DATUM SYSTEMS

**M7 Modem Technical Note**

# The “mec” M7 Modem Ethernet Control Program

**Revision History**

| | | |
|---|---|---|
| Rev 0.1 | 8-6-2013 | Initial Release. |
| Rev 0.2 | 8-12-2013 | Add Point to Multipoint setup info. |
| Rev 0.22 | 9-3-2013 | Add additional information. |

## Overview

It is convenient to be able to easily control modem parameters from an external computer rather than directly at the front panel. The external M7 modem control mechanisms consist of 1) an Ethernet web interface, or 2) serial or Ethernet binary packet control. This document briefly describes a Python program for control of M7 modems via a LAN Ethernet connection. The program is called “Modem Ethernet Control”, or mec for short. The mec program converts from relatively simple command line arguments to the M7 modem's binary packet format. Although not as “pretty” as the GUI web interface it provides a command line interface and the ability to automate control processes run by other programs. Note that there is a companion control program that can control modems via their RS-232 or RS-485 control port.

The mec program currently consists of a group of files intended as a demo of Modem Ethernet Control. It is a Python script program that requires that the Python application be installed on either your Windows or Linux computer. The same program will work on either and is installed in the same way on either.

## 1.0 Installation and Physical Setup--

The mec program requires the Python programming language for operation. If you are using Windows then you probably do not have Python installed, so start at Section 1.1. Linux computer users already have Python so you can start at Section 1.2. You do not have to know how to program in Python to use this program, just how to type commands on a console command line. You also need to carefully use spaces and capitalization as shown.

### *1.1 Python Installation --*

Virtually all Linux PCs include Python by default. If not it can be installed using rpm or apt-get as appropriate. Any version 2.5 or greater should work and the current one is 2.7.5. These programs have not been tested with the 3.x versions of Python.

For Windows you should go to the www.python.org web site and down-load the latest 2.x Windows msi installer which will be about 15.5 MBytes. After download simply run the installer and in a few minutes you will have a very modern, powerful and useful scripting language far beyond the capabilities of any scripting language your likely to find on Windows. Python runs typically at 1/2 to 1/10 the speed of C compiled programs too.

## *1.2 "mec" Program Installation --*

The mec files are combined in a zip file that only requires extracting to a convenient location in order to run. I, for example, have a folder in my home directory named Projects and under that M7 and under that pc-apps. So I extract the zip file into the pc-apps folder, but it could be almost anywhere. On a Linux computer it is also convenient to place personal programs in the /home/<user>/local/bin folder (does not always exist) or the main /usr/local/bin folder which generally is part of the executable path. The installation of Python creates a path to the main Python executable location so that it is not needed for running Python itself or programs placed in most locations.

The files consist of the following:

- **mec.py** - the main program that you will run.
- **crc16ds.py** - the Datum crc16 calculation module.
- **m7defs.py** - the M7 modem parameter definitions. This file is auto-generated by another Python program from the current modem def files.
- **maddr.py** - a default modem IP address file. This file will likely be added to in the future for other uses. You can edit the first ip address entry titled 'hub1' to provide a default modem control address so it does not have to be typed on the program line.

You will also notice later that different versions of the m7defs, crc16ds and maddr.py files will be created with a .pyc extension. These are compiled versions of the modules created by Python.

The mec program can also create files in the PC's temporary folder which is /tmp on Linux/BSD/Unitx or \temp on Windows. These disappear when the machine is rebooted. The main one is a configuration file for the modem named ***mdmconfig.py***, created the first time the mec program is run. Each time a new unit is designated the configuration for that unit is added to this file. If you look at the file it contains the cards available in the modem in a format known in Python as a lists and dictionaries. The m7defs contains many dictionaries, each of which is like a database table.

## *1.3 Hardware Connections and Modem Setup --*

The mec program communicates between your PC and the modem via the controller's Ethernet control port. So you must connect both your PC and the modem to your local LAN using Ethernet cables. The modem front panel is used to set the controller's Ethernet address

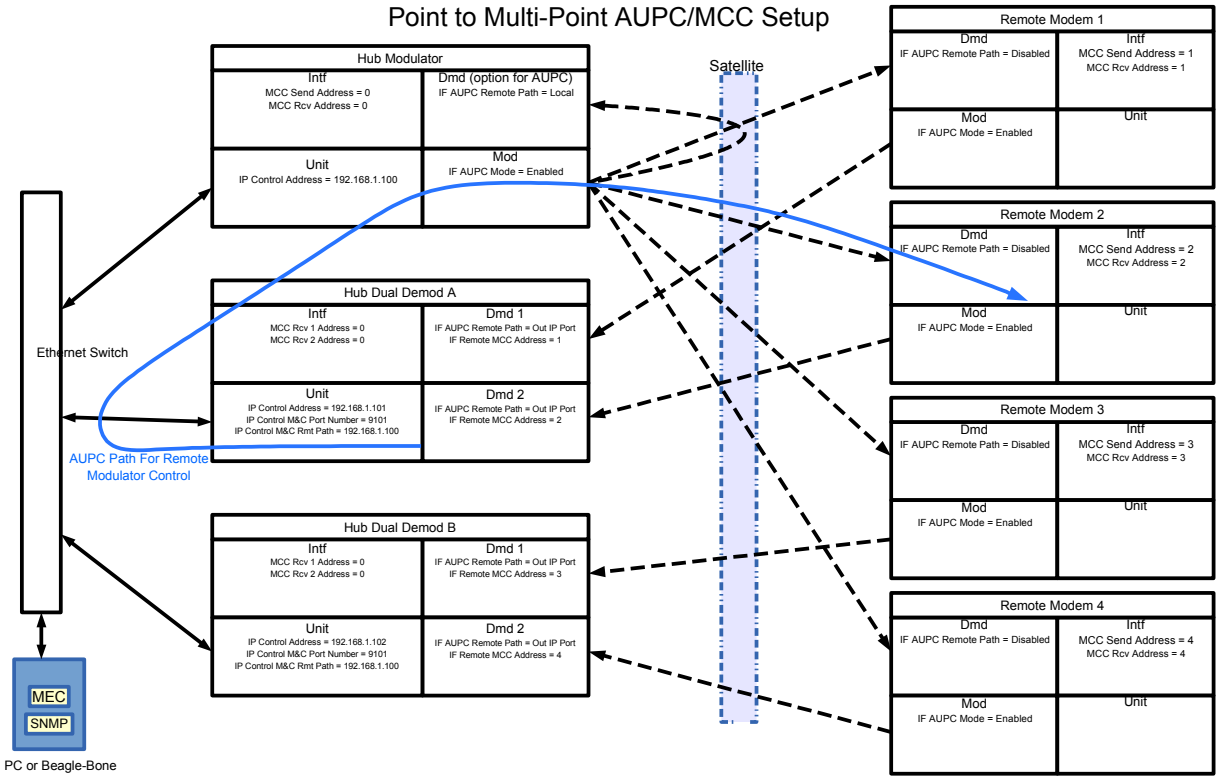
to one within the LAN mask.

You will also need to set the Interface MCC Rcv Address on remote units. In point-to-point links it can be any value 1 to 254 (the default is 1). In point-to-multipoint remotes the Interface MCC Rcv Address of each unit should be set to a unique number between 2 and 254, while leaving the hub transmit unit at a Rcv Address of 1. You also need to set each of the hub receive only unit's  Unit “M&C Remote Path” to the IP address of the transmit unit. All of the units should have the Interface MCC Mode to 2 and MCC Protocol to 0.

Examples of setup using these guidelines for a point-to-multipoint system is shown in the following table.

| **Parameter** | **Local or Hub Unit** | | | **Remotes** | |
|---|---|---|---|---|---|
| | **Transmit** | **Receive 1** | **Receive N** | **1** | **N** |
| Mod Frequency | OB Freq | N/A | N/A | IB Freq 1 | IB Freq N |
| Demod Frequency | N/A | IB Freq 1 | IB Freq N | OB Freq | OB Freq |
| Unit IP Control Address | IP Addr 1 | IP Addr 2 | IP Addr N | N/A | N/A |
| Unit IP Control M&C Remote Path | | IP Addr 1 | IP Addr 1 | N/A | N/A |
| Intf MCC Mode | 2 | 2 | 2 | 2 | 2 |
| Intf MCC Protocol | 0 | 0 | 0 | 0 | 0 |
| Intf MCC Rcv Address | 1 | 1 | 1 | 2 | N+1 |
| Intf MCC Send Address | 1 | ---- | ---- | 2 | N+1 |
| Intf MCC Send Rate Limit | 10.0 kbps | ---- | ---- | 10.0 kbps | 10.0 kbps |

This table shows the more complex point-to-multipoint case, where for simple point-to-point links only the MCC Mode and Protocol need to be checked and the MCC Rcv and Send Addresses can all remain at the default value of 1. A diagram of a point-to-multipoint system with values as above applied is shown below




# 2 Operation

A little preliminary inside information will help visualize the organization of the program operating commands. The M7 modem consists of 5 card slots connected by a backplane inside the chassis. One card is always the “Unit” controller representing the modem itself. There are then two “RF” slots which can contain transmit modulators or receive demodulators A standard modem would have one of each in either IF or L-Band versions. The last two slots are for two interface cards and normally only one is required to supply the data line interface. However two of the same or different interfaces may be installed either for optional use or one used for transmit and the other for receive.

In the mec operation we are always referring to one or more of these cards on each invocation. The standard names are unit, tx, rx, and io, and it is good practice to end each with a “:” colon to distinguish it. When there is more than one of any type card the first instance is referred to as xx or xx1, while the second is always xx2. So for example a dual demodulator unit will have rx: or rx1: for the first and rx2: for the second. Interfaces can be io, io1 or io2. Designating a card as the target for following parameters is essential because some parameter names may be repeated and thus not unique.

## *2.1 Running the "mec" Program --*

To invoke the program start a command shell and cd (change directory) to the folder containing the program -- where you extracted the zip. For example:

```
cd \Projects\M7\pc-apps\m7-mec
```

Then a first test run which will show all of the available “unit” controller mnemonic commands and their current values. Note: Linux will require you use “./” before “mec.py” if mec.py was not placed on a folder on the user's PATH environment variable.

Lets jump right in a ask for just a little information about the modem configuration.

```
mec.py -v -u 192.168.2.57 -v unit: id0 id1 id2 id3 id4
```

And here is what this particular unit (located elsewhere on the internet) returned

```
mike@asusqd:~$ mec.py -u 172.244.91.166 -v unit: id0 id1 id2 id3 id4

Modem Unit Card in Slot: 0
   Status Controller ID = 'M709001A-1r3.01'
   Status RF 1 ID = 'M709004A-1r3.01'
   Status RF 2 ID = 'M709004A-1r3.01'
   Status Interface 1 ID = 'M709017A-1r9.91'
   Status Interface 2 ID = 'M709016A-1r0'
```

We will explain a little bit more later about what exactly is being specified on this line, but this should return with the value of the parameters id0 through id4 from the unit controller card set with the M&C IP control port address of 172.244.91.166. If you were another program looking for just raw data you would send the same command but remove the “-v” option to get back raw data.

Now lets get a lot of information all at once. The -a or –all option tells the program to go get all of the available modem parameters for the following card name.

```
mec.py -v -u 192.168.2.57 -a unit
```

replace the 192.168... address with that of your local modem that was set above. This will get and display about 150 parameters for the main M7 “Unit” card. If you edit the maddr.py file as indicated above then you do not need the -u <address> on the invocation line. Using -u over-rides the contents of that file.

To see more about how to enter commands simply type:

```
mec.py -h
```

It also helps to stretch the command prompt window down so that it is long enough for easy reading. Note in all of the following examples that its easier to use the up and down arrow keys to recall the command line history and simply edit the one you want. In Linux the history is maintained even between boots, while Windows only retains some during the current command line session.

The entries on the command line after the program name are divided into options, directives and arguments, following Unix/Linux conventions.

Options are single letter designators preceded by a dash “-” and they are used to control program options, like formatting, where input comes from and ultimate control destination. Usually there are also more descriptive option names and those use the format “--option-name=” with two dashes and an equal sign only if the option requires an argument itself. For example you can get verbose output using -v or --verbose  and set a remote using -r 2 or –remote=2.

Arguments represent the commands that will be part of the communications with the modem, mainly 3 letter mnemonics and optional values, plus the name of the modem card or function you are targeting. The cards represent physical functions in the modem like modulators, demodulators and interfaces.

The next sections shows most of this by examples. You can always refer to the programs help function by just typing mec.py or mec.py -h.

## *2.2 Getting and Setting One or More Parameters --*

The -a <card> or --all=<card> command shown above is both a test and a way to show all of the available commands for each card in the modem. The cards are generally unit, tx, rx and io. It is intended that if tx, rx or io have a number attached to the end, like rx2 then the second instance of that card is selected. I don't have one of those so its not tested. There is also nothing fixed about the card names so you can use various card names like for example the modulator card has the following names defined:

'tx', 'tx:', 'mod:', 'md', 'xmt', 'xmt:', 'transmit', 'transmit:'

Note that they most have a version with a colon appended. That is to allow commands to be easily read where the colon, if used, designates the card while parameter names use no colon. Use of the colon is also desirable in case a 3 letter parameter mnemonic happens to coincide with one of the card designators.

You must specify the card and one or more parameters for that card when invoking the program. An example might be to read the transmit and receive frequency and data rate, which could be done as here:

```
mec.py -v tx: frq bit rx: frq bit
```

and the response should show you this information formatted verbose, which is what the -v option specifies. Remove the -v to see what the raw output would look like, or replace it with a -x for xml. Note that this and following examples assume that you have edited the maddr.py file to include the unit IP Address and therefore do not need to use the “-u” option.

### 2.2.1 Setting Parameter Values

You can set one or more of the values on a single invocation by placing a new value after the 3 letter mnemonic for the parameter, for example:

```
mec.py -v tx: frq 61000125 bit rx: frq 82725000 bit
```

which will set the transmit and receive frequencies but only read the transmit and receive data bit rates. The mnemonics are the lower case version of the 3 letter codes in the M7 definition files. All values are entered as integers with no decimals, which means that for many value entries you may need to know the increments. Mnemonics followed by a number are considered a set command while mnemonics without a following number are considered a get operation with the exception in the next paragraph.

Because the mec program detects that a parameter value has been entered by it being a number, the few cases where a string is set as a parameter requires the keyword "set" as in

```
mec.py unit: sid set 'Mikes 3.6.25, Test 27'
```

If any spaces are in the string then quotes are required, either single or double are usable.

### 2.2.2 Formating Output

As mentioned above one main purpose for command line programs of this type are to provide an automation method. The default output format of values are just numbers or strings, with carriage returns separating multiple values. For human consumptions you will normally use the -v (verbose) format option, but there are also options as shown here:

```
-v, --verbose  Print output in user friendly text format.
-q, --qvb      Print output verbose and add 3 letter command.
-e, --expand   Print output verbose with formatted numbers/options.
-x, --xml      Print output in xml format.
-c, --csv      Print output in csv format.
-s, --ssv      Print output in space separated format.
-l, --loadable Print output in re-loadable format.
```

and some of those can be combined, for example space separated xml output which might be used to send to a file which can later be easily parsed. To see some of these try replacing the “-v” in the following command with -q, -x, -xs, -c and -e

```
mec.py -v tx: frq bit rx: frq bit
```

Want to see what the transmitted and received binary packets are composed of? Try using the -d option to dump the packet contents.

## *2.3 Working with Remote Modems --*

You talk to local modems, those within your LAN address mask, by using their individual Ethernet IP address. It is possible to direct packets to modem(s) on the far end of a satellite link by addressing the local modem it is linked with plus the remote units “MCC Receive Address”.The mec.py program then puts the necessary entries into the packet structure causing the local modem to act as a relay, passing the packet on to the remote modem, and the response packet back to the mec.py program. This capability requires that an MCC overhead channel be available.

If the interface selected in the modems on both end of a link are either I7 or E7 then the modem is capable of setting up an HDLC based packet insertion and extraction without a multiplexer being present and turned on. This allows simulation of the multiplexer's MCC channel for monitor and control of a far end modem. Although I don't have one to try, remote linked modem control should also work with SS, HSSI and G.703 interfaces if the mux and MCC are enabled.

To specify that a mec generated packet be sent to a remote and linked modem with an MCC Rcv Address of X, then you would add to the mec.py invocation the -r X or --remote=x argument option. So lets say the remote modem has a MCC Rcv Address (mra) of 1 then you could see all of its interface parameters with:

```
mec.py -v -r 1 -a io:
```

and then change its MCC Rcv address to 2 with:

```
mec.py -vr1 io: mra 2
```

After which the remote will only respond with a -r2 option

Note: We changed option formats slightly above to demonstrate typical input styles. For single letter options the option argument can be placed immediately after the letter without the normal space. And multiple single letter options can be placed together sharing a single '-' as the -v and -r options shown in the second case above do.

## *2.4 Local and Remote Modem Identification --*

The designation of modems is different in this Datum Systems control architecture as compared to many other control architectures because of our unique use of the modem's MCC channel to pass control messages on to remote modems separated via the satellite link. This has the distinct advantage that Internet access is not required at remote modem

locations. If straight SNMP were used for example the only normal way to access any modem is via a unique IP address. The local modem is acting as a “gateway” to the remote modem. Note that this action however only traverses one link.

Any modem unit is identified by the IP address of the local unit it is linked to (via RF) and its “MCC Receive Address”. The modem at the IP address is given a MCC Receive address of '0' while remote modems have receive addresses of 1 to 254. So a complete address is the IP address plus the remoter receive address. If you were to use the command line to cat (Linux) or type (Windows) the contents of the temp directory mdmconfig.py file then you might see entries like:

```
config_67_0 = {0: 'unit', 1: 'modif', 2: 'demif', 3: 'intI7' 4: 'Empty'}
config_67_2 = {0: 'unit', 1: 'modif', 2: 'demif', 3: 'intI7' 4: 'Empty'}
```

The number after 'config' is the last octet of the local IP address and the next number is the remotely linked modem MCC Receive Address.

In a point-to-multipoint system there may be multiple remote addresses using one IP address as a gateway. That hub unit would be the one that acts as the outbound transmit unit, while the several demod only units would specify that transmit unit's IP address as the Unit “M&C Remote Path”.

## *2.5 Macros --*

The mec program is capable of using internally defined and external user defined macros. Currently these consist of a named list of commands that can be easily executed.

A macro consists of a named list of arguments that are added into the argument stream automatically. For example consider that you might repeatedly need to use the command:

```
mec.py -v rx: frq lst cst est mod spc fro
```

to see a variety of demodulator IF parameters to determine the health of the receive. Actually such an internal macro already exists with the name “dif” short for demod IF. It is defined internally in the m7defs.py file as:

```
'dif': ['rx:', 'frq', 'lst', 'cst', 'est', 'mod', 'spc', 'fro']
```

which is a Python dictionary entry with a key work of “dif”. You can call this internal macro up simply by using its name, even in-line with other commands.

```
mec.py -v io: hid dif
```

which will show the io hardware ID plus the Demodulator data parameters defined by the macro.

### 2.5.1 Internal Macros

You can see the names of the internally per-defined macro list by using the directive “macros”.

```
mec.py macros
```

To view the full contents or even edit/modify the internal list they are defined in the file “m7defs.py” near the bottom of the document. Editing may not be worthwhile in the long run because program updates will likely overwrite this file.

### 2.5.2 External User Defined Macros

You can create your own macros in an external file and then call it up by using the option “-i” short for input, and then the name of the file. Macro files are searched for by default in Linux in the /usr/local/etc/mmc/ folder which will probably have to be created. For Windows the default folder has not yet been defined. Alternately the macro files can be placed in any folder and the full path given with the name when calling the program.

The contents of an external macro file consists of a simple list or arguments separated by spaces, exactly like the normal command line format. The dif macro as an external macro named draco would consist of a text file named draco with the contents:

```
rx: frq lst cst est mod spc fro
```

Internal macros are executed in-line where placed, while external macros are added to the end of the command line.

END