SnIP Modem Control Guide

SnIP875 Technical Note

Revision History

Rev 0.1	8-15-2009	Initial Release.
Rev 0.2	5-25-2010	Update for new modem polling status command.
Rev 0.3	7-28-2010	Update for new redundancy methods.
Rev 0.4	2-07-2011	Update alias description, polling methods, examples.
Rev 0.5	4-09-2011	Update for new virtual LO and RF use.
Rev 0.6	10-16-2011	Update for new Global configuration command.

1.0 SnIP Modem Control Overview

The SnIP is a fairly complete Linux computer and contains a standard set of over 100 individual programs, scripts and other elements required for proper functioning. It also has a small separate Operating System “kernel” which is the base of Linux and held in a separate location from other files.

The SnIP has 2 major uses: 1) to act as an Ethernet/IP data interface to the modem, and 2) to provide monitor and control functions for the modem both directly and via the Ethernet/IP interface. The SnIP875 is designed to do this for both the PSM-500 and PSM-4900 series of satellite modems. This document describes the monitor and control functions available in the SnIP875.

There are 6 main methods of monitor and control of the modem via the SnIP:

1. Login into the Linux OS command line shell either locally using the rear panel console port or remotely via Ethernet using a Telnet or SSH virtual console session. Multiple command line programs are available but the main ones are the “m500ctl” and “m49ctl” control programs for the PSM-500 and PSM4900 series modems respectively.
2. Using the SnIP’s built-in mp-save and mp-set modem configuration commands which allow multiple modem parameter settings to be saved and executed from a single file named a modem profile. See Section 2 of this document
3. Using the SnIP’s built-in Web Server, accessing the SnIP via an HTTP or HTTPS connection. See Section 3 of this document.
4. Using the SnIP’s built-in SNMP Server and Agent. This currently only includes SnIP control and not modem control. See Section 4 of this document and the separate document titled “SnIP SNMP Guide”.
5. Remotely executing SnIP and modem control commands using SSH secure key authentication procedures. This method allows both manual and external program based control using the SnIP control programs mentioned in method 1 or 2. See the separate document titled “SnIP Remote Execution Guide”.
6. Custom programs that run on the SnIP and remote computers for socket based network and modem control.

All of these methods except at least one case of method 6 use a single program built into the SnIP, which provides a common control interface between the SnIP and the modem. This Tech Note described the procedures for using the “m500ctl” and companion “m49ctl” programs, which provides this capability. It also provides basic instructions on use of the modem profiling, web server and SNMP functions.

The “m500ctl” and companion “m49ctl” programs are almost identical externally encompassing only the differences necessary to the function and control structure between the PSM-500 and PSM-4900 modems. Most of the single parameter commands are the same between the two modem control programs however there are few significant underlying differences. The m500ctl control program contains many new “virtual” commands which allow a single control for multiple related paramters, such as complete FEC setup with a single command. In addition the the Unit Global configuration command and its related ability to configure the remote end modem via the SnIP are only available with m500ctl and a companion PSM-500 series modem running software version 1.22 or above.

The physical link between the SnIP and host modem used for monitor and control is an RS-232 serial port inside the modem. The protocol is the standard modem binary packet protocol used for remote modem control as described in the respective modem’s Appendix B. It is also possible to control one or more additional modems using the same programs and an external RS-485 connection as described later. The main physical control connections are shown in Figure 1 below.

User control for the modem can interface with either the TCP/IP virtual ports or the physical port at the rear panel console connection.

Modem control can be directed to either the host modem on the internal connection or external modems connected via an RS-485 bus.

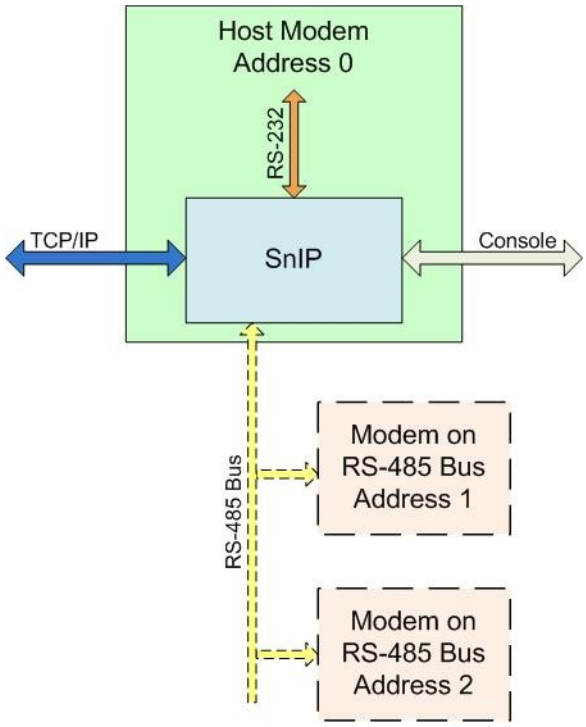

Figure 1 - Local Modem Control

When using the PSM-500 series of modems an expansion of control is possible extending to the remote modems linked via satellite. This control expansion requires modem software version 1.22 or above and m500ctl version 0.53 or above. In addition the modem multiplexer must be enabled on both ends of the link with the MCC channel enabled, preferably at 1200 baud or higher. The multiplexed modem control channel, MCC, provides the communications path for remote link control. This MCC channel represents a very small increase in bandwidth, usually about 1% or less in most links. A depiction of this extended control is shown in Figure 2 below.

Extreme care must be taken when using remote modem control to avoid losing link connection which results in loss of modem control. Section 3 Later describes methods for safe remote modem control.

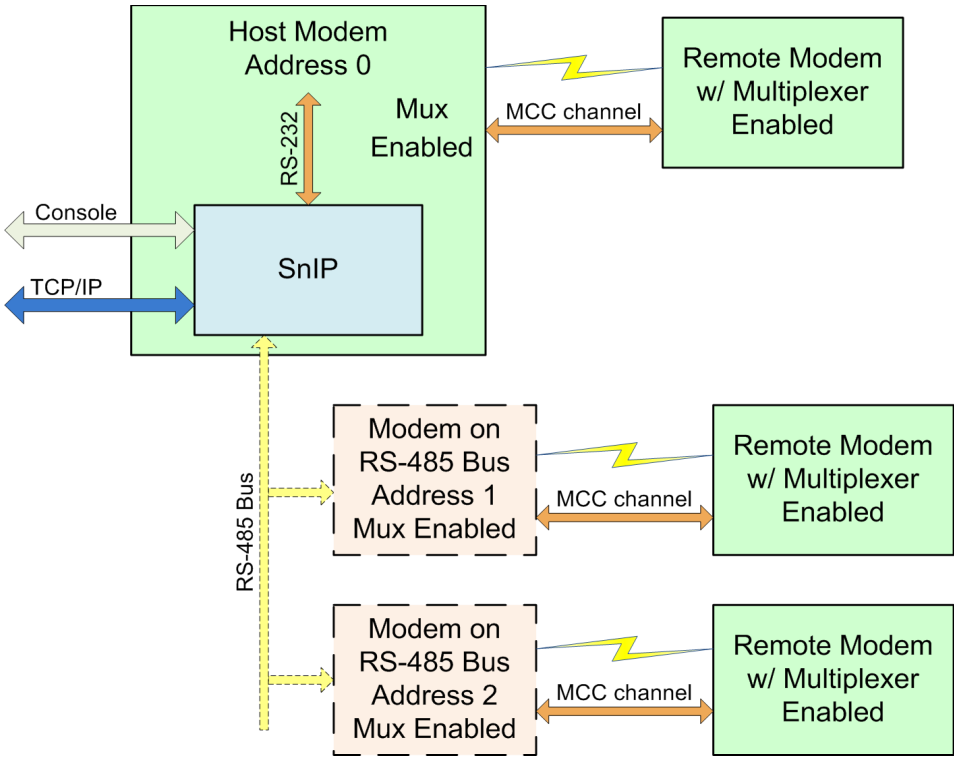

Figure 2 - Extended Local and Remote Modem Control

As you can see this control scenario allows control of an entire small network or modems via a single SnIP, even if it is not the modem's data interface, but just serving as a control element.

The m500ctl and m49ctl programs are written in the C programming language and cross-compiled to run on the Freescale big endian processor used in the SnIP. Both programs contain special routines such as semaphores to control access to the single serial modem connection from multiple programs simultaneously. That is the programs are designed to handle modem requests via the command line, web interface and SNMP or others simultaneously, queuing requests on a first come, first served basis. This is shown in the diagram of Figure 3 below.

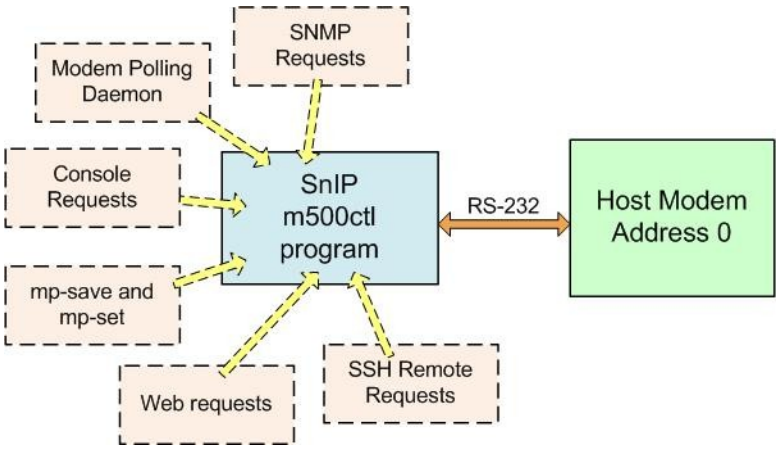

Figure 3 - m500ctl Request Sources

The serial link to the modem is not fast, running at 9600 bps, and many command responses have packet lengths over 100 bytes, meaning that a typical command and response may take up to ½ second for completion. Plus there are multiple sources of requests as shown in Figure 3. That means that making constant fast requests for information every 1 second are not going to work. The response times in later versions are vastly improved due to the automatic use of a local database cache as described in Section 1.1.5, but generally it does no good to request updates faster than every few seconds. Modem status “latching” also prevents loss of transient events making fast updates normally unnecessary.

1.1 m49ctl and m500ctl Applications – Command Line Use

These are Linux command line applications that accept relatively easy to remember mnemonics for monitor and control. The programs convert the input commands into binary packet messages between itself and the modem, taking the modem response and formatting it into a man or machine-readable form. The human readable or “verbose” result is directly usable at the console or via telnet or SSH from anywhere accessible to the Ethernet interface.

The “m49ctl” is intended for the M5 series of PSM-4900 modems and uses its binary packet protocol while the “m500ctl” program is for use with M500 series of PSM-500 modems. Since the M500 series has M5 compatible commands, the m49ctl program could potentially be used with either modem type within the limitations of the M5 capabilities.

Use of these applications remotely requires that a SnIP IP address has been set, and that a remote console is first logged into. The applications are called from the console or telnet/SSH session terminal command line.

The basic usage is to type the program name itself followed by optional switch controls followed by the unit address of the modem and then a sequence of one or more command and parameter pairs. The unit address of 0 is used for the modem which hosts the SnIP – i.e. the SnIP is installed within. The usage statement from the program itself is shown below.

Note:
In 2010 we released a newer series of SnIP filesystem (0.6.xx and above) which can use modified M500 series modem software allowing the creation and use of a local database within the SnIP temporary directory. This local cache of information provides considerable speed improvements for most requests from the m500ctl program. The local database is demand driven and only keeps data previously requested. Combined with the new m500ctl “up” command it allows creating efficient management or monitor and control functions. See Section 1.1.7 of this document which provides more detailed information on the operation of these new procedures. This change does not include the m49ctl software.

```
[snip-hub21:l /root]# m500ctl
M500 Modem Control Version 0.53 - Requires Modem SW Ver. 1.22+
Copyright (C) 2006 - 2011 M. Boutte <mike@datumsystems.com>

Usage:
 m500ctl [-p] [format] [options] ADDRESS Cmd Param..[Cmd Param]
   Format output options
     -v prints output verbose and formatted. 2 letter codes are default verbose
     -r prints status output raw. Used for input to other programs.
     -x prints status output raw with xml tags. For input to other programs.
     -s prints output raw. For SNMP changes return 0 on success or error code.
     -g prints output of ug command in Unit Global configuration format.
   Other options
     -e Tells modem to send command to the far end remote linked modem.
     -t prints time in seconds with us, ms, ds or is commands.
     -m Force read from modem - no caching (default without modem polling).
     -u Update the local database for selected commands.
     -d Forces use of an interface device. e.g. -d dev/ttyUSB0 or -d USB1.
     -c prints Parameter to Command Codes reminder.
     -p shows sent and receive packet contents. Sets -m
     -V prints Version number.
   Notes:
     ADDRESS = 0 (Internal) or 1..253 (External RS-485). No address = 0
     Param = ? to get, or value to set (set in integer increment)
     [Command Param] must be in pairs, but ? is optional.
     Use 'ug help' for help on the ug parameters and order of entry.
 Examples:
     m500ctl -v 0 mif ? to display the internal Mod IF Frequency formatted
     m500ctl 0 mif 71251000 to set the internal Mod IF Frequency
     m500ctl -v 1 mi ? di ? gets both from external modem address 1
     m500ctl 0 mdfec 4,4,2 sets Mod FEC to TPC,CT,Rate 3/4 in one command.
     m500ctl 0 ddmfr 1,4,4,2,0 sets Demod MTOCR value in one command.
        This is useful because it avoids incompatible partial settings.
     m500ctl 0 usn 'Hub to Denver' Note: quotes required with spaces in name.
     m500ctl 0 ug f:/etc/config/rp2.gc runs global config from named file.
     m500ctl -g 1 ug > /tmp/hub-base.gc creates global config file.
```

Figure 4 - m500ctl Command Line Usage

The required command elements are the unit address and at least one command and corresponding command parameter. The program name and all elements or parameters are separated by a space. Therefore the format is

m500ctl[space]Switches[space]Address[space]Command[space]Param.

The optional switches must be preceded by the hyphen as shown, cannot be combined with a single hyphen, and must be used in the order shown. This restriction was removed in m500ctl versions 0.42 and higher.

The **–c** option is always used by itself to show a quick guide to the mnemonic codes.

The **–sp** option if used must be first and tells the program to show both the sent and received binary packets in hexadecimal format. This is mainly used for testing, but would be extremely useful if developing a separate program for modem control.

The **–t** option is only used with the “**us**”, “**ms**”, “**ds**” and “**is**” status commands to print out the current time with the status in seconds. This is useful for record keeping.

The output format is controlled by using one of the –v, -r or –x options to modify the default output format. The default format for two letter commands is verbose because they contain multiple parameters, and are printed in human readable format. All other commands print out raw numeric data by default which can be used by other programs. Two letter commands can be set to output raw numeric data for each parameter separated by carriage returns using the –r switch. By using the –x switch the parameters are separated by spaces on a single line with open and close XML style tags. Generally when another program will be reading the m500ctl output all data is forced to raw format using the –r switch. Examples of this type use are shown in section 2.2. Single parameter outputs can be forced to verbose mode by using the –v switch. More information on output formats is given in Section 1.1.1 below.

Either of these programs uses a 2 to 5 letter mnemonic representing the command and a value as either a "?" for query or a numeric entry to send that value with the command. The command – value are always represented by a pair of entries separated by a space. The mnemonic entries mirror the convention used on the front panel in most cases. The first letter is the first letter of the 4 areas available, either "u", "m", "d", "i" standing for Unit, Mod, Demod or Interface. The second letter represents the packet (which coincides with the column on the front panel display, e.g. "**d**" or Data or "**i**" for IF or "**t**" for Test) and the third is the parameter.

Some examples will clarify the program usage. For example the modulator IF frequency is represented by the letters "**mif**", while the entire modulator IF packet is "**mi**". Staying with this example, to show the modulator IF frequency you would type:

m500ctl 1 mif ?

Try the simple alias commands for these:
*"**mr mif**" displays raw.*
*"**mq mif**" displays verbose.*

And the program might respond with

75125000

representing 75.125 MHz.

The "1" after the command mnemonic is the address of the unit to be read or controlled. The address "**0**" is used for the internal connection to the modem hosting the SnIP card. Any other address is directed out the rear panel DB9 connector on the RS-485 bus. Multiple modems can be connected to this RS-485 bus as long as they all have unique addresses between 1 and 253.

Note also that all input to the programs are **lower case**.

If you wished to have a more easily readable response you could use one of the available switches, "**-v**", to tell the program to provide a verbose output.

m500ctl –v 1 mif ?

And the modem might respond with

Modulator IF Freq = 75.125000 MHz

Try the simple alias commands here:
*"**ms mif 75.125**" sets the value.*

To set the demodulator IF frequency you would replace the "?" with the desired frequency in 1 Hz increments. Therefore to set it to 67.052 MHz, the command would be:

m500ctl 1 dif 67052000

Most commands either have values as above or selections. The selections and numbers used to set them are the same as those used on the front panel. So, to enable any "on/off" parameter, you would use "**0**" for disable or set off, and "**1**" to enable or set on. Values for selections can be found in the modem's Appendix B – Remote Control Protocol. They are also the same values as those used on the front panel. For example to read or set the demodulator modulation mode the selection values are 0 = BPSK, 1 = QPSK, 2 = OQPSK, 3 = 8PSK, 4 = 8QAM, 5 = not used, 6 = 16QAM. The read and write commands for 8QAM might look like:

m500ctl 0 dim ? */* Request the current setting of unit 0 */*

m500ctl 0 dim 4 */* Set unit 0 to 8QAM */*

One difference to note is that the current implementation of the m49 and m500ctl programs normally use the full value input down to the lowest increment possible. The exception is the modulator and demodulator IF frequency which can be entered with a decimal point in MHz values since these represent no ambiguity. Thus in the above example you could have entered "67.052" instead of the full "67052000", or "71" for 71000000 MHz. Another exception is in L-Band modems where the BUC and LNB LO frequencies are entered in MHz values with no decimal points. No known BUCs or LNBs use sub MHz increments today. But for example a modulator data rate of 1.544 could not be used because it is ambiguous possibly representing 1.544 Mbps (1544000 bps) or 1.544 kbps (1544 bps). The data rate therefore always requires use of bps terms.

m500ctl versions 0.51 and above allow very flexible use of the command line switches. These versions also do not need the Unit Address "0" as that is the default.

The "?" for a query for a query is also no longer mandatory.

You can also have multiple commands on a single command line. To see all of the Modulator and Demodulator IF and Data Settings and to set the Modulator IF output on, you would use the command line:

m500ctl 1 mi ? di ? md ? dd ? mie 1

The simple alias commands do not work for multiple entries on one line.

Available switches are "–v" for "verbose", "-r" for raw mode output and "–p" for "show packet", which is used for debugging to see the actual commands sent and received between the SnIP and the modem. There is also a Time output for the status commands only in verbose mode using the "-t" switch. To see the currently available mnemonics us the command "m500ctl –c".

The order of switches was important in m500ctl versions prior to approx. 0.43. They should be used in the order: -sp, -v or r or x, -t. Newer versions allow any order and grouping of switches.

1.1.1 m49ctl and m500ctl Command Set

The currently implemented m500ctl commands can be viewed by using the m500ctl program itself and typing

m500ctl -c

Which will result in a display similar to the one in Figure 5 below.

```
[snip-hub21:l /root]# m500ctl -c
M500 Modem Control Version 0.53
   Parameter to Command Letter Codes Reference
2 letter green codes print verbose packet full info,
3-4 letter codes print/set individual parameters
** The violet codes are virtual - only available on the SnIP. **

 -- Unit -------    -- Modulator -----   -- Demodulator --  -- Interface --
us - Status        ms - Status         ds - Status         is - Status
uss - Red Switch   mi - IF             di - IF             isbr - Reset BERT
usn - ID Name      mif - Freq          dif - Freq
                   miif - IF Freq      diif - IF Freq
                   mirf - RF Freq      dirf - RF Freq
                   milo - LO Freq      dilo - LO Freq
                   mifd - Freq Delta   difd - Freq Delta
                   mio - Offset        dia - Acq Range
                   mil - Cxr Level     dil - Low Level Alm io - Int IO
uc - Config        mie - Cxr Enable    die - Low Eb/No Alm iom - IO Mode
ucs - Store Cnfg   mim - Modulation    dim - Modulation    ioa - IO Async
ucr - Recall Cnfg  mis - Spectrum      dis - Spectrum      ioc - IO Control
ucy1..8 - Delay    mifl - Filter       difl - Filter
                   mit - Mute Control
                   mip - AUPC Enable
                   mipc - AUPC Setup
uo - Ref Osc       md - Data           dd - Data           ie - Ethernet
uos - Osc Source   mdr - Data Rate     ddr - Data Rate     iep - IP Address
                   mdm - Modulation    ddm - Modulation    iem - IP Mask
                   mdf - FEC Type      ddf - FEC Type      iex - MAC Address
                   mdo - FEC Option    ddo - FEC Option
ux - Remote        mdc - Code Rate     ddc - Code Rate
uxp - Protocol     mdfec - FEC T,O,C   ddfec - FEC T,O,C
uxt - Port Type    mdrs - R-S Mode     ddrs - R-S Mode
uxr - Port Rate    mdrp - R-S n,k,d    ddrp - R-S n,k,d
uxa - Port Addr    mdmfr - M,T,O,C,R   ddmfr - M,T,O,C,R
                   mdif - Diff Encoder ddif - Diff Decoder
ur - Redundancy    mds - Scrambler     dds - DeScrambler
ure - Red Enable   mdk - Clock Source  ddk - Clock Source  ia - Alm Map
urc - Send Config                      ddd - Buffer Delay  iat - Alm Test
urs - Switch On                        ddb - Buffer Bits   iab - Alm BER
urh - Hold Time    mm mmz- Mod Mux     dm dmz - Demod Mux  iao - Alm Opt
                   mmm - Mux Mode      dmm - Mux Mode
                   mme - ESC Ovhd Rate dme - ESC Ovhd Rate
                   mmc - MCC Ovhd Rate dmc - MCC Ovhd Rate
                   mmp - ESC Port Type dmp - ESC Port Type
                   mmr - ESC Port Rate dmr - ESC Port Rate
                   mmf - ESC Port Frmt dmf - ESC Port Frmt
                   mmt - ESC CTS Mode  dmt - ESC DTR Mode
                                       dms - ESC DSR Mode
up - Poll Status   mb - BUC            dl - LNB
                   mbf - BUC LO        dlf - LNB LO
                   mbe - BUC Pwr En    dle - LNB Pwr En
                   mbr - BUC 10MHz Ref dlr - LNB 10MHz Ref
                   mbvn- BUC Vmin      dli - LNB Iout
                   mbix- BUC Imax      dlix- LNB Imax
                   mbin- BUC Imin      dlin- LNB Imin      it - Test
ua - Alm Map       ma - Alm Map        da - Alm Map        itt - Terr, LB
ut - Test          mt - Mod Test       dt - Demod Test     its - Sat LB
ug - Global Config mtm - Test Cxr Mod  dtl - IF Loopback   itber-BER Tx,Rx,Dir
```

Figure 5 - m500ctl Command Set and Mnemonics

The four columns here represent commands for the Unit, Mod, Demod and Interface, and as can be seen each command begins with the first letter of one of these names. The second letter is the first letter of the short packet name that the parameter belongs to, which is also the same as the first letter of the column name on the front panel display – examples are “d” for Data, “I” for IF, “m” for Mux, “s” for Status, “a” for Alarm, “t” for Test, etc. The remaining letters are attempts at mnemonics for the particular parameter.

Additional commands may be implemented in the future. Notice that the commands follow the general format of the front panel controls in that they consist of 3 elements to define a particular parameter. In the Excel worksheet idiom they are page, column and row, where page is either Unit, Mod, Demod or Interface (the 4 buttons on the front panel labeled Unit, Mod, Dem, Intf). The rows and columns are symbolic of the up/down and left/right front panel arrow controls. As explained in the modem manual the scheme is synonymous with moving to a particular Excel spreadsheet page, then cell row and column using a computer’s keyboard arrow keys.

1.1.1.1 Virtual and Extended Commands

The display or setting of parameters on the front panel of the modem is limited to a single element at a time. For some inter-related parameters the modem asks for a sequence of entries during the process. The same limitations do not hold for our SnIP based entries, so extended entries allow setting all of the related parameters in a single command.

Three of the commands, Mod Data FEC <**mdfec**>, Demod Data FEC <**ddfec**> and Interface Test BER <**itber**>, are unique in that they require 3 parameters separated by commas when changing. Make certain that you do not have any spaces in the parameters. The FEC commands also have commands to set the individual elements, but the 3 parameter version is recommended because it avoids the possibility that one value is set but not allowed because of the others currently set. The data FEC values can be read directly from the Modem FEC Modes document available on the web or in the manual.

For the M500 there is additionally the **mdmfr** and **ddmfr** commands which set 5 related parameters at once for complete data setup. They are the numeric selections for **M**odulation, FEC **T**ype, FEC **O**ption, FEC **C**ode Rate and **R**eed-Solomon (acronym MTOCR for short). For example “m500ctl 0 mdmfr 11000” sets the modulator to QPSK, Viterbi, Standard Code rate ½ without R-S. This one command plus the IF frequency (mif/dif) and data rate (mdr/ddr) are all that is normally required to set up a modem. The MTOCR can be entered as for example 32001 or more properly 3,2,0,0,1.

Virtual Commands -

m500ctl version 0.45 released in April of 2011 also contains several virtual commands intended to simplify and avoid errors specifically for remote control. The initial set of these are for setting the transmit and receive real IF frequencies plus virtual RF frequencies based on a virtual LO as used in a BUC or LNB.

The Mod BUC and Demod LNB controls on the modem front panel and in m500ctl do already contain a setting for these LO frequencies, but use of these entries poses problems for remote control and it is ***highly*** recommended that these not be used in conjunction with remote control applications. The main problem is that setting one of these LO frequencies requires immediate setting of the RF frequency, and in the intermediate time the real IF frequencies are set to either a maximum or minimum possible and the transmit carrier output is dropped.

Current virtual parameters include:

- **miif** – modulator real IF transmit frequency. This value does not change as **mif** does in an L-Band modem when a BUC LO entry exists – from **mbf**, the web BUC-LNB page or the modem front panel Mod>>BUC>>LO Frequency parameter.

- **mirf** – modulator virtual RF transmit frequency. This value is computed from the combination of miif and the virtual transmit IF to RF translation frequency (milo).
- **milo** – modulator virtual IF to RF translation frequency. For an L-Band modem this would be the BUC LO frequency. LOs are either high side or low side, being either above or below the RF frequency range. For **milo** a high side LO is entered as a negative number.

- **diif** – modulator real IF receive frequency. This value does not change as **dif** does in an L-Band modem when an LNB LO entry exists – from **dlf**, the web BUC-LNB page or the modem front panel Demod>>LNB>>LO Frequency parameter.
- **dirf** – modulator virtual RF receive frequency. This value is computed from the combination of diif and the virtual receive IF to RF translation frequency (dilo).
- **dilo** – modulator virtual RF to IF translation frequency. For an L-Band modem this would be the LNB LO frequency. LOs are either high side or low side, being either above or below the RF frequency range. For **dilo** a high side LO is entered as a negative number.

On power up the SnIP reads the current mod BUC and demod LNB settings plus reads a configuration file at "/etc/config/rf-config". It then saves the initial transmit milo and receive dilo values in the /tmp/mlof and dlof files respectively. After initial setting all LO values are taken from and stored in these files and no changes are reflected in the rf-config file. To set a persistent initial value the settings in the rf-config file should be changed using an editor like vi or nano.

The operation of these parameters varies from the modem's BUC and LNB LO frequencies. The main difference is that setting the **milo** or **dilo** values never changes the real IF transmit or receive frequency, only the virtual RF frequency relative to the LO. Since both the IF and RF frequencies are exposed, the modem's frequency setting can be made from either. The other difference is that there is no guessing about whether the LO is high or low side. Instead positive LO values are always low side and negative values are always the less common high side. The fixed formulas for the relationship between RF and LO are as follows:

Tx RF Freq (mirf) = mod LO (milo) + mod IF (miif) for low side, non-inverting LO
Tx RF Freq (mirf) = mod LO (milo) – mod IF (miif) for high side, inverting LO

and

Rx RF Freq (dirf) = demod LO (dilo) + demod IF (diif) for low side, non-inverting LO
Rx RF Freq (dirf) = demod LO (dilo) – demod IF (diif) for high side, inverting LO

An example would be for a typical BUC the Ku Band LO frequency might be 13050 MHz. Therefore and IF frequency of 1250.75 MHz yields an RF frequency of 14300.75 MHz.

Note: The M5 modem controls available in the m49ctl program are very close to those shown above for the m500ctl program. The main difference is that the M5 modem's FEC controls are simpler than those in the M500 plus they do not contain extended or virtual commands.

1.1.2 The Unit Global Configuration Command - UG

The Unit Global Configuration command, referred to as "UG" or "ug" in the remainder, is a single command packet which can read or set all common configuration parameters in an M500 modem with a single packet. Coupled with m500ctl it allows users to store and configure modems from a single text string. This capability allows easy home base configuration of modems and provides the possibility of easily programming remote modems.

1.1.2.1 Requirements for using UG with a SnIP

A modem equipped with a SnIP only requires that the SnIP's m500ctl program be version 0.52 or later and that the modem software version be 1.22 or greater. It works with all variations of PSM-500 modems, IF or L-Band. The SnIP does not have to be the data interface device.

For configuring a remote modem, that linked modem must also have software version 1.22 or higher. No SnIP is required in the remote linked modem if the transmit and receive multiplexer is enabled on both ends with the MCC channel enabled, preferably at 1200 baud or higher.

1.1.2.2 Setup for using UG with the m500ctl program

These procedures are applicable to either a terminal session in the SnIP or a terminal session via telnet or SSH over the Internet. The basic procedure is as follows:

It is highly recommended that you configure all modems with at least a minimal multiplexer setup in Custom mode and the MCC channel set to 1200 baud or higher. For most links this represents a small added overhead of 1% or less.

1. If remote modem configuration is desired then both modems should have the modulator and demodulator multiplexer enabled in mode 3 (Custom) with the MCC channel set to 1200 baud or preferrably higher, such as 4800 (5) or 9600 (6) baud. The ESC channel can be set to disabled unless it is being used.

2. Issue the configuration string using m500ctl. For example:
   m500ctl ug tx:p1,p2,p3,...:rx:p1,p2,p3,...:mx:p1,p2,p3...:io:p1,p2,p3...
   where p1,p2 are the configuration parameters. To configure the remote modem via the multiplexer MCC channel the "-e" switch is added, for example:
   m500ctl -ed USB0 1 ug tx:p1,p2,p3,...:rx:p1,p2,p3,...:mx:p1,p2,p3...:io:p1,p2,p3...

If you are configuring both modems on the link remember that you must first configure the remote modem and then the local.

1.1.2.3 The UG Configuration String

The modem packet protocol for the Unit Global command is a binary packet with 69 parameters and 69 write enable flag bits related to the parameters. Writing this packet would be very difficult without the companion m500ctl program. The program simplifies the process by several methods, using a text string with parameters organized into 4 categories (tx: for transmit, rx: for receive, mx: for multiplexer and io: for the data interface) plus methods for inputting only those parameters desired. Any parameters not specifically entered are left unchanged in the modem. Parameters within a category are separated by commas, and no spaces are allowed within the string.

Although the modem binary packet requires that parameters are entered in a specific order for execution, the ug command string is organized with the most common parameters first in each category. Incomplete category entries and non-numeric entries are treated as "no-change" allowing setting of only those parameters desired.

The easiest way to show proper formatting and use of the ug configuration string is by multiple examples. But to start you can see the full format of the data and string by using m500ctl to query the ug packet contents.

```
m500ctl ug          [ For the modem hosting the SnIP ]
m500ctl 1 ug    [ For an external modem on the RS-485 bus at Unit address 1 ]
```

That command outputs the ug information in verbose human readable form. In the following examples we will not use the device parameter for clarity, but it may be required in your system. To see the actual full string use the -g formatting switch:

```
m500ctl -g ug
```

and the resulting out should be similar to the following

```
$ m500ctl -g ug

tx:1098000000,-158,0,2048000,1,4,0,4,0,219,201,0,0,0,0,0,0,60,50,-
350,0,1,0:rx:1402500000,3700000,1,4,0,4,0,219,201,0,30000,0,100,0,0,20,-
688,1,0,1,0,11,4,1:mx:0,0,6,0,0,6,0,0,2,0,0,0:io:8,0,0,1,0,0,0,0,0,0
```

This modem happens to be an L-Band which is why the transmit and receive frequencies (first parameter in the tx: and rx: categories) are in the L-Band range. Note also that the command invocation was preceded with "**./**", or dot slash, because the m500ctl program was in the current directory.

As you can see you need some instructions to tell which parameter is which. One way to see that is to type:

```
m500ctl ug help
```

which bring up the following information (formatted here for slightly easier viewing)

```
ug f:config-file-name to get parameters from a file, or

ug tx:p1,p2,p3...p26:rx:p1,p2,p3...p33:mx:p1,p2,p3,p4...p12:io:p1,p2...p10

  tx: params= Freq,Level,Cxr_en,BitRate,Modulation,Fec_Type,Opt,CodeRate,\
     RSmode,RSn,RSk,RSd,Spectrum,Filter,MuteMode,Imped,AUPCen,AUPC_EbNo,\
     AUPC_max,AUPC_min,DiffEnc,Scrambler,ClkSource,MuxMode,ESC_Oh,MCC_Oh

  rx: params= Freq,BitRate,Modulation,Fec_Type,Opt,CodeRate,RSmode,RSn,RSk,RSd,\
     Acq_range,SweepMode,SweepTime,Spectrum,Filter,LowEbNo,LowLvl,Imped,\
     DiffDec,Scrambler,ClkSource,Buff_delay,Buff_Size,FEC_Hold,MuxMode,\
     ESC_Oh,MCC_Oh,ESC_Port,ESC_Rate,ESC_Frmt,ESC_CTS,ESC_DTR,ESC_DSR

  mx: params= Mod_mode,ESC_Oh,MCC_Oh,Dem_mode,ESC_Oh,MCC_Oh,ESC_Port,ESC_Rate,\
     ESC_Frmt,ESC_CTS,ESC_DTR,ESC_DSR

  io: params= IO_mode,Async_Frmt,RTS,CTS,DCD,DTR,DSR,Xdata,Rdata,Rclk

Notes on format:
 => Category labels are used to go to common entry positions. Parameters in
  a category not used are left unchanged. Categories may be in any order.
 => All parameters are numeric only separated by commas, no spaces! Alphanumeric
  entries are placeholders and result in that parameter remaining unchanged.
 => All params after the last entered in a category are left unchanged. Every
  param position from the first to the last entered in a category must have
  either a valid numeric entry or any placeholder text. e.g. '1,275,*,sc,4'
 => The Mod and Demod Mux parameters are repeated for ease in the mx category.
```

Here are more examples of using variations of the configuration string to clarify the instructions:

Enter just a few initial parameters in one category – set demod frequency and bit rate. Everything else in the modem is left unchanged.

```
m500ctl ug rx:72125000,2048000
```

Enter just a few non-contiguous parameters in one category – set demod frequency and FEC Type, Option and Code Rate. The bit rate and modulation in position 2 and 3 below are left

unchanged as are all other modem parameters. Note that any non-numeric entry becomes an empty placeholder, as “*”, “p”, “nc”, or “CxrLevel”, but multiple commas with nothing in between looks like a single comma to the program input.

```
m500ctl  ug rx:72125000,*,p,4,0,4
```

Enter just a few parameters in several categories – set the mux and mod frequency and cxr level. Note the delimiter between categories is the colon “:”. The “-e” switch sent this command to the far end linked modem, which must have already had the multiplexer enabled as well as the local modem. Note that we purposely put the mx before the tx category to show that this order does not matter.

```
m500ctl -e ug mx:3,0,6,3,0,6:tx:72125000,-146
```

The next command following the one above should be to the local modem, and complementary to reconnect to the far end, such as:

```
m500ctl ug multiplexer:3,0,6,3,0,6:rx:72125000
```

which sets the mux identically and the local receive frequency to match the new transmit on the remote end.

Note that the category labels are very flexible. Transmit requires it contains the lowercase letter “t” and the colon, receive requires “r” and the colon, multiplexer requires either “m” or “x” and the colon while interface requires either “i” or “o” and the colon.

1.1.2.3.1 Saving and Getting the UG Configuration String Using a File

The full string would more commonly be used to completely configure a new modem from previously determined settings. The setting could be saved in one or more files and recalled as needed for each modem. Lets say that 5 hub and 5 remote modems are ready to be deployed. The initial base files to configure each of these modems could be created from one hub and one remote remote modem manually configured. Then these two modem configurations are saved to files with the command similar to the following:

```
echo “#  PSM-500 Hub base config 12Nov11” > /home/mike/mconfig/hub-base.gc
m500ctl -g ug >> /home/mike/mconfig/hub-base.gc
```

The first command puts some id info into the file and the second appends the string representing the current modem configuration. You would then do the same for the base remote modem configuration. Next create 5 copies of each renaming them to hub-lc21.gc, hub-lc22.gc … and rem-lc21.gc, rem-lc22.gc … rem-lc25.gc. Modify the parameters in each as desired using a text editor and finally configure the modems with commands such as:

```
m500ctl  1 ug f:/home/mike/mconfig/hub-lc24.gc
```

If the remote modems are already at the remote locations then the file can be emailed to someone there for configuring using a command similar to the one above.

If the remote modem is currently linked with the multiplexer enabled it can be reconfigured using the same command to the local modem but with the -e switch to send it over the link.

```
m500ctl  -e 1 ug f:/home/mike/mconfig/rem-lc24.gc
```

Cautions: The modem will refuse to program most non-valid configuration parameters, which can result in a faulty configuration, especially if sent to a remote modem. It is strongly advised that configurations be tried on local modems first to insure that they are valid. The process of creating

an initial string using the -g formatted ug command output or the separate html based page generator (see below) is also recommended.

Be especially careful in specifying IF frequencies and data rates. Incorrect entries could result not only in “losing” a remote modem, but also outputs that could interfere with adjacent carriers.

1.1.2.4 Using the HTML Page to Generate the UG Configuration String

We also provide an html web page that can be opened on any computer, Windows, Linux or Mac that has a web browser. This page can also be downloaded from our web site and will operate stand alone on any computer with a web browser. The page presents a graphical display of modem parameters that can be set as desired and then the necessary configuration string can be generated in a text box. The string can then be copied and pasted for use with the m500ctl program or put into locally saved text files for later modem configuration. Normal client page protections prevent an html page from actually inputing to and running a program on your computer and therefore it cannot directly configure a modem.

1.1.3 Output Formatting

There are a 3 command line switches that are used to format the output of the programs response to commands. Output formatting can make the response verbose and readable or just raw data in formats suitable for programmable use. There are also special outputs from specific commands that contain useful status information as explained below.

1.1.3.1 Raw and Verbose Outputs

The default output format for two letter commands, or any command that results in multiple response items is verbose. To force these outputs to a raw data format that would be machine readable, use the “-r” switch. In this mode each output parameter is separated by a carriage return. Then a language like Perl, which the SnIP includes, can directly import the information into a standard array.

For example if we take the Unit Status command, “**us**”, and output it in its 3 possible formats, verbose, raw and xml-raw, the results would be:

1.) No forced formatting on 2 letter command, default **verbose** output.

```
[snip-12:1 ~] m500ctl 0 us ?   ← This is the command invocation
Modem - Model: PSM-500,  S/N: 13293,  SW Ver: 1.11.000
Feature Set: M523-8PSK-16QAM
FEC A:  Card 2,  Version 109,  S/N 3001849
        Type 2 - TPC-4k, Viterbi, R-S
FEC B:  Card 0,  Version 0,  S/N 0
        Type 0 - Not Installed
Interface Option: Type 3,  Version 0,  S/N 2010084
Unit Status: Cxr Enabled, Demod Unlocked
        Events: Modem
        Alarms:  Dem   Redun No_Back-Up
        Pending Tests:

Unit ID: LAN Hub Tx1
```

2.) Forced raw output on 2 letter command, **raw** output separated by carriage return.

```
[snip-12:1 ~] m500ctl -r 0 us ?   ← This is the command invocation
"PSM-500"
13293
"1.11.000"
"M523-8PSK-16QAM"
2
109
0
0
3
0
50
0
9
"LAN Hub Tx1"
```

3.) Forced raw xml style output on 2 letter command, **raw** output separated by spaces on single line with **XML** style tags.

```
[snip-12:1 ~] m500ctl -x 0 us ?   ← This is the command invocation
<Unit_Status> "PSM-500" 13293 "1.11.000" "M523-8PSK-16QAM" 2 109 0 0 3 0 50 0 9
"LAN Hub Tx1" </Unit_Status>
[snip-12:1 ~]
```

Notice on both the raw and raw xml style outputs that non-numerical parameters are surrounded by quotation marks. This allows a program to differentiate between numbers and text strings and strings which include spaces.

1.1.3.2 Status Outputs

The 4 status commands, "**us**", "**ms**", "**ds**" and "**is**" contain information useful for multiple applications, for example when polling. The raw and verbose outputs of these commands have 3 elements at the beginning of the responses:

- **Events**
- **Alarms**
- **Tests**

In addition the verbose outputs of these four commands can optionally display the current time before the Events output by adding the "-t" switch. The time is in seconds since January 1, 1970 (GMT) which is the default date on power up without an rdate change (this date has something to do with Linux). In 2007 that number is approx 1,169 Million. The "–t" switch is used before the address and after the "–v" switch if used. Raw outputs assume a computer with its own time is used to read the data, and adding a variable may modify array settings.

The Event flags are one time read flags and indicate if a change has occurred in this parameter since the last read. The read is therefore destructive, resetting the flag to 0. The Alarm and current Test flags are live, indicating the current status.

The verbose outputs group these with labels, and indicate only those currently active. The raw outputs have 3 numbers as the first 3 returned scalar values which contain bit variables for the individual status elements as shown below.

Unit Status Raw Output Events (10th value) - Integer Scalar Value 9

Bit 0 – Modem
Bit 1 – Reference
Bit 2 – Redundancy Switch
Bit 3 – Name
Bit 4 – Test

Modulator Status Raw Output Events (1st value) - Integer Scalar Value 0

Bit 0 – Carrier
Bit 1 – Data
Bit 2 – Bit Clock
Bit 3 – Test

Demodulator Status Raw Output Events (1st value) - Integer Scalar Value 0

Bit 0 – Carrier Lock
Bit 1 – Eb/No
Bit 2 – Carrier Level
Bit 3 – Carrier Offset

Interface Status Raw Output Events (1st value) – Integer Scalar Value 0

Bit 0 – IO Mode
Bit 1 – RTS
Bit 2 – CTS
Bit 3 – DCD
Bit 4 – DTR
Bit 5 – DSR
Bit 6 – Test
Bit 7 – BER Test

To determine the status of any one bit you simply “AND” the value with the appropriate bit value. An example use of the Event flags would be to determine if any reads were necessary. First read the unit status and if the first value is 0 then nothing has changed since the last read. But, if the value were 1 then that would mean that the modem had a value change. You would then read the mod and demod status to see what changed. If the mod status event value was 0 and the demod event value was 1 then the demod carrier lock had changed. Reading further within the demod status would show if the carrier was now locked or unlocked.

Unit Status Raw Output Alarms(11th value) - Integer Scalar Value 10

Bit 0 – Mod
Bit 1 – Demod
Bit 2 – Reference
Bit 3 – Clock
Bit 4 – Redundancy
Bit 5 – No Backup (valid only in Redundancy)
Bit 6 – Backup in Alarm (valid only in Redundancy)
Bit 7 – Over Temperature
Bit 8 – Mod Hardware Failure
Bit 9 – Demod Hardware Failure
Bit 10 – FEC A
Bit 11 – FEC B
Bit 12 – Optional Interface Failure
Bit 13 – No Send Data Activity
Bit 14 – No Receive Data Activity

Modulator Status Raw Output Alarms (2nd value) - Integer Scalar Value 1

Bit 0 – Carrier

Bit 1 – Data
Bit 2 – AUPC at Limit
Bit 3 – AUPC Fail
Bit 4 – Data Timing
Bit 5 – No Data Activity
Bit 6 – Clock
Bit 7 – Output Level
Bit 8 – LO Alarm
Bit 9 – Synth Step Alarm
Bit 10 – System Alarm
Bit 11 – Reference Alarm
Bit 12 – OCXO Alarm (L-Band only)
Bit 13 – FEC Alarm

Demodulator Status Raw Output Alarms (2nd value) - Integer Scalar Value 1

Bit 0 – Eb/No
Bit 1 – AGC
Bit 2 – Data
Bit 3 – Level
Bit 4 – LO Alarm
Bit 5 – Synth Step Alarm
Bit 6 – System Alarm
Bit 7 – Reference Alarm
Bit 8 – Backward Alarm
Bit 9 – Buffer Slip

Interface Status Raw Output Alarms (2nd value) – Integer Scalar Value 1

Bit 0 – Interface
Bit 1 – Optional Interface Failure
Bit 2 – Optional Interface in Reset

The third standard variable returned to the raw status commands shows the current test modes in process.

Unit Status Raw Output Tests (12th value) - Integer Scalar Value 11

Bit 0 – Unit Test Active
Bit 1 – Mod Test Active
Bit 2 – Demod Test Active
Bit 3 – Interface Test Active

Modulator Status Raw Output Tests (3rd value) - Integer Scalar Value 2

Bit 0 – Pure Carrier Test Mode
Bit 1 – Alternate 1/0 Carrier Test Mode
Bit 2 – SideBand Carrier Test Mode

Demodulator Status Raw Output Tests (3rd value) - Integer Scalar Value 2

Bit 0 – IF Loop-Back Enabled
Bit 1 – Fixed Carrier Mode

Interface Status Raw Output Tests (3rd value) – Integer Scalar Value 2

Bit 0 – Terrestrial Loop-Back Enabled
Bit 1 – Satellite Loop-Back Enabled
Bit 2 – Modulator Pattern Generator Enabled (BER)
Bit 3 – Demodulator Pattern Generator Enabled (BER)
Bit 4 – Demodulator Pattern Generator Locked (BER)

1.1.4 Command Aliasing

It is possible using the Bash/Ash shell facilities to make commonly used commands easier. For example the commands "**m500ctl -r xx ?**" is very common for many cases where a query is required, or the command "**m500ctl -r xx value**" is the normal form for setting a single parameter. The functional aliasing of those commands plus one for a verbose query can be achieved by entering the following three lines into a bash/ash shell:

```
mq() { m500ctl -v 0 "$@" ? ;}

mr() { m500ctl -r 0 "$@" ? ;}

ms() { m500ctl -r 0 "$@" ;}
```

These create three "alias functions" in the current session that allow entering abbreviated m500ctl command invocations. Rather than having to type these in later versions of the root filesystem contain a small script that does this by simply typing "**. mset**". This script is called automatically upon loging into the SnIP from the console or a Telnet or SSH session and does not normally have to be manually done except for special cases. If this facility is present in your current filesystem (those after 0.6.19) then you will see a line presented at login saying that "Modem control abbreviated commands are set...". After any method the shortcut 'mq' then queries verbosely using just the mnemonic - '**mq xx**', mr queries with raw output using '**mr xx**', and 'ms' sets a parameter using just the mnemonic and value - '**ms xx value**'. So for example to query and set the demod data rate to 2 Mbps the commands would be

```
mq ddr

ms ddr 2000000
```

These function aliases are lost on entering a separately spawned shell. So in a script you would need to to issue the command "**. mset**" again as part of the script. Note that this command is a period followed by a space and then "mset". In Linux this "sources" the lines in the script.

1.1.5 A Unique m49ctl and m500ctl Case - Redundancy

There are no special connections required if the SnIP is used for control in a system where it is not also used as the Data Interface except for one case - redundancy. Occasionally one may want to use the SnIP for control where the synchronous serial interface is used in a redundant mode. A modem with a SnIP installed will normally not allow enabling redundancy because the same serial port is used both for redundancy communications between the modems and for SnIP internal communications. If this is the desired connection then two methods are possible to resolve the contention:

- In older modem software versions prior to 1.07 a special internal cable between the SnIP and modem must be installed that removes both the serial port and the SnIP identification lines. The modem then does not know that a SnIP is installed and it therefore is not an available interface option. The SnIP to host modem communication must now be connected via an external RS-485 cable.
- All later modem software versions above 1.12 use a different method. Redundancy communications is transferred to the normal Remote Control port at J6, requiring an "Alternate Redundancy Cable" part number DRF08-084. All internal cabling remains the same. This method also permits redundancy between modems when the SnIP is used as the primary data interface. For more information on techniques by this method please refer to the separate guide titled "***SnIP Redundancy Guide***".

1.1.5.1 Redundancy Control via "m49/m500ctl"

The m49ctl and m500ctl applications include redundancy control parameters, but are only usable in this special case stated above. There are several redundancy control limitations, which are stated in the main manual, and repeated here for reference.

- One unit of a redundant pair can send its full configuration to the other unit in the pair. But, the “Send Configuration” command, “urc 1” can only be executed from the currently On-Line unit. The configuration will be sent even if the Off-Line unit is in alarm.
- The command to switch to the redundant unit, “uss 1”, is essentially switch away from the currently On-Line unit to the currently Off-Line unit. It can only be issued to the unit that is currently On-Line. There is no effect to telling the Off-Line unit to switch. The switch will also fail if the currently Off-Line unit is in alarm.

Other items that are not obvious is that the Unit Redundancy packet/command, “ur ?”, does not contain the information of which unit is currently On-Line. That information comes from the Unit Status request, “us ?”. Likewise the “urs” command does not control which unit is on-line, instead it is a control for what results in a switch request, whether any alarm, Alarm A, Alarm B, etc. The On/Off-Line status of a unit and the request to switch from a currently On-Line unit to its backup is done from the Unit Status. Thus to switch, the command is “m500ctl 1 uss 1”. The “-v 1 uss ?” verbose query returns not only the On/Off-Line status, but also the backup status if the unit is currently On-Line and its own status as ready to go On-Line if currently Off-Line.

Unit address setting discipline is a great help in redundancy control. In order to know the address of both units in a redundant pair it is strongly suggested that adjacent odd-even address pairs be used. For example one pair would use the addresses 1 and 2, and the next pair would use 3 and 4. If Telnet control is being used the same type of numbering scheme might be used with the IP address used to access the SnIP cards.

The M5 and M500 modem’s built in redundancy operates on a non-priority basis. That means that there is no preference for which unit is On-Line if both are out of alarm and operating normally.

A proper sequence for bringing up a redundant pair might be as follows:

1. Set the Unit Address of the first unit of a pair to an odd number, e.g. 1. Then set the other unit’s address to the next even number, e.g. 2.
2. Set each unit into 1:1 redundancy mode. “m500ctl 1 ure 1” and “m500ctl 2 ure 1”.
3. Set the unit parameters on the first unit for normal operation and insure that all alarms are off. If the alternate is in alarm, this should place this unit On-Line.
4. Insure that the first unit with the correct configuration settings is On-Line, if necessary, go to the other unit and force it offline with the “uss 1” command.
5. Send the first unit’s configuration to the backup with the command “m500ctl 1 urc 1”.
6. The backup unit should now go out of alarm, and be ready for redundancy service.

1.1.6 Other Control Manipulation

Since the SnIP is running Linux, this implementation has multiple other possibilities to manipulate anything controllable from the command line, including the m49 and m500ctl programs. These are the bash script and Perl. Both contain powerful processes for executing commands and using the output. Indeed, the web server uses Perl to issue m500ctl commands. It then takes the responses in raw mode and places them into variable arrays. Those arrays are handed off to JavaScript arrays when a web page is created, allowing display of current values.

No attempt is made here to show how this is done. However refer to Section 2.0 below for a script set used to manipulate modem profiles. You can also view all of the cgi, Perl and Javascript contents used to perform these actions using a text editor.

1.1.7 More Efficient Monitor and Control Methods

A new set of software for the M500 series of modems released in 2010 beginning with Version 1.19 includes a special “Unit Polling Status” binary packet, command # 0x0A. This command is designed to make monitor and control easier and more efficient. SnIP m500ctl programs at version 0.36 and above coupled with the filesystem 0.6.09 and above can use that software to advantage.

The modem remote binary packets and front panel menus reflect a division of control into 4 functional areas: Unit, Modulator, Demodulator and Interface. These may be logical for human control but are not ideal for remote monitor and control. From that perspective modem data might best be divided into 3 categories:

1. **Fixed Data** - Should never change except on power-up. (e.g. Unit Type or Serial Num.)
2. **Configurable Data** - Is changed on command, never by itself (e.g. Mod Data Rate)
3. **Status Notification** - Can change autonomously like Eb/No, Alarms, Tests, etc.

The purpose of this new Unit Polling Status packet is to allow an M&C system like the SnIP itself, or an SNMP agent or client to efficiently manage data and not constantly resend huge volumes of never changing data in order to transfer simple information needed. The structure of the new 0x0A packet allows that. It contains all of the "common" required current status information plus notification of other items which have changed in a fairly concise manner. See the table below for its contents.

The SnIP polling routines and the m500ctl use this one packet to create an internal demand driven database of modem data. This makes most m500ctl functions 2 to 15 times faster, and allows low latency status information to create an efficient AJAX based web that displays status information live (within a few seconds delay)

How it works in the m500ctl program is fairly simple, and similar procedures can be used in an external M&C system connected to the modem's remote control port. The SnIP's background modem polling routine periodically (every 1 to 5 seconds) requests the 0x0A packet data and stores that information in a temporary file. Then as a request for data from a particular packet arrives a decision is made based on **1)** local availability of that packet data plus **2)** the polling status information indicating if that data has changed since the last read (is fresh). If either is not satisfied then the packet with the desired data is read from the modem and the results stored for future use. If a future request can be satisfied from the local data copy then no request is made to the modem. When a successful change is made to category 2 items the local database is updated.

This system is called “demand driven” because no periodic polling is done of all the data. Instead data is stored locally only when a request demands that data. Only the Polling Status data is periodically requested. The poll timing can also be demand driven, where repeated requests cause the polling interval to decrease.

The Polling Status packet contains virtually all data in category 3 above and are designated in the Polling Status Packet structure shown below as “volatile” for bytes 17 through 29. Items in category 3 are masked within the modem from the Command Change flags in Bytes 10 to 16 of the Polling Status packet shown below. Thus those change flags indicate that other items in packet status have changed. This keeps you from re-reading a packet for changed data you already have. If however configurable data in category 2 is changed by someone else that bit will be set indicating that the packet needs to be requested again. There is one category 3 parameter that also falls in category 2 and requires special attention. It is the Modulator Transmit Carrier Level which can be changed under program control but also changed automatically when the AUPC is enabled.

1.1.7.1 Latched Status and “m49/m500ctl”

The Change flags in Bytes 0 to 3 of each packet and the Status flags are “latched”, that is they hold a change in status until read. Therefore changes are not lost if they return to their previous state between reads. The read interval does not determine if an event is lost, just the granularity of its detection. The Unit Polling Status command has a slightly different set of change flags because of its purpose. The Byte 0-3 change flags are reset upon reading this packet, as in all other packets, but the command change flags in Bytes 10-16 are cleared by reading the particular

packet that the flag refers to. A separate set of individual command change flags are maintained by the modem for each interface, the internal one used by the SnIP, the rear panel remote control port and the MCC channel control port.

The Polling Status packet also permits easier implementation of an “event recorder” where significant events are time stamped and stored for later review.

M500 Series Modem Remote Unit Polling Status Command 0x0A									
Byte	***Bit 0***	***Bit 1***	***Bit 2***	***Bit 3***	***Bit 4***	***Bit 5***	***Bit 6***	***Bit 7***	***Notes***
Byte 0	UnitStat	UnitCfg	ModStat	ModCfg	DemStat	DemCfg	IntStat	IntCfg	This Cmd Change Flags. Cleared reading this cmd.
Byte 1									
Byte 2									
Byte 3									
Byte 4	Mod Cxr	Mod Mjr	Mod Mnr	Mod Test	Dem Lock	Dem Mjr	Dem Mnr	Dem Test	Frnt Panel LED Image
Byte 5	Unit Sum	Unit Local	Unit Remt						
Byte 6	OnLine	MCxrEn	DemLck	ModAlm	DemAlm	RefAlm	OcxoAlm	ClkAlm	Modem Status and Alarms.
Byte 7	RdnAlm	NoBckAlm	BckUpAlm	TempAlm	MHardFail	DHrdFail	FecAFail	FecBFail	
Byte 8	IntfOptFail	SndDtaAct	RcvDtaAct	UnitTst	ModTst	DemTst	IntfTst	RdnEn	
Byte 9	BufStatus	BufSlip	BufSlipSign	BUCAlm	LNBAlm				
Byte 10	Cmd 00 **us**	Cmd 01 **uc**	Cmd 02 **uk**	Cmd 03 **ux**	Cmd 04 **uu**	Cmd 05 **uo**	Cmd 06 **ur**	Cmd 07 **um**	Command Change Flags. Cleared by reading the specific command.
Byte 11	Cmd 08 **ua**	Cmd 09 **ut**	Cmd 0A **up**						
Byte 12	Cmd 40 **ms**	Cmd 41 **mi**	Cmd 42 **md**	Cmd 43 **ma**	Cmd 44 **mt**	Cmd 45 **mm**	Cmd 46 **mb**		
Byte 13									
Byte 14	Cmd 80 **ds**	Cmd 81 **di**	Cmd 82 **dd**	Cmd 83 **da**	Cmd 84 **dt**	Cmd 85 **dm**	Cmd 86 **dl**		
Byte 15									
Byte 16	Cmd C0 **is**	Cmd C1 **io**	Cmd C2 **ia**	Cmd C3 **it**	Cmd C4 **ie**				
Byte 17	Unsigned char - Buffer Fill Status 0 - 200								Demod Volatile Data.
Bytes 18-19	short int - Receive Eb/No 0 to 200 in 0.1 dB								
Byte 20-23	int - Receive Carrier Offset								
Byte 24-25	short int - Receive Carrier Level								
Byte 26-27	short int - Estimated Receive BER								
Byte 28-29	short int - Transmit Carrier Level								Mod Volatile Data
Byte 30-31									Undefined

1.1.8 Customizing m500ctl

The m500ctl program normally uses a base set of text variables including the program name “m500ctl” itself and the model number “M500”. If for some reason you want to change this the program is built to allow some simple modifications. For example, if your unit is private labeled then it may use a different model number. One simple change will allow you to use a different command name and have the program print your unit's model number.

Decide what name you wish to use. We will use the term “***modem***” as an example new program name. Next create a symbolic link to the m500ctl program in the /usr/bin directory:

cd /usr/bin

ln -s m500ctl modem <-- Use your desired name here

That's it. To test the change try calling the program under the new name with one of the following options that would have printed the program name and model number:

modem -V

modem -c

modem

The program outputs will now not refer to m500ctl or M500 as the model number (unless that is the model number).

2.0 Modem Profiles – Saving, Setting and Manipulation

Two basic modem profile scripts are provided in SnIP file-system versions 0.5.40 and above. These are “**mp-save**” and “**mp-set**”. Their purpose is to allow saving and loading configuration profiles from and to modems. Modem profiles are stored in text files kept normally in the “/etc/config/profiles” directory. The files consist of comments, directives and modem commands. The modem commands are all legal m500ctl commands, so the commands in the file can be fed into a modem via “mp-set”. Profiles can be created manually or automatically via the “mp-save” script.

The **mp-save** script contains a default list of m500ctl query commands which are executed in order. It can also use an external command list file allowing saving of variable elements to be easily achieved. It only executes requests and the responses from the modem are formatted and saved into the named modem profile. Since a modem’s configuration can be easily saved into a profile and then loaded into the same or another modem, one of its main uses is to create basic configuration profiles.

The **mp-set** script reads a profile created by any means and issues the commands to the modem. The script can also contain “directives” used to control the control flow. The two existing directives allow setting whether command outputs are shown (verbose) or quiet, and a “wait” command that can delay execution for a specified number of seconds. Examples of a profile created by the mp-save script shown below. Note in this profile that the comments enclosed in standard C style (/* */) were added and not a valid part of the profile file format. Lines beginning with the hash character # are valid comments which the mp-set script ignores.

An example of creating a modem profile using the mp-save command script and then viewing the profile contents using the Linux “cat” command is shown below. Comments on the listing contents are shown in yellow highlighting. This particular unit is an L-Band and differs from an IF modem mainly by the transmit and receive frequencies.

Several lines of commented information is put into the profile at the beginning by the mp-save script to allow identification later.

```
[snip-L64:l /root]# mp-save my-config.mp
    /* Save default parameters to file my-config.mp */
Saving to /etc/config/profiles/my-config.mp...
Profile saved.

[snip-L64:l /root]# cat /etc/config/profiles/my-config.mp     /* This show the contents of profile created */
# M500 Modem Profile
# Saved Thu Feb  3 11:04:38 PST 2011
# <Unit_Status> "M500L" 13509 "1.19.040" "M505" 2 110 0 0 3 1 2 0 8 "n/a" </Unit_Status>
# <Config> 0 0 1 1 0 0 0 0 0 0 0 0 0 </Config>
0 mie 0          /* Always add this to make sure carrier is off before settings */
0 ucc 3          /* From here on is read from the modem and put in this file */
0 ucp 0
0 uos 0
0 uxp 2
0 uxt 0
0 uxr 5
0 uxa 1
0 ure 0
0 urs 0
0 urh 6
0 mif 6000000000
0 mil -125
0 mis 0
0 mifl 0
0 mipc 60,50,-350
0 mip 0
0 mdmfr 1,4,0,4,0   /* This is the modulation, FEC type, option, code rate and RS setting */
0 mdr 2000000
0 mds 1
0 mdk 0
0 mmz 0,6,3,0,5,2,0
0 mie 0             /* Current IF carrier enable status - will turn carrier on again if it was set */
0 dif 1700000000
0 dis 0
0 difl 0
0 ddmfr 1,4,0,4,0
0 ddr 4000000
0 dds 1
0 ddk 0
0 dmz 0,6,3,0,5,2,0,0
```

Modem profiles can also be used to run any series of commands or to program test procedures. For example a profile can be used to put a modem into loopback mode and run the internal BER Test set for a fixed number of seconds, returning to normal operation when the time runs out. An example profile which accomplished that function is shown here:

```
---------- run-local-ber.mp --------------------
# This profile sets the modem to a standard BERT to the
# satellite side for the specified number of seconds, and
# also sets the IF loopback.
0 dtl 1
0 itber 1,1,0
wait 30
verbose 1
0 is ?
verbose 0
# Now recover to normal
0 itber 0,0,0
0 dtl 0
 #done
------------------------------------------------------------
```

New directives will shortly be added to allow inline sending of modem response outputs to a file and showing the file contents.

2.1 mp-save Script

The “mp-save” script is used to save the configuration of the specified modem into a named file called a “modem profile” here. The configuration is saved as an ordered, sequential list of m500ctl commands. Informational comments are added to the file to allow identifying it later. The list of parameters are customizable.

A default list of common parameters is part of the program as is a sequence that includes common practices. For example the first command listed in the modem profile is one to turn the carrier off so that subsequent use by the mp-set command provides a measure of safety. The default list can be over-ridden by creating your own list of parameters in a text file and calling it as part of the command invocation.

The mp-save command script is invoked as follows:

mp-save [-p <parameterfile>] [-u <UnitAddress>] [/path/]filename

The required filename is the name of the modem profile. If the optional path is included, which must begin with the leading “/” character, then the file specified by the full path plus filename is used. If no path is supplied then the modem profile is saved in the “/etc/config/profiles” directory. A typical use of this would be to create a temporary profile in the /tmp directory which is RAM based instead of permanently stored in the flash. For example to save a profile named test1.mp in the /tmp directory you would use the command:

mp-save /tmp/test1.mp

The “**-p**” option loads an alternate parameter list from the file “parameterfile”.

The mp-save script current default list and sequence is:

Unit:
ucc ucp uos uxp uxt uxr uxa ure urs urh.
Modulator: (mie 0 added to file first)
mif mil mis mifl mipc mip mdmfr mdr mds mdk mmz mie

Demodulator:
dif dis difl ddmfr ddr dds ddk dmz
Interface:
iom.

The format of the alternate parameter file should follow that of the default list, which can be viewed by looking at the mp-save script itself. Basically that includes a list of commands separated by spaces. No parameters are used and the list must not contain any carriage returns or new-line characters. Formatting can be added to avoid a single long line by using the standard Linux line continuation character "****" immediately preceding a new line. An example might be:

```
mif -miif -mirf milo mil mis mifl mie mdmfr mdr mdk mbf mbe mbr \
dif -diif -dirf dilo dis difl ddmfr ddr ddk dlf dle dlr \
iom iep iem
```

Notice the "–" sign before some of the parameter acronyms in the listing above. The – causes that value to be saved as a commented line, which would prevent its use if the resulting profile was used with mp-set.

The "**-u**" option allows saving the profile of a modem connected to the external RS-485 bus. The unit must be attached and powered, and have a remote control interface set to binary command mode using RS-485. The Unit Address must also be set to a unique value between 1 and 253.

2.2 mp-set Script

The "mp-set" script is used to execute the commands within a modem profile. Commands within the file consist of of a sequential list of m500ctl commands plus a set of execution directives which control other actions as shown in section 2.2.1 below.

The mp-set command script is invoked as follows:

mp-set [-v] | [-m] | [-f <ipaddress>] [/path/]filename

The optional switches v, m and f are used to set the starting condition of the verbose, mirror and far directives respectively which are described in section 2.2.1 below. Normally they would be used to apply a directive to an entire profile, making it multipurpose. For example a profile created for one modem can be applied to another by using the –f switch.

The "**-f**" option requires that the procedures and SSH DSA keys are in place to allow SSH remote execution as described in the separate document titled "***SnIP Remote Execution Guide***"

The required filename is the name of the modem profile. If the optional path is included which must begin with the leading "/" character, then the file specified by the full path plus filename is used. If no path is supplied then the "/etc/config/profiles" directory is searched for the filename and executed. For example to execute a profile named test1.mp in the "/tmp" directory you would use the command:

mp-set /tmp/test1.mp

2.2.1 mp-set Directives

Directives are commands which can be put into modem profiles to perform other actions during the execution of the modem m500ctl commands. Most directives are followed by parameters which control their operation. For example some use a 0 or 1 parameter to enable or disable the function, or a file name to send output to. Currently the directives available are

- **wait** nnn – Uses the Linux sleep command to stop executions for the specified number of seconds nnn, after which it resumes.
- **verbose** 0 or 1– When verbose is set to 1 a verbose output of the commands following this one is provided until verbose is set to 0 (the default).
- **mirror** 0 or 1– performs a link inverse of the following commands when enabled. Continues until disabled by setting the parameter to 0.
- **far** 0 or IP Address– Sends commands to the following commands to the named ip address for remote execution. Assumes that the named unit has been previously set up with SSH keys for operation. Continues until redirected to the local modem by setting the parameter to 0 or directed to another modem using a new IP address.
- **write** filename– Creates or overwrites the named file.
- **append** filename – adds the following contents to the named file.
- **show** filename – similar to the Linux cat command, used to display the contents of a file.
- **print** "quoted text" – Like Bash echo this prints the following text in quotation marks.

3.0 Remote Modem Configuration via the SnIP

Beginning with modem software version 1.22 two major changes were made in the modem control software. The first was the addition of the Unit Global or "bulk" modem configuration command. This command allows setting most operating parameters of a modem with a single string of values. The second change was that the SnIP access to the remote modem for set commands was unlocked. Prior to this version remote end modem parameters could be read but not set.

These changes open significant new possibilities in modem control without the necessity of other communications channels. They also open the possibility of dangerous mistakes that could interfere with other traffic on a satellite transponder, or cause loss of remote modem control making it necessary to send an operator to that location to correct the errors.

The purpose of this section is to describe safe and reliable methods for remote modem control.

3.1 Safe Remote Modem Configuration Procedure

Consider the equipment setup as shown in Figure 2 above. Let us assume that you are an operator at the hub site with several modems connected to several remotes as depicted. A logical sequence necessary to program any changes in the remote modem that affect link connectivity might be as follows:

1. Insure that the link is locked in both directions.
2. Insure that the multiplexers are enabled in both directions and the MCC channel is available.
3. Insure that the transmit power in both directions will be sufficient to allow demodulator lock with the new settings.
4. Store the current configuration of the remote end modem in an available configuration "slot".
5. Set a "Restore Time" on the remote configuration saved above. If demod lock is not achieved within the set time, the remote will return to the old (current known working) configuration, allowing recovery.
6. Store the current configuration of the local end modem in an available configuration "slot".

7. Set a “Restore Time” on the local configuration saved above. If demod lock is not achieved within the set time, the local will return to the old (current known working) configuration, re-locking with the remote's old configuration and recovering the link.
8. Program the remote modem with the desired parameters using the m500ctl ug string. This will cause the link to lose lock.
9. Program the local modem with the desired complementary parameters using the m500ctl ug string. This should cause the link to regain lock.
10. Wait an appropriate time and check that the link is indeed locked in both directions.
11. Remove the “Restore Time” setting on the remote end for its old configuration.
12. Remove the “Restore Time” setting on the local end for its old configuration.

This type of sequence should probably be accomplished under program control, such as a Bash script, to insure proper timing and execution. An example of such a simplistic script is given in the attached addendum.

We know that any configuration change is going to disrupt traffic, but a scripted method like this will be significantly faster that doing the changes manually. It may be appropriate to reduce the time even more by increasing the MCC channel overhead before making the change and then reducing it after. The change time is the modem processing time, plus the link transfer time for each remote command set, so a significant percent decrease in time could be achieved using this method.

3.2 Modem Configuration Aids

The ug control string can be lengthy especially if significant changes are being made. It is very difficult to enter the control string manually, so several techniques are made available to ease this task. Foremost is that the m500ctl ug string can be saved to and read from a file, using the type operations shown is section 1.1.2.3.1 above. This also includes developing application profiles where all the configuration settings are stored in all modems for this application. Later they can be customized by just setting those parameters that differ between them, such as perhaps just the IF frequencies and for on demand services, the data rates.

3.2.1 Using the HTML Page to Generate the UG Configuration String

We also provide an html web page that can be opened on any computer, Windows, Linux or Mac that has a web browser. This page can also be downloaded from our web site and will operate stand alone on any computer with a web browser. The page presents a graphical display of modem parameters that can be set as desired and then the necessary configuration string can be generated in a text box. The string can then be copied and pasted for use with the m500ctl program or put into locally saved text files for later modem configuration. Normal client page protections prevent an html page from actually inputing to and running a program on your computer and therefore it cannot directly configure a modem.

3.2.2 Using the Sample mlink Script to Automate Configuration Changes

We also can provide an example of a simple Bash script that will make changes to a pair of modems together as a link operation. We plan on later expanding this into a full script for flexible link programming from link “profiles”.

4.0 Web Server Monitor and Control

The SnIP also runs a web server by default. The web server can be used to control the SnIP itself, the SnIP Interface functions including bridging and routing and the host modem plus others connected to the external RS-485 port.

The web interface is accessed at the IP Address that has been set for the modem. So to show the SnIP’s web page for a modem/SnIP whose IP address is set to 192.168.1.131 you would open your favorite browser in either Windows or Linux and type the IP Address into the browser’s

URL address window. A SnIP at that Address will respond by requesting the user name and password. By default use “root” and “datum” for these two which should then bring up the main SnIP web page.

Selecting the “Modem” tab will show the main modem informational page and present a new set of sub-menu items under the tabs which allow navigating to the various modem control pages that are available. The amount of information the modem can present is too large to show on a single page and is therefore divided into functional areas.

5.0 SnIP SNMP Monitor and Control

The SnIP contains the “Net-SNMP” suite of programs which are not enabled by default. They can be enabled or disabled either from the command line or from the SnIP>>SNMP page of the web server.

The SNMP server in the SnIP consists of both the standard Agent daemon and the coupled modem sub-agent which are enabled via a single control. There is also a configuration file in the /etc/snmp/ directory that allows persistent settings that are loaded on each boot of the SnIP.

Operation and use of SNMP is described in detail in the separate document titled “***SnIP SNMP Guide***” available on our website. Some basic configuration information is given below for reference.

5.1 SNMP Configuration

SnIP SNMP configuration consists of setting the correct variables for your system in the SNMP configuration files and enabling the SNMP daemon (server) operations from the configwiz program.

5.1.1 SNMP Configuration Basics

When enabled the SNMP daemons wait and listen for messages on any interface port containing messages with the SNMP protocol. If the messages are addressed to this units IP Address and are properly formed the agent will act upon the message and return a response.

5.1.2 Enable or Disable SNMP via configwiz

From a terminal session (console or telnet or ssh) you can control whether SNMP or SNMP Trap daemons will start on SnIP bootup. The configwiz options (1) selects the SNMP master agent and the modem sub-agent. Save the changes and on the next reboot the selected agents will start automatically.

5.1.3 Enable or Disable SNMP Manually

From a terminal session (console or telnet or ssh) you can control start or stop the SNMP Master agent daemon. These commands are not persistent and a reboot will not return to the same state unless it is saved via the configwiz method above.

/etc/init.d/snmpd start

Or to stop

/etc/init.d/snmpd stop

5.2 SnIP Modem SNMP MIBs

SnIP SNMP MIBs are located on our website at www.datumsystems.com. Since these are relatively new they are subject to change and should be checked frequently to insure that changes in modem capabilities and software are reflected in your version of the MIBs.

The MIBs are also included within each SnIP's file-system for easy access. They are located in the “/usr/share/snmp/mibs” directory.

End of Document.