to get a new configuration file, a follow up GET request is sent by the DXM Controller to start the file transfer.

```
GET /RSP/D009EE9/WLConfig.xml
```

The server begins the transfer by putting the XML configuration file into TCP data payloads. (Wireshark messages are shown)

```
TCP segment data (1380 bytes)
TCP segment data (1380 bytes)
TCP segment data (1380 bytes)
TCP segment data (1380 bytes)
TCP segment data (1380 bytes)
TCP segment data (1380 bytes)
TCP segment data (1380 bytes)
TCP segment data (858 bytes)
[8 Reassembled TCP Segments (10518 bytes): #2338(1380), #2340(1380), #2342(1380), #2344(1380),
#2346(1380), #2348(1380), #2350(1380), #2352(858)]
```

After the configuration file is loaded and verified, it will reboot the DXM Controller to use the new configuration file.

ScriptBasic File Transfer from Webserver

A ScriptBasic file transfer is similar to a configuration file transfer except that the filename is the unique filename for the ScriptBasic program file. Any file transfers that are not configuration files are stored in the root directory on the internal SD card.

The file transfer mechanism starts with the next push message received by the webserver from the DXM Controller.

```
GET /push.aspx?id=0c2a907a-2014-42b6-820d-a7115af8d4f5&pkt=20170302152500&reg0=-1
```

In the acknowledgment message to the GET from the DXM Controller, the webserver puts the request for the DXM Controller to get the ScriptBasic file. (HTML format)

```
<html><head><title>HTTP Push Ack</title></head><body>id=0c2a907a-2014-42b6-820d-
a7115af8d4f5&dlf=/RSP/D009EE9/MarkTest.sb</body></html>
```

The DXM Controller creates a new GET message to the webserver to start the file transfer.

```
GET /RSP/D009EE9/MarkTest.sb
```

The server begins sending the file to the DXM Controller.

```
TCP segment data (1380 bytes)
TCP segment data (588 bytes)
[2 Reassembled TCP Segments (1968 bytes): #1162(1380), #1163(588)]
```

The DXM Controller is not rebooted after a ScriptBasic file download.

If the XML file verifies correctly, the DXM Controller reboots then attempt to load the new XML file. If the XML file is loaded correctly, the DXM Controller sends a GET message back to the webserver. The payload is called an initialization packet because it includes Meta data from the Controller.

- lat (latitude)
- lon (longitude)
- mod (model number)
- ser (serial number)
- fwv (firmware version)
- ipaddr (ip address)
- maddr (mac address)

```
GET /push.aspx?id=b3ce7448-b38b-4ea7-b994-
17162a779eac&pkt=20161111183709&lat=0.000000,0&lon=0.000000,0&mod=000000&ser=000000&fwv=v
1.00.00&ipaddr=10.10.82.132&maddr=M:00:23:d9:ff:02:27&reg1=0&reg6=0&reg8=0
```

Transfers on cellular will have more metadata in the init packet about the cellular modem and connection. Some metadata is not available depending upon the cellular modem type.

- csq (cellular signal strength)
- cnum (cellular phone number)
- cfw (cellular modem firmware)
- meid (mobile equipment identifier)

Host-Initiated Protocol

The host-initiated protocol allows a host system to send commands directly to the DXM Controller. The commands include the ability to access local Modbus registers, remote Modbus registers, RTC data, IP settings, and file system functions. The DXM controller can also be forced to immediately push data and forced to be reset

The DXM API commands are used from a host system to interact with the DXM Controller. When using a cellular connection the carrier must provide a fixed IP address for the DXM controller to create a virtual point to point private network by which to communicate. The API commands are simple ASCII text strings sent from a host to the DXM controller using a TCP packet on port 8844.

Basic command structure is started by the letters “CMD” immediately followed by four numeric numbers that define the function, for example CMD0001. The return response to a command will have a similar seven-character format, RSP0001.

Local registers 1-900 are 32-bit integer values. Accessing these registers using the API commands will access all 32-bits. Accessing these registers via Modbus will only use the lower 16-bits.

Local Registers 1001-1900 are floating point registers in IEEE 754 format. Accessing these registers using Modbus requires the host system to read two registers to form one floating point value (registers 1001, 1002 form one value). Accessing these registers with the API commands requires the host to request only the first Modbus register. Reading 1001 returns the concatenated value from registers 1001 and 1002. Reading two floating point registers starting at register 1001 results in two 32-bit values returned, Local Registers 1001-1002, 1003-1004.

Following the fixed, seven-character command code is a series of comma separated parameters for the command code. There can be a space as a delimiter between the command code and the parameters, but it is not required.

The common parameter definitions are:

StartReg – Starting address of the first Modbus register to read, 1 - 65535
RegCount – The number of Modbus registers to read, 1 - 256
SlaveID - The slave ID defines the Modbus Slave ID for remote register access. For Local register access, set this parameter to zero.
Modbus CMD – A parameter for remote registers read/write functions to describe the Modbus command that will be used to access the data.
- 0 = Holding Register
- 3 = Coil
- 4 = Discrete Input
- 5 = Input Register
- 6 = Single Coil
- 7 = Single Register

Timeout –The timeout parameter (ms) defines the maximum of time the controller waits for a reply message from a remote device. The timeout parameter is for remote register access only. Leave at zero to apply the default timeout parameter controlled by the configuration tool.
CSV Data – Flexible length comma separated data fields

Read Local Register

This command reads one or more local registers of the DXM controller. The response includes the starting register followed by the read data. The parameters are shown below. Set the Slave ID, Modbus CMD, and Timeout fields to zero.

```
CMD0001 StartReg, RegCount, Slave ID, Modbus CMD, Timeout
RSP0001 StartReg, CSV Data
```

Example: Read 10 local registers in the DXM controller starting at address 5. The response includes the starting address 5 and the read data.

```
CMD0001 5,10,0,0,0
RSP0001 5,1111,2222,3333,4444,5555,6666,7777,8888,9999,1010
```

Write Local Register

This command writes one or more local registers of the DXM controller with the supplied data. The response is RSP0002. Set the Slave ID, Modbus CMD, and Timeout fields to zero.

```
CMD0002 StartReg, RegCount, SlaveID, Modbus CMD, Timeout, CSV data
RSP0002
```

Example: Write 5 local registers in the DXM controller starting at address 25. The response is only an acknowledge, RSP0002

```
CMD0002 25,5,0,0,0,1111,2222,3333,4444,5555
RSP0002
```

Read Remote Register

This command reads one or more remote Modbus registers. The Modbus slave device can be an internal DXM controller device or a device connected to the external RS485 bus. The response includes the starting register followed by the read data. The parameters are shown below. Set the Timeout field to zero.

```
CMD0003 StartReg, RegCount, SlaveID, Modbus CMD, Timeout
RSP0003 StartReg, CSV Data
```

Example: Read 10 Modbus registers from the I/O board (Slave ID 200) on the DXM controller starting at address 1. The response includes the starting address 1 and the read data.

```
CMD0003 1,10,200,0,0
RSP0003 1,1111,2222,3333,4444,5555,6666,7777,8888,9999,1010
```

Write Remote Register

This command writes one or more remote Modbus registers. The Modbus slave device can be an internal DXM controller device or a Modbus slave device connected to the external RS485 bus. The response is RSP0004. The parameters are shown below. Set the Timeout field to zero.

```
CMD0004 StartReg, RegCount, SlaveID, Modbus CMD, Timeout, CSV data
RSP0004
```

Example: Write 5 Modbus registers to an externally connected Modbus slave device, slave ID 11, starting at address 25. The response is only an acknowledge, RSP0004

```
CMD0004 25,5,11,0,0,1111,2222,3333,4444,5555
RSP0004
```

Invoke Register Push

Sending this command causes the device to immediately push data to the defined IP address. Mobile initiated data transfers are accomplished with a HTTP GET or POST.

Example: Force an immediate push from the DMX Controller.

```
CMD0006
RSP0006
```

Clear Http.log File

Sending this command erases the Http.log file stored in the root directory of the SD card. The Http.log file accumulates all missed push messages to a webserver. If the DXM Controller was disconnected from the webserver for any length of time, it is a good practice to delete this file if the data has no value.

Example: Erase the Http.log

```
CMD0008
RSP0008
```

Get Default IP

This command reads the IP address associated to the device.

```
CMD0016
RSP0016 IP Address
```

Example: Read the IP address. The response includes the IP address.

```
CMD0016
RSP0016192.168.0.1
```

Get Default Subnet

This command reads the subnet mask associated to the device.

```
CMD0017
RSP0017 subnet mask
```

Example: Read the subnet mask. The response includes the subnet mask setting.

```
CMD0017
RSP0017255.255.254.0
```

Get Default Gateway

This command reads the gateway setting associated to the device.

```
CMD0018
RSP0018 gateway
```

Example: Read the gateway. The response includes the gateway IP address setting.

```
CMD0018
RSP0018192.168.0.10
```

Get Modbus SLID

This command reads the Modbus Slave address of the DXM Controller. The Modbus Slave address is used when a host communicates with the DXM Controller using the secondary Modbus Slave RS485 port.

Example: Read the DXM Controller Modbus Slave ID; returns a value of 24.

```
CMD0028
RSP002824
```

Set RTC

Set the real time clock in the DXM Controller. The time is UTC.

```
CMD0100 Year,Month,Day,Hour,Minute,Second
RSP0100
```

Example: Set the real time clock to 2014/07/24 15:23:00

```
CMD0100 2014,7,24,15,23,0
RSP0100
```

Get RTC

Read the real time clock in the DXM Controller. The time is UTC.

```
CMD0102
RSP0102 timestamp
```

Example: Read the DXM Controller time.

```
CMD0102
RSP0102 2014,7,24,15,23,0
```

Get Firmware version

Reads the firmware version of the DXM processor board using this API command.

Example: Read the DXM Controller firmware version; returns a value of 24.

```
CMD0103
RSP0103v0.04.52,fw182124
```

Get MAC Address

This command reads the MAC address on the device.

```
CMD0112
RSP0112 MAC address
```

Example: Read the MAC address. The response includes the MAC address of the device.

```
CMD0112
```

```
RSP011200:23:d9:ff:00:11
```

Get Model Number

This command reads the model number on to the device.

```
CMD0113
RSP0113 model number
```

Example: Read the model number. The response includes the factory model number of the device. (Example: 178325)

```
CMD0113
RSP0113178325
```

Get Serial Number

This command reads the device serial number on the device.

```
CMD0114
RSP0114
```

Example: Read the serial number. The response includes the device serial number of controller.

```
CMD0114
RSP0114123456
```

Reset Device

Send a reset command to the DXM Controller. A response is not returned.

```
CMD0200
```

Cellular Provisioning

The CDMA cellular modem must be provisioned on the wireless network before it can be used by the DXM Controller. After a wireless plan is attached to the MEID of the modem, the cellular modem can be provisioned using the DXM LCD display menu system or by sending this API command.

Example: Start a provisioning process with the Cellular modem. This expects the cellular modem to be installed and a wireless plan in place. The DXM Controller responds immediately, but the provisioning process will take a few minutes to complete with the wireless carrier.

```
CMD0202
RSP0202
```

Push Webserver Page

Set the Webserver parameter Page using this API command. The next reboot cycle loads the Page parameter setting from the XML configuration file.

Example: Change the Webserver Page parameter.

```
CMD0205 /push.aspx
RSP0205
```

Push Webserver Server/IP

Set the Webserver parameter Server/IP using this API command. The next reboot cycle loads the Server/IP parameter setting from the XML configuration file.

Example: Change the Webserver Server/IP parameter.

```
CMD0206 demopush.sensonix.net
RSP0206
```

Push Webserver Host Header

Set the Webserver parameter Host Header using this API command. The next reboot cycle load the Host Header parameter setting from the XML configuration file.

Example: Change the Webserver Host Header parameter.

```
CMD0207 MyHostHeader
RSP0207
```

Push Webserver Site ID

Set the Webserver parameter Site ID using this API command. The next reboot cycle loads the Site ID parameter setting from the XML configuration file.

Example: Change the Webserver Site ID parameter.

```
CMD0208 b677648d-645d-43cf-9d17-7b74764b0901
RSP0208
```

File Transfer

The file content is transferred in normal text like is normally found in the file itself. The exception to this is the newline characters (carriage return, line feed), which must be substituted as follows:

```
CR: 0x0D substitute with 0x1E
LF: 0x0A substitute with 0x1F
```

These substitutions are made before any checksums or CRCs are calculated.

Send the device a CMD1003 at the beginning of a file transfer; this force closes any file that may have been interrupted or left in a bad state.

The process for transferring file consists of sending the ‘Open File’ command, followed by a series of ‘Send Data’ commands, and finally a ‘Close File’ command

Possible API responses that can occur on any command:

```
UNSUPPORTED
API_BAD_COMMAND
API_BAD_SYNTAX
QFULL
IPC_PARAMS
```

API FTP Begin (Open File Command)

This command requests a file to transfer between the host system and the DXM controller; similar to an 'Open File' command. Required parameters include filename, read or write (0/1), file size in bytes and control flags. When writing an XML configuration file, the filename MUST always be "WLConfig.xml".

```
CMD1001 filename, r/w, file size, flags
RSP1001 result

Filename = full filename of the file to transfer
R/W = read flag = 0; write flag = 1
File size = file size in bytes
Flags = normally zero, specifying a 1 will show diagnostic messages on the console

Possible responses, prefixed by RSP1001:
OK (or blank)
API_FILE_ALREADY_OPEN
API_NO_MEMORY
API_INVALID_FILESIZE
API_NO_FILE
```

Common error responses would be API_NO_FILE if the file cannot be found or API_NO_MEMORY if the file specified is too large to fit into available memory.

Example: Open the file to write, WLConfig.xml; byte length of the file is 4011. A successful command returns RSP1001

```
CMD1001 WLConfig.xml,1,4011,0
RSP1001
```

API FTP Data (Send / Deliver Command)

This command transfers a data packet between the host system and the DXM controller. The maximum data packet size is 512 bytes. The packet number parameter facilitates multiple data packets to form a complete file. Required parameters include Length, CRC, Packet number, and Data. The parameters vary between a write operation and a read operation.

```
Write:
CMD1002 length, CRC, packet number, data
RSP1002

Length = fragment length, maximum fragment length is 512 bytes
CRC = Checksum (Cyclic Redundancy Check), follows Modbus CRC method.
Packet Number = packet number for this fragment.
Data = Fragment data from 1 to 512 bytes of ASCII data.

Possible responses, prefixed by RSP1002:
        API_BAD_DATA
        API_NO_PAYLOAD
        API_PARAM_ERROR
        API_BAD_SEQUENCE

Read:
CMD1002 packet number
```

```
RSP1002 packet number, CRC, data
```

CRC = Checksum (Cyclic Redundancy Check), follows Modbus CRC method.
Packet Number = the file content is delivered in packets of 512 bytes, which are normally read sequentially, though reading in order is not required.
Data = Fragment data from 1 to 512 bytes of ASCII data.

Possible responses, prefixed by RSP1002:

```
API_BAD_DATA
API_NO_PAYLOAD
API_PARAM_ERROR
API_BAD_SEQUENCE
```

Example: Send Data Command (write) (note the last two characters are the substitutions for carriage return/line feed):

```
CMD1002 43,7947,1,<?xml version="1.0" encoding="us-ascii"?>-
```

Example: Send Data Command (read) (note the last two characters are the substitutions for carriage return/line feed):

```
CMD1002,1
RSP1002512,2f31,<?xml version=”1.0” encoding="us-ascii"?>
```

When the end of the file has been reached:

```
RSP10020,ffff,EOF
```

API FTP End (Close File)

This command closes the file.

```
CMD1003
RSP1003
```

Possible responses, prefixed by RSP1003:

```
None
IPC_PARAMS
```

The IPC_PARAMS response is issued when the amount of transferred data does not match what was specified in the ‘Open File’ command. Also, intentionally mismatching the byte counts is a way to cancel a transfer in progress.

Example: XML Configuration File Write

Writing an xml configuration file to the device consists of sending the API FTP Begin command, followed by a series of API FTP Data commands, and finally an API FTP End command. The filename for the XML configuration file MUST be "WLConfig.xml".

Close command, this forces a clean beginning, if a file transfer was left in a bad state prior to this request.

1. Open WLConfig.xml for writing, 367 bytes
2. Send the data, 367 bytes, crc = 7947, packet number= 1, (replacing CR, LF)
3. Close.

```
CMD1003
```

```
CMD1001 WLConfig.xml,1,367,0
CMD1002 367,7947,1,<?xml version="1.0" encoding="us-ascii"?>-
CMD1003
```

Example: XML Configuration File Read

Reading an xml configuration file from the device consists of sending the API FTP Begin command, followed by a series of API FTP Data (Deliver data) commands, and finally an API FTP End command.

Close command, this forces a clean beginning, if a file transfer was left in a bad state prior to this request.

1. Open file WLConfig.xml for reading
2. Request first packet of data
3. Close file

```
CMD1003
CMD1001 WLConfig.xml,0,0,0
CMD1002 1
RSP1002512,2f31,<?xml version=”1.0” enc … (abbreviated)
When the end of the file has been reached:
RSP10020,ffff,EOF
CMD1003
```

File System List

This command reads the device file system and returns a delimited list of file names. The directory parameter if omitted will give the directory listing of root.

```
CMD1004 <directory>
RSP1004 Delimited File List
```

Example: Read the file system list.

```
CMD1004
RSP1004  CmVmon.txt  GPS_Read.sb  TestScript.sb  EOL
```

File System File Exists

This command reads the device file system and returns a zero/one if the file exists on the device.

```
CMD1005 filename
RSP1005 0 = doesn’t exist, 1 = exists
```

Example: Look to see if a file exists.

```
CMD1005 GPS_Read.sb
RSP1005 1
```

File System Delete

This command deletes a file in the file system.

```
CMD1006 filename
```

Example: Delete file.

CMD1006 GPS_Read.sb

File System Rename

This command renames a file in the file system.

CMD1007 currentfilename, newfilename

Example: Rename a file in the file system.

CMD1007 GPS_Read.sb

File System List Extended

This command reads the device file system and returns a CRLF delimited list of names with added attributes, date modified, time modified fields are coma separated. The directory parameter if omitted will give the listing of root files.

CMD1008

RSP1008 *Attributes, Date Modified, Time Modified, Name*

Attributes are of flag type with bitmask (note long filename is a combination of the low four bits)

0x01 : Read only
0x02 : Hidden
0x04 : System
0x08 : Volume label
0x0F : Long filename entry
0x10 : Directory
0x20 : File

Date:[15:9] [8:5] [4:0]

Bits15-9 : number of years since 1980
Bits 8-5 : non-zero month
Bits 4-0 : non-zero day of month

Time: [15:11] [10:5] [4:0]

15-11 : Hours since midnight (0-23)
10-5   : Minute of the hour (0-59)
4-0   : Second (0-59)

Example: 0000482c,00006745

Year   : 0x24 = 36 + 1980 = 2016
Month : 0x1 = 1   = January
Day   : 0x0c = 12   = 12th

Hour   : 0x0c = 12   = noon
Minute : 0x3a = 58
Second : 0x05 = 5   = 12:58:05 pm

```
CMD1008
#
RSP1008
00000020,0000482c,00006745,TCS_Mgage.sb
00000020,00004839,00006425,CmVMon.txt
00000020,0000482b,000081ee,HttpLog.txt
00000010,00002800,00000000,_sxi
00000020,0000482c,00009f48,SystemPowerSavePoll.sb
00000020,0000482b,000083e3,MultiHop.sb
00000020,0000482b,0000805a,LogFile1.txt
00000020,0000482e,00008b1e,SystemPowerSavePoll2.sb
EOL
```

The following examples read subdirectories from the root directory. This example shows reading "_sxi" directory and then reading "_sxi/20160111" directory

```
CMD1008 _sxi
#
RSP1008
00000010,0000482b,00008075,20160111
EOL
```

```
CMD1008 _sxi/20160111
#
RSP1008
00000020,00004796,00008b24,20160111160353.txt
00000020,0000482b,00008125,20160111161202.txt
EOL
```

DXM API Command Summary Table

		Comma separated parameters					
Function	**Command/ Response**	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**	**Param 6**
Read local register	CMD0001	StartReg	RegCount	0	0	0	
	RSP0001	StartReg	CSV data				
Write local register	CMD0002	StartReg	RegCount	0	0	0	CSV data
	RSP0002	-					
Read remote register	CMD0003	StartReg	RegCount	SlaveID	Modbus CMD	Timeout	
	RSP0003	StartReg	CSV data				
Write remote register	CMD0004	StartReg	RegCount	SlaveID	Modbus CMD	Timeout	CSV data
	RSP0004	-					
Invoke register push	CMD0006						

		Comma separated parameters					
Function	**Command/ Response**	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**	**Param 6**
	RSP0006						
Clear Http.log File	CMD0008						
	RSP0008						
Get Default IP	CMD0016						
	RSP0016	IP Address					
Get default* subnet	CMD0017						
	RSP0017	Subnet Mask					
Get default gateway	CMD0018						
	RSP0018	Gateway					
Get Modbus SLID	CMD0028						
	RSP002824						
Set RTC	CMD0100	Year	Month	Day	Hour	Minute	Second
	RSP0100	-					
Get RTC value	CMD0102	-					
	RSP0102	Date/time					
Get Firmware version	CMD0103						
	RSP0103	v0.04.52,fw182124					
Get MAC address	CMD0112	MAC					
	RSP0112	MAC address					
Get Model Number	CMD0113						
	RSP0113	Model Number					
Get Serial Number	CMD0114						
	RSP0114	Serial Number					
Reset Device	CMD0200	-					
Cellular Provisioning	CMD0202						
	RSP0202						
Push Webserver Page	CMD0205	/push.aspx					
	RSP0205						
Push Webserver Server/IP	CMD0206	demopush.sensonix.net					
	RSP0206						
Push Webserver Host Header	CMD0207	MyHostHeader					
	RSP0207						

		Comma separated parameters					
Function	**Command/ Response**	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**	**Param 6**
Push Webserver Site ID	CMD0208	b677648d-645d-43cf-9d17-7b74764b0901					
	RSP0208						
API FTP Begin	CMD1001	Filename	R/W (0/1)	File Size	Flags		
	RSP1001	Result					
API FTP Data	CMD1002	Length	CRC	Line Number	Data		
	RSP1002						
API FTP End	CMD1003						
	RSP1003						
FileSystem List	CMD1004	Directory					
	RSP1004	Delimited File List					
FileSystem File Exists	CMD1005	Filename					
	RSP1005	0=Doesn't exist 1 = Exists					
FileSystem Delete	CMD1006	Filename					
	RSP1006						
FileSystem Rename	CMD1007	Current Filename	New name				
FileSystem List Attributes	CMD1008	Directory					
	RSP1008	Attributes	Date Modified	Time Modified			

API Command Example

The following example shows an API command to read a local register of the DMX controller. The first TCP fragment shows the ASCII data sent to 10.10.82.179 on port 8844 (the API port). The data is "CMD00011,1,0,0,16000" The API read command for a local register is CMD0001. The important data following the API command is the starting register (1), followed by the number of registers (1). The data after the number of registers is discarded. The next TCP fragment shows the response.

Source	Destination	Protocol	Info
10.10.83.188	10.10.82.179	TCP	55245→8844 [PSH, ACK] Seq=1 Ack=1 Win=16445440 Len=22

```
Frame 26: 76 bytes on wire (608 bits), 76 bytes captured (608 bits)
Ethernet II, Src: 6c:62:6d:60:18:91 (6c:62:6d:60:18:91), Dst: 00:23:d9:ff:00:92 (00:23:d9:ff:00:92)
Internet Protocol Version 4, Src: 10.10.83.188 (10.10.83.188), Dst: 10.10.82.179 (10.10.82.179)
Transmission Control Protocol, Src Port: 55245 (55245), Dst Port: 8844 (8844), Seq: 1, Ack: 1, Len: 22
        Source Port: 55245 (55245)
        Destination Port: 8844 (8844)
```

```
          Data (22 bytes)

0000      43 4d 44 30 30 30 31 31 2c 31 2c 30 2c 30 2c 31          CMD00011,1,0,0,1
0010      36 30 30 30 0d 0a                                        6000..
```

The read data response is sent back with the API command RSP0001. After the command is the starting address (1) followed by the data, 123456.

Source	Destination	Protocol	Info
10.10.82.179	10.10.83.188	TCP	8844→55245 [PSH, ACK] Seq=1 Ack=23 Win=1478 Len=16

```
Frame 27: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
Ethernet II, Src: 00:23:d9:ff:00:92 (00:23:d9:ff:00:92), Dst: 6c:62:6d:60:18:91 (6c:62:6d:60:18:91)
Internet Protocol Version 4, Src: 10.10.82.179 (10.10.82.179), Dst: 10.10.83.188 (10.10.83.188)
Transmission Control Protocol, Src Port: 8844 (8844), Dst Port: 55245 (55245), Seq: 1, Ack: 23, Len: 16
          Source Port: 8844 (8844)
          Destination Port: 55245 (55245)
          Data (16 bytes)
0000      52 53 50 30 30 30 31 31 2c 31 32 33 34 35 36 2c          RSP00011,123456,
          Data: 52535030303031312c3132333435362c
```

The following TCP fragment is an example of the API write command (CMD0002)and the response. The API write command (CMD0002) is followed by the starting register address (1) and the number of registers to write (10). The write data is 1111,2222,3333,4444,5555,6666,7777,8888,9999 and 1010. The next TCP fragment shows the response. (RSP0002)

Source	Destination	Protocol	Info
10.10.83.188	10.10.82.179	TCP	56402→8844 [PSH, ACK] Seq=1 Ack=1 Win=16445440 Len=74

```
Frame 234: 128 bytes on wire (1024 bits), 128 bytes captured (1024 bits)
Ethernet II, Src: 6c:62:6d:60:18:91 (6c:62:6d:60:18:91), Dst: 00:23:d9:ff:00:92 (00:23:d9:ff:00:92)
Internet Protocol Version 4, Src: 10.10.83.188 (10.10.83.188), Dst: 10.10.82.179 (10.10.82.179)
Transmission Control Protocol, Src Port: 56402 (56402), Dst Port: 8844 (8844), Seq: 1, Ack: 1, Len: 74
          Source Port: 56402 (56402)
          Destination Port: 8844 (8844)
Data (74 bytes)
0000      43 4d 44 30 30 30 32 31 2c 31 30 2c 30 2c 30 2c          CMD00021,10,0,0,
0010      31 36 30 30 30 2c 31 31 31 31 2c 32 32 32 32 2c          16000,1111,2222,
0020      33 33 33 33 2c 34 34 34 34 2c 35 35 35 35 2c 36          3333,4444,5555,6
0030      36 36 36 2c 37 37 37 37 2c 38 38 38 38 2c 39 39          666,7777,8888,99
0040      39 39 2c 31 30 31 30 2c 0d 0a                            99,1010,..
```

Source	Destination	Protocol	Info
10.10.82.179	10.10.83.188	TCP	8844→56402 [PSH, ACK] Seq=1 Ack=75 Win=1426 Len=7

```
Frame 235: 61 bytes on wire (488 bits), 61 bytes captured (488 bits)
Ethernet II, Src: 00:23:d9:ff:00:92 (00:23:d9:ff:00:92), Dst: 6c:62:6d:60:18:91 (6c:62:6d:60:18:91)
Internet Protocol Version 4, Src: 10.10.82.179 (10.10.82.179), Dst: 10.10.83.188 (10.10.83.188)
Transmission Control Protocol, Src Port: 8844 (8844), Dst Port: 56402 (56402), Seq: 1, Ack: 75, Len: 7
          Source Port: 8844 (8844)
```

```
            Destination Port: 56402 (56402)
    Data (7 bytes)

    0000  52 53 50 30 30 30 32                                          RSP0002
            Data: 52535030303032
```