Module Interface Board - MIB

Service Port

Interface Control Document

Version 1.2.0

Table of Contents

1 Introduction 1
2 Connection 1
3 Data Organization 1
4 Interface Commands 2
Table 1. SP Command Qualifiers 2
4.1 Command Line Separators 2
5 Document Keys 3
5.1 Syntax Key 3
Table 2. SP Syntax Key 3
5.2 Terminology Key 3
Table 3. SP Terminology Key 3
6 Command Format to the MIB 3
6.1 Command selection criteria 3
6.2 Wildcard Operations 4
6.3 Get 5
6.4 Set 7
6.5 Attribute Lists 9
Table 4. Analog MP attributes list 9
Table 5. Digital MP attributes list 10
Table 6. Analog CP attributes list 11
Table 7. Digital CP attributes list 12
Table 8. Engineering Units 12
Table 9. Analog Monitor Conversion Types 13
Table 10 Device Types 13
7 Response Format from the MIB 14
8 Time-deferred SP Commands 15
8.1 Indicating a Time-Deferred Command 15
8.2 Time-Deferred Command Format 16
8.3 Restrictions on Commands 16
8.3.1 Number of set operations 16
8.3.2 Time must specify future time 16
8.4 Command Queuing 16
8.5 Command Execution 16
8.5.1 Command Execution Time Missed 17
8.6 Command History 17
8.7 Administration of the Queue 17
9 Service Port Details 18
9.1 Terminology 18
Table 11. Service Port Facility Terms 18
9.2 Service Port Facility Description 18
9.2.1 Input Handling 18

9.2.2 Output Handling 19
9.2.3 Queue Dump Function 20
9.3 Service Port Server Description 20
9.3.1 Service Port Server Implementation 20
9.3.2 Service Port Server Task 21
9.4 Execution Tasks 21
9.5 Shell Interface 22
9.6 Service Port Test Program 22
10 MIB Device Points 23
Table 12. Digital Control Points 23
Table 13. Analog Monitor Points 23
11 Appendix 25
Figure 1. MIB Software Diagram 25
Table 14. Flash Memory Layout 26

Revision History

Revision	***Date***	***Author(s)***	***Description of Changes***
1.0.0	Unknown	Elwood C. Downey	Original version.
1.1.2	October 8, 2003	Elwood C. Downey	Base revision in CVS.
1.2.0	June 30, 2004	C. Frank Helvey	Updated to reflect actual implementation of MIB framework and service port server. Implemented document revision number as document property "REVISIONLEVEL" which is accessed through the File->Property->Custom menu.
	August 9, 2004	C. Frank Helvey	Incorporate review comments from Bill Sahr.
	September 22, 2004	C. Frank Helvey	More review comments from Bill Sahr.

1 Introduction

This document describes the communication protocol over the Service Port (SP) on the Module Interface Board (MIB). The MIB is physically connected to the EVLA LAN on the one hand and to one or more pieces of electronic equipment on the other. The MIB functions as a uniform LAN interface for this disparate equipment.

The SP is used primarily by software processes to perform operational array activities. The same command syntax is available for engineering personnel via telnet from the operating system shell connection. Likewise, the same response syntax is used on the operating system shell as on the SP. Software client processes are discouraged from using the shell connection.

For completeness, it should be mentioned there exists another LAN connection on the MIB called the Data Port. It transmits all MPs with UDP datagrams on a periodic basis for any process wishing to know these values. It will also sends out alert data if a MP enters the alert state or transitions out of the alert state. MPs have attributes which affect the rate at which they are transmitted. Details of the Data Port are described in the MIB Data Port Interface Control Document.

See *Figure 1. MIB Software Diagram* for a schematic depiction of the overall MIB software.

2 Connection

The MIB presents UDP port 7000 for service port commands. All communications use ASCII lines. Each command fits within one UDP packet so no terminator is required.

The operating system shell is presented on TCP port 23 (the TELNET port).

3 Data Organization

The MIB presents the electronics equipment to the SP as one or more logical Devices, each of which has one or more Monitor Points (MP) or Control Points (CP). Each MP and CP has a set of Attributes including an Attribute that represents the value of the point. The other Attributes expand upon certain auxiliary issues related to that value, such as allowed minimum and maximum; each Attribute is defined to be a read/write or read-only field.

A logical Device need not correspond to a physical device. The association between logical Devices on the Service Port and physical devices at the hardware level is made by the module specific code and the MIB framework code, and depends upon the exact equipment connected to the MIB. The MIB itself will be one such logical Device for controlling and monitoring functionality wholly within the MIB.

Device, MP and CP names consist of alpha, digit and underscore (_) characters only. Case is not significant. MP and CP names taken together must be unique for one Device. Only the first 7 characters are significant for the Device names, and only the first 23 characters are significant for MP and CP names.

4 Interface Commands

The only two SP commands are `get` and `set`. `Get` requests information from the MIB's logical points (MP or CP) and may cause communications activity with the equipment associated with the Device. `Set` actively makes changes on the MIB and possibly to the equipment associated with the effected Device.

There are two command qualifiers that can be used for the `get` and the `set` commands. Those qualifiers are given in the following table.

Table 1. SP Command Qualifiers

Qualifier	Applies To	Meaning
`@`	`Set`	Introduces a time of execution for this command. If the time format (see below) and the `set` command's parameters are valid this will cause the rest of the `set` command to be processed at close to the time given. The exact time of execution is after the EVLA system timing pulse that occurs just before the time given. See section 8, "Time-deferred SP Commands" for more information.
`-v`	`Get` or `Set`	This is a "verbose" option to produce additional output for the command for which it is specified. For the `get` command this produces a "MIBstats" system statistical and current operating system state output. For the `set` command this produces a message indicating success or failure of the command; if the command is successful, the message will either be a count of matching Attributes found or if combined with the `@` qualifier, the success of the queuing of the time-deferred command and the command sequence number assigned to it. If the command fails then some type of informative message will be displayed.

After receipt of a valid `get` command the MIB will always return one response containing the requested information. A response from a valid `set` command is dependent on the presence of the `-v` option or on the severity of the error encountered in trying to queue a time-deferred command. Syntax errors in commands will generate always generate an error message.

4.1 Command Line Separators

Multiple whole commands may be packed into one packet separated by semicolon (;) or newline (\n) characters. The two character sequence backslash-n (e.g., "\n") is also considered a valid separator. The Shell connection can use the semicolon or multicharacter sequence but usually not the newline character.

A command line may be continued by using the backslash character followed by the newline (\n) character or a carriage return (\r) character. However, this does not allow the command to span more than one UDP packet or shell command buffer, so its usefulness is limited to making commands more readable to a user.

5 Document Keys

5.1 Syntax Key

All SP commands and responses are in a different font than the rest of the text in this document; the font used is the one in the following syntax overview.

Table 2. SP Syntax Key

Syntax	Meaning
`abc`	literal
`<abc>`	variable
`[abc]`	optional
`{a|b|c}`	one of a set

5.2 Terminology Key

The following terms are used in the document.

Table 3. SP Terminology Key

Term	Meaning
`device`	A name of a device connected to the MIB, including the MIB itself; generally there is a 1:1 correspondence between a device and a single piece of hardware. This can be up to 7 characters long.
`property`	A particular CP or MP on a device. This can be up to 23 characters long.
`attribute`	The name of an individual piece of data that is part of a MP or CP entry. This can be up to 23 characters long. Each MP and CP has an attribute called `type` that further qualifies its attribute list; the `type` attribute's value can be either **`analog`** or **`digital`**. The tables below summarize the attributes available for each type of MP and CP.
`value`	The value attribute of a CP or MP. This can be up to 47 characters long.

6 Command Format to the MIB

6.1 Command selection criteria

A `get` or `set` command selects the data item(s) it operates on by using a set of data called a triple. This is a set comprised of a device, property, and attribute name. These three components

are collected into a triple using a period (.) as the separator. From one to four triples may be present, each separated by one or more blanks. Each component may be specified as a name to indicate a particular instance or it can be the wildcard character (the asterisk (*) character) to indicate all instances. The device name is the only required name, and the attribute name can be omitted if the property name is present.

6.2 Wildcard Operations

The * character is used as a wildcard in the device, property, and/or attribute names, and for the `set` command as a value. The presence of the wildcard character in a device, property, or attribute position means that position can have any value; thus the rest of the command's information will determine the CP and/or MP that are matched by that command. When the wildcard character appears as the value it indicates that the CP or MP matched by the rest of the command should be set to its default value.

Wildcard operations involving selection criteria are considered to have succeeded if a single device.property.attribute is found that matches that operation's request and is valid for that operation.

For a `get`, because all attributes of all properties are readable, any matching device.property.attribute value will work, and the command's response will report all of the matches as appropriate.

For a `set`, not only does a particular device.property.attribute have to match the command's request, but the resulting match must also be writeable. If no writeable matches are found for the command then the "read-only" error message will be output if the `-v` option was also specified.

If the `-v` option was specified and the set command succeeds, the success message that is output will indicate the number of matches found for that command. This is done for sets with the `-v` option no matter if the wildcard character is present or not.

> WARNING
> Use of a wildcard in the selection criteria for a `set` of the `value` field of a point is allowed but, in general, should not be done, especially if it is the 'MIB' device whose values are to be modified - the MIB's 'Reboot' property will cause the device to reboot if set to any value via the Service Port.

If the asterisk character is used as the value to which an attribute is to be set, then the SP will access the default value for the device.property.attribute set and set the current value to that default. The MIB stores the default values of attributes in nonvolatile memory as an aspect of its configuration; if not present in the configuration the default value will be 0 (or the empty string for string-valued attributes).

> WARNING
> Note that using the asterisk in the value position of the command can be significantly slow.

6.3 Get

```
get [-v] {<device>|*}[.{<property>|*}[.{<attribute>|*}]]
```

The `get` command queries information about devices, properties, and their attributes. If the property component is absent the response only includes device information. If the attribute component is absent the `value` attribute is used.

See the Table 1. SP Command Qualifiers above for the options that the `get` command supports.

Here are some example get commands and their associated output; these commands are for a particular MIB software implementation, so another implementation could have different device and property (CP and/or MP) names. Note the timestamps are in MJD format and represent when the output was begun to be formed so it could be sent out to the client.

Get the name and other general information of all logical devices for this MIB: `get *`

```
<EVLAMessage location='Antenna 13' timestamp='53198.804700'>
      <device name='device1'>
      </device>
      <device name='device2'>
      </device>
      <device name='MIB'>
      </device>
</EVLAMessage>
```

Get the value of all properties (CPs and MPs) on all devices for this MIB: `get *.*`

```
<EVLAMessage location='Antenna 13' timestamp='53198.804811'>
      <device name='device1'>
            <monitor name='mx' type='analog' value='0' />
            <monitor name='my' type='digital' value='1' />
            <control name='cx' type='analog' value='12.123'/>
            <control name='cy' type='digital' value='0' />
      </device>
      <device name='device2'>
            <monitor name='mx' type='analog' value='7.9' />
            <monitor name='my' type='analog' value='0.4' />
            <control name='cx' type='analog' value='4.567'/>
            <control name='cz' type='digital' value='0' />
      </device>
      <device name='MIB'>
          <monitor name='MIBVERSION' type='analog' value='0.17' />
          <monitor name='MODULEVERSION' type='analog' value='0.11' />
          <monitor name='SYSMEM' type='analog' value='144368' />
          <monitor name='TELNET_S' type='analog' value='4384' />
          <monitor name='BugfixCount' type='analog' value='0' />
          <monitor name='HeartInterval' type='analog' value='0' />
          <monitor name='HeartTime' type='analog' value='0' />
          <monitor name='HeartReset' type='analog' value='0' />
          <monitor name='SeqMissCmds' type='analog' value='0' />
          <monitor name='codeLoader' type='analog' value='0' />
          <control name='xmlLoader' type='digital' value='0' />
          <control name='reboot' type='digital' value='0' />
          <control name='wantArchive' type='digital' value='1' />
          <control name='wantScreen' type='digital' value='1' />
          <control name='wantObserve' type='digital' value='0' />
```

```
      </device>
</EVLAMessage>
```

Get the values of all properties (CPs and MPs) on device1: `get device1.*`

```
<EVLAMessage location='Antenna 13' timestamp='53198.805387'>
      <device name='device1'>
            <monitor name='mx' type='analog' value='0' />
            <monitor name='my' type='digital' value='1' />
            <control name='cx' type='analog' value='12.123'/>
            <control name='cy' type='digital' value='0' />
      </device>
</EVLAMessage>
```

Get the values of all attributes for the mx property on device1: `get device1.mx.*`

```
<EVLAMessage location='Antenna 13' timestamp='53198.805395'>
      <device name='device1'>
            <monitor name='mx' type='analog' value='0' target='0'
            engr_unit='m' conv_type='NO_CONVERT' slope='1' intercept='0'
            max='100' min='0'  alert_arm='0' alert='0' a_period='600'
            s_period='50' o_period='50' aa_period='300' msg='' />
      </device>
</EVLAMessage>
```

Get the values of all the max attributes for all properties (CPs and MPs) on device1: `get device1.*.max`

```
<EVLAMessage location='Antenna 13' timestamp='53198.805403'>
      <device name='device1'>
            <monitor name='mx' type='analog' max='100' />
            <control name='cx' type='analog' max='15.68'/>
      </device>
</EVLAMessage>
```

Get the value of one property on device1: `get device1.mx`

```
<EVLAMessage location='Antenna 13' timestamp='53198.805413'>
      <device name='device1'>
            <monitor name='mx' type='analog' value='0' />
      </device>
</EVLAMessage>
```

Get the value of the same property on all devices that support it: `get *.my`

```
<EVLAMessage location='Antenna 13' timestamp='53198.805500'>
      <device name='device1'>
            <monitor name='my' type='digital' value='1' />
      </device>
      <device name='device2'>
            <monitor name='my' type='analog' value='0.4' />
      </device>
</EVLAMessage>
```

Get several different values and attributes in one command: `get device2.mx device2.mx.max device1.cx.min`

```
<EVLAMessage location='Antenna 13' timestamp='53198.805670'>
      <device name='device2'>
            <monitor name='mx' type='analog' value='7.9' />
      </device>
      <device name='device2'>
            <monitor name='mx' type='analog' max='240' />
      </device>
      <device name='device1'>
            <control name='cx' type='analog' min='0' />
      </device>
</EVLAMessage>
```

Produce an error trying to `get` a non-existent attribute: `get device1.mx.badattr`

```
<EVLAMessage status='err'>
  badattr: no such attribute
</EVLAMessage>
```

6.4 Set

```
set [@<time>] [-v] {<device>|*}.{<property>|*}[.{<attribute>|*}]={<value>|*}
```

The `set` command instructs the MIB to install new values for device properties and their attributes. These three components are collected into a triple using a period (". ") as the separator. A triple is assigned a value by following it with equals (=) then the value. Spaces are not allowed on either side of the equals. One to four triples and their assignments may be present, each separated by one or more blanks. If the attribute component is absent the `value` attribute is used.

See the section 6.2, "Wildcard Operations" for a discussion on the wildcard (asterisk, or *) character in a set command.

See the Table 1. SP Command Qualifiers above for the options that the `set` command supports. Some more information is given below as well for them.

Set several different values and attributes in one command: `set device2.my.max=40 device1.mx=5`

Confirm: `get device2.my.max device1.mx`

```
<EVLAMessage location='Antenna 13' timestamp='53198.808726'>
      <device name='device2'>
            <monitor name='my' type='analog' max='40' />
      </device>
      <device name='device1'>
            <monitor name='mx' type='analog' value='5' />
      </device>
</EVLAMessage>
```

`[@<time>]`

The moment when this `set` is to occur is given by the `<time>` value. If the time format is valid but specifies a time that has already past, then the command treatment depends on the MIB Framework software build; refer to section 8, "Time-deferred SP Commands" for more information. There are two formats allowed:

A) Modified ISO 8601 UTC time - YYYY-MM-DDTHH:MM:SS.mmm

Where:

YYYY - year, example "2004"

MM - month, 01 to 12

DD - day of month, 01 to the max day depending on leap year and month

T - time type, P for PM, or A for AM

HH - hour from 00 to 12

MM - minute from 00 to 59

SS - second from 00 to 59

mmm - millisecond from 000 to 999

B) Modified Julian Date (MJD) time - DDDDD.FFFFFFFF[FFFFFFFFF]

(MJD = JD - 2400000.5)

Where:

DDDDD - 5 digits of days since Nov 17, 1858 at 00:00:00.00

FFFFFFFF - fractional days, 8 digits gives millisecond resolution.

If the `@` symbol is specified but the <time> value is missing or invalid then the command will not be executed, and an error will be output even if the `-v` option was not specified.

Note that the ordering of the `@` and `-v` options are per the command syntax above; reversing them will generate a parsing error.

`[-v]`

This is the verbose switch as detailed in the qualifiers table.

6.5 Attribute Lists

Table 4. Analog MP attributes list

Attribute Name	Access Type	Description
name	Read	name of MP property.
type	Read	type of monitor point; analog in this case.
value	Read/write	current value, engineering units (default attribute). This is a floating point value.
target	Read/write	target value for a control loop if needed.
engr_unit	Read	units of value, target, max, min, intercept, and slope (see Table 8. Engineering Units).
conv_type	Read	conversion type to use to create value from raw data; slope and intercept could also be used depending on the conversion specified. See Table 9. Analog Monitor Conversion Types.
slope	Read/write	along with conv_type and intercept determines how raw units are converted to engineering units.
intercept	Read/write	along with conv_type and slope determines how raw units are converted to engineering units.
max	Read/write	maximum value in engineering units.
min	Read/write	minimum value in engineering units.
hi_alert_arm	Read/write	if 1 enables checking value against max; if value > max alert counter is incremented.
lo_alert_arm	Read/write	if 1 enables checking value against min; if value < min alert counter is incremented.
alert	Read	1 if an alert condition (either hi_alert or lo_alert) is active on this point.
hi_alert	Read	1 if a high alarm is currently being asserted, else 0.
lo_alert	Read	1 if a low alarm is currently being asserted, else 0.
a_period	Read/write	period between archive broadcasts on data port, in 100 ms units.
s_period	Read/write	period between screen broadcasts on data port, in 100 ms units.
o_period	Read/write	period between observing broadcasts on data port, in 100 ms units.
aa_period	Read/write	period between archive broadcasts on data port if the point is in alert state, in 100 ms units.
msg	Read/write	47 character string which could be anything the user wants.

Table 5. Digital MP attributes list

Attribute Name	Access Type	Description
name	Read	name of MP property.
type	Read	type of monitor point; digital in this case.
value	Read/write	current value (default attribute). This can be only 0 or 1.
alert_arm	Read/write	if 1 enables checking value against alert_on1; if value <> alert_on1 then an alert is declared (set the alert attribute to 1).
alert_on1	Read/write	this is the normal, non-alert value of the point. Deviation from this causes alert state to be entered immediately if the alert_arm value indicates alerting is desired.
alert	Read	1 if an alert condition is active on this point.
a_period	Read/write	period between archive broadcasts on data port, in 100 ms units.
s_period	Read/write	period between screen broadcasts on data port, in 100 ms units.
o_period	Read/write	Period between observing broadcasts on data port, in 100 ms units.
aa_period	Read/write	Period between archive broadcasts on data port if the point is in alert state, in 100 ms units.
msg	Read/write	47 character string which could be anything the user wants.

Table 6. Analog CP attributes list

Attribute Name	Access Type	Description
`name`	Read	name of CP property.
`type`	Read	type of control point; analog in this case.
`value`	Read/write	current value, engineering units (default attribute). This is a floating point value.
`dev_type`	Read	device IO type used for communication to the device that this point is for (see Table 10 Device Types).
`engr_unit`	Read	units of `value, step, max, min, intercept,` and `slope` (see Table 8. Engineering Units).
`slope`	Read/write	along with `conv_type` and `intercept` determines how raw units are converted to engineering units.
`intercept`	Read/write	along with `conv_type` and `slope` determines how raw units are converted to engineering units.
`p0`	Read/write	general purpose value for use by functions operating on this control point.
`p1`	Read/write	general purpose value for use by functions operating on this control point.
`p2`	Read/write	general purpose value for use by functions operating on this control point.
`p3`	Read/write	general purpose value for use by functions operating on this control point.
`p4`	Read/write	general purpose value for use by functions operating on this control point.
`p5`	Read/write	general purpose value for use by functions operating on this control point.
`p6`	Read/write	general purpose value for use by functions operating on this control point.
`p7`	Read/write	general purpose value for use by functions operating on this control point.
`min`	Read/write	minimum value in engineering units.
`max`	Read/write	maximum value in engineering units.
`step`	Read/write	for control loops, gives maximum engineering value to change output by in 1 cycle.
`a_period`	Read/write	period between archive broadcasts on data port, in 100 ms units.
`s_period`	Read/write	period between screen broadcasts on data port, in 100 ms units.
`o_period`	Read/write	Period between observing broadcasts on data port, in 100 ms units.
`aa_period`	Read/write	period between archive broadcasts on data port if the point is in alert state, in 100 ms units.
`msg`	Read/write	47 character string which could be anything the user wants.

Table 7. Digital CP attributes list

Attribute Name	Access Type	Description
`name`	Read	name of CP property.
`type`	Read	type of control point; digital in this case.
`value`	Read/write	current value (default attribute). This can be 0 or 1.
`dev_type`	Read	device IO type used for communication to the device that this point is for (see Table 10 Device Types).
`a_period`	Read/write	period between archive broadcasts on data port, in 100 ms units.
`s_period`	Read/write	period between screen broadcasts on data port, in 100 ms units.
`o_period`	Read/write	Period between observing broadcasts on data port, in 100 ms units.
`aa_period`	Read/write	Period between archive broadcasts on data port if the point is in alert state, in 100 ms units. (For future use)
`msg`	Read/write	47 character string which could be anything the user wants.

Table 8. Engineering Units

Name	Symbol	Meaning
UNKNOWN	(space)	Default, no engineering units.
VOLTS	V	Measure of electrical force.
MILLIVOLTS	mV	$1/1000^{th}$ of a Volt.
AMPS	A	Measure of electrical current.
MILLIAMPS	mA	$1/1000^{th}$ of an Amp.
OHMS	Ohm	Measure of electrical resistance.
FARADS	F	Measure of electrical capacitance.
PICOFARADS	PF	$1/1{,}000{,}000{,}000{,}000^{th}$ of a Farad.
JOULES	J	Measure of electrical energy.
WATTS	W	Measure of power.
MILLIWATTS	MW	$1/1000^{th}$ of a Watt.
HERTZ	Hz	Cycles per second.
MEGAHERTZ	MHz	Millions of Hertz.
GIGAHERTZ	GHz	Billions of Hertz.
GAUSS	Gauss	Measure of magnetic field strength.
CELSIUS	C	Measure of temperature, 0 degrees being the freezing point of pure water at sea level pressure.
KELVINS	K	Measure of temperature.
HUMIDITY	Humidity	Measure of evaporated water content of air.
PASCALS	Pascal	Atmospheric pressure.
METERS	m	Measure of length.
CENTIMETERS	cm	$1/100^{th}$ of a Meter.
MILLIMETERS	mm	$1/1000^{th}$ of a Meter.

Name	Symbol	Meaning
KPH	KPH	Measurement of speed.
DEGREES	degrees	Measurement of rotation
DEGREESSEC	degrees/sec	Rate of rotation
DEGREESMIN	degrees/min	Rate of rotation
ARCSECONDS	arc-sec	Measure of rotation
LITERS	l	Volume
NEWTONS	N	Force
MJD	day	Time
SECONDS	second	Time
MPH	MPH	Speed
BITFIELD	bitfield	Typeless field to be output in hexadecimal

Table 9. Analog Monitor Conversion Types

Conversion Name	Symbol	Meaning
NO_CONVERT	NO_CONVERT	This is the default transform from raw to Engineering units. A module's specific code could provide its own transformation method and use this to have the result reported to the MP.
LINEAR	LINEAR	Use `value =raw*slope+intercept`
POLYNOMIAL	POLYNOMIAL	Module-specific implementation.
SIGNED_LINEAR	SIGNED_LINEAR	Use `value =field*slope+intercept` where `field` is a signed 2's compliment number derived from a bitfield in the `raw` value.

Table 10 Device Types

Device name	Symbol	Description
No device associated	NULL_DEV	No device needed.
GPIO	GPIO	MIB parallel I/O port.
25LC040	EEPROM_25LC040	SPI EEPROM driver.
TLV2556	ADC_TLV2556	Driver for TLV2556 A/D converter.
MAX6629-MAX6632	TEMP_MAX66XX	Driver for MAX6629-MAX6632 family of temperature sensors.
TLV5624	DAC_TLV5624	Driver for Data Acquisition chip.
DAC716	DAC1_716	Driver for Burr-Brown 16 bit Data Acquisition chip #1.
DAC716	DAC2_716	Driver for Burr-Brown 16 bit Data Acquisition chip #2.
FPGA/PIC	FPGA_INTERFACE_1	P301/ALMA communications driver.
GPIO	ACU_GPIO	ACU/FRM-specific GPIO driver.
Power Supply Control	PSC_INTERFACE	Driver for PSC interface to D30X.

Device name	Symbol	Description
various SPI devices	T304_DNCVTR	T304 downconverter interface driver.
Data Acquisition Board, version 1 and 2	DAQ_INTERFACE	DAQx board protocol driver, used in M301.
AD9852 DDS and L301/L302 FPGA	DDS	Driver for the AD9852 DDS chip and FPGA used in the L301 and L302.
MAX186-MAX188	ADC_MAX18X	SPI driver for A/D converter chips.
Data Monitor Board	DMB_INTERFACE	Driver for Data Monitor Board, a.k.a. Analog Monitor Board.
generic I/O device 1 (generally, GPIO)	MODULE_IO_1 (module-specific name)	Generic driver whose name will be provided by the implementing module code.
generic I/O device 2 (generally, GPIO)	MODULE_IO_2 (module-specific name)	Generic driver whose name will be provided by the implementing module code.
generic I/O device 3 (generally, GPIO)	MODULE_IO_3 (module-specific name)	Generic driver whose name will be provided by the implementing module code.

7 Response Format from the MIB

All commands are immediately checked for validity before any action is taken. This checking does include syntax errors and unknown device, property or attribute names. Failure at this step results in no change in status to the MIB or any connected equipment modules. Syntax errors are always reported by both `get` and `set` commands, regardless of the absence of the `-v` option. Unknown device, property, and/or attribute names are always reported by the `get` command, and optionally by a `set` command if the `-v` option was used. Finally, a time-deferred `set` command will return an error message when no more time-deferred commands can be accepted, regardless of the presence or absence of the `-v` option.

All responses, if sent, are in XML format. The outer element is `EVLAMessage` with three optional attributes: `status`, `location` and `timestamp`. For a `set` command with the `-v` option that was successful the value of the `status` attribute value is `ok`. For a `get` command that was successful the `location` and `timestamp` attributes are present. For either command, if there was an error, the response will have a `status` value of `err` and a brief message about the problem will follow. Here are some examples of failure messages:

If the command is: `get device3^`

The response is:

```
<EVLAMessage status='err'>
  Illegal character: ^
</EVLAMessage>
```

If the command is: `get device3.*`

The response is:

```
<EVLAMessage status='err'>
  device3: no such device
</EVLAMessage>
```

If the command is: `set device3.*`

The response is:

```
<EVLAMessage status='err'>
  Missing property assignment
</EVLAMessage>
```

If the command is: `set -v device3.mx=1`

The response is:

```
<EVLAMessage status='err'>
  device3: no such device
</EVLAMessage>
```

NOTE: if the `-v` option was not present, this command would fail but no message would be output.

If the command is: `set device1.mx=1 device2.my=0 device1.my%=45`

The response is:

```
<EVLAMessage status='err'>
  Illegal character: %
</EVLAMessage>
```

Additional sub elements will be present within the `EVLAMessage` element for a successful `get` command, in order to report the queried values. See the discussion for the `get` command for examples of successful responses.

8 Time-deferred SP Commands

This section discusses in more detail the operation of the `set` command `@` qualifier.

8.1 Indicating a Time-Deferred Command

A `set` command (only) can have an optional time parameter specified after the `set` command verb. This time parameter is introduced by the `@` character and is assumed to be an absolute time in one of two formats. If a `@` character is not detected as the first non-white space character after the `set` command verb, then a time-deferred option cannot be specified for that `set`.

8.2 Time-Deferred Command Format

```
set @<time> [-v] {<device>|*}.{<property>|*}[.{<attribute>|*}]={<value>|*} ...
```

See the description of the `set` command in section 6.4 for more information on the set command format, and in particular the time formats supported for the time-deferred command qualifier.

8.3 Restrictions on Commands

8.3.1 Number of set operations

As per the standard service port command format, there can be no more than 4 triples (sets of device.property[.attribute]=value) specified for each set. Wildcards can be used and more than 4 resulting set operations can be performed as a result of that wildcard.

8.3.2 Time must specify future time

By default, the software will automatically execute a command that specifies a time that is past or too close to the future, rather than discarding or queuing that command. This behavior is selectable at compile time; the same symbol selects the sequencer task's handling of commands that have times < current time. If the default action is not selected the sequencer itself will discard a queued command if the time stamp it gets is already past the command's desired execution time

If the default action is not selected, then the time given for the command's execution must be at least 2 interrupt events (see below) duration in the future in order to be accepted. This will guarantee the command will be executed; anything less may result in the command getting discarded and is not allowed.

8.4 Command Queuing

All commands that are received and validated will be queued, with a maximum number of 50 allowed at any one time. Exceeding that number will generate an error message back to the initiator regardless of the state of the -v option flag. Likewise any parsing error will generate an error message to the initiator with an indication of the problem, if possible.

Commands are sorted in the queue in time-execution order. Commands queued to the same execution time are sorted in FIFO order.

8.5 Command Execution

The commands queued will be executed based on their specified execution time, and the EVLA system heartbeat event and heartbeat interval duration. If there is no heartbeat event coming into the MIB, the deferred command will be executed based on a simple elapsed time timer event and that timer's duration; in the following discussion both of these events (heartbeat and elapsed timer) are called "event".

A command will be executed if the system calculates that the command's execution time meets the following criteria:

Current event time <= execution time < (current event time + event interval duration)

The system supports execution of multiple commands based on one event; each command is

tested in order until the criterion is not met, at which point the sequencer will look for new commands and another event.

Actual execution of the command is done using the same routine employed by the service port's `set` command processing function.

Regardless of the presence of the `-v` option in the original command, no feedback for the executed command will be sent to any connected service port clients or the serial port.

8.5.1 Command Execution Time Missed

Base on a compile time symbol's value, the above execution condition can be modified to include or exclude commands whose time of execution is < the current time. If set to 0 (the default), the logic for discarding commands based on their time being < current time is disabled, and the command will be executed.

If the symbol's value is not 0, and if the next command in the queue is found to have an execution time < current event time, that command will be discarded and an error counter incremented. This error counter is called `SeqMissCmds` and is available in the MIB logical points list regardless of the state of the compile time symbol.

One possible cause of a missed execution time would be the MIB's time clock base value "leaping" forward in time such that at the previous event time the condition for execution was not met, but now at the current event time the clock has passed by that point in time.

8.6 Command History

All commands executed will be preserved in the free command entry list for as long as possible. Their sequence numbers will have their 31st bit set to indicate they were executed.

All commands that were discarded will be preserved in like manner, with their 30th bit set to indicate that they were discarded and not executed.

8.7 Administration of the Queue

Currently the only administrative function that exists for this MIB feature is a "seq_dump" command in the shell, which allows the user to dump out the sequencer task's queue of commands. Options available allow dumping All, Pending, Completed, Missed, or Complete&Missed entries in the queue. These options use the sequence number and the bit flags in it to determine the state of a command. This function does lock access to the queue while performing its action, so it does interrupt the normal processing of any pending commands in the queue.

The sequence number is designed to be used as a means to cancel outstanding queued commands before they execute, and the entire list could potentially be dumped via the service port's `get` function instead of using the "seq_dump" command.

9 Service Port Details

This section describes the internal workings of the service port facility of the MIB in more detail. This includes the UDP-based service port interface to the MIB as well as the shell interface to the MIB.

There is a set of common service port facility functions which support both the telnet interactive shell and the UDP-based service port server implementation. The service port interface code for each of the connection methods basically just has to handle the communications set-up and tear-down requirements for that connection method.

9.1 Terminology

Besides the terms used in the table 5.2, "Terminology Key", the following are used in this section of the document.

Table 11. Service Port Facility Terms

Term	Meaning
Command line buffer	This means a string of bytes that will be processed using the rules in the first parts of this document, looking for SP commands.
Command	This means a service port command as per section 6, "Command Format to the MIB".
Command data structure	This is a set of data items derived from the parsing and validation phases of the service port common routines. These data items will either describe a single "Command" or will contain an error generated from the parsing of a command or string that was supposed to be a command.
Command set	This is a set of 1 to N "Command data structures" all of which were produced from a single "Command line buffer". A command set is sorted in FIFO order.

9.2 Service Port Facility Description

This section describes the common service port command handling functions.

9.2.1 Input Handling

The service port input parser supports the parsing of the command line buffer received from the clients of the service port facility. All of the commands in the command line buffer are parsed into a set of intermediate command data structures, with pre-execution validation being performed to catch errors up front.

Any errors detected have their associated error messages written into the command data

structure. This was done because a UDP socket must have its data delivered as a complete message, not piecemeal like the TCP socket the shell uses.

All commands detected from a single command line buffer are queued into a command set in FIFO order so that when they are executed they will execute in the same order the user or program provided them.

If there should not be enough free command data structures to represent the detected commands from a command line buffer, then all of the commands in that buffer will be discarded and an error message returned to the caller of the command line parsing routine. That error may or may not be presented in turn to the user of the interface.

Once all commands have been detected and stored in the command set, the command set will be processed. See the section on Output Handling below for more information.

The delimiters used for commands are covered in section 4, "Interface Commands".

9.2.1.1 Command Limits

The software is currently limited to 50 simultaneously active, building, or pending execution commands. Attempts to exceed this number will generate an immediate error message back to the shell or UDP server client as appropriate, and all commands that have already been detected from that command line buffer will be discarded.

9.2.1.2 No Password Protection

There is currently no password protection on the service port for performing commands. Any user or program that can connect to the appropriate port on the MIB will be able to issue commands and receive responses. Adding a password requirement should be easy since both the shell and the service port server use the same command line parser function.

9.2.2 Output Handling

As stated previously in the "Input Handling" section the commands in the command line buffer are all parsed into command data structures before they are executed. Some may have error messages already written to them, if errors were detected during the parse or validation phase of the service port processing.

Once the command detection phase is over the next step depends on the task processing the command line buffer. For the shell, the command set which has been built will be executed immediately; for the service port server, the command set will be queued to an execution task selected from the pool of execution tasks (see below).

In either case, once the set of commands derived from the command line starts to be processed, a buffer of 32000 bytes worth of dynamic memory is allocated for the results of the command set. This buffer will then have each command's output written into it sequentially, in the order in which the commands were detected and queued into the command set. Once all of the commands have been processed, the resulting output buffer will be written out to the TCP (shell) or client UDP (server) socket.

Thus, the entire command set found on the command line can produce no more than 31999 bytes of output.

Because of the common code being used for the processing of the commands, the output format for the shell and the service port server is the same; the data will have carriage return and linefeed pairs at the end of each line.

9.2.2.1 No Output Length Checking

Currently, there is no output length checking in the service port facility software. However, during the validation phase of the command parsing it should be possible to do output size estimation before the commands are actually executed, and if needed break the resulting outputs into smaller portions and/or allocate an appropriately sized dynamic memory buffer for the results. Plans do exist to implement output length checking. (Note that UDP has a maximum output limit of approximately 65000 bytes for a single transmission.)

9.2.2.2 Statistics Output

The statistics output added to a `get` command's output when the `-v` option is given has a field for the total number of commands processed by the shell interface as well as the queue message processed counts for the service port server execution tasks.

9.2.3 Queue Dump Function

A shell function called "sp_dump" is available to display the contents of the service port command data structures queue. The queue has two components - a free list, and a pending list. Most of the time the pending list will appear empty since the commands are processed quickly by the execution tasks; the commands that are used will be queued to the free list at the end, preserving their information for as long as possible. Note, however, that the order in which they were submitted may not be the same as the ordering in the free list since the multiple execution tasks could release some of the command data structures faster than others.

The dump function also supports output of the commands processed and queue messages processed statistics for the service port facility.

9.3 Service Port Server Description

The service port server is comprised of 1 main task which receives data from a UDP port (port number 7000), and from 1 to N (N being 3 at the moment) "execution" tasks which actually perform the service port commands that are given.

9.3.1 Service Port Server Implementation

This section describes the service port server implementation using UDP sockets as the communication mechanism between client programs and the MIB.

9.3.2 Service Port Server Task

This task is blocked on the UDP known port waiting for data to arrive. It will accept up to 1 Ethernet MTU (1514 bytes) of input in one message; this is the command line buffer.

The task will immediately check the length of the received buffer against the known minimum service port command length (5 bytes currently). If the length is not within specifications an error message is generated back to the client and the task will go back waiting for more data to show up.

If the command line buffer is large enough the task will process it using the common service port facility command line parsing routine. See the "Input Handling" section above.

Once a command set has been generated, the entire command set will be queued to one of the service port execution tasks. The tasks are given command sets in a round-robin fashion, with preference given to a task which is not currently busy processing a command set.

9.4 Execution Tasks

These tasks are each blocked on their associated operating system queues waiting for a new command set to be sent to it. Each task has its own statistics counter, queued command counter, and queue. Each queue is capable of handling 10 command data sets.

Once a queue message is received by an execution task, it will increment a queue message counter and validate the message. If the message is not valid a serial port message will be generated and the count of waiting queued data items is decremented; then the task will return to waiting for the next queued message.

If the queue message received is valid, the task will increment a count of valid queue messages processed and attempt to acquire the output buffer (see the Output Handling section) for the output of the command set, blocking until it succeeds.

Once the buffer is allocated, the command set is given to the service port common command set execution routine. That routine will execute the commands in the set, writing their outputs into the output buffer sequentially until all commands have been processed. Then the entire command set is released to the free command data structures queue.

After the command execution routine is finished and returns the resulting output buffer, the execution task will send that buffer to the client using the UDP socket information from the queue message it received with the command set index. A new socket will be allocated for each such transmission.

Next, the output buffer is released and the number of queued data items waiting to be processed will be decremented. The task will then return to waiting for the next command set from its queue.

9.5 Shell Interface

The shell interface to the MIB is implemented as a function which is called whenever the main shell task detects a string of input beginning with either `set` or `get`. The function will further process the command line buffer given using the service port common command line processing function. Once the command line has been parsed into command data sets, the shell interface function will call the service port common command set execution function for all command sets detected.

Each command set will have its output written out before the next command set is processed.

There are no differences between the service port server's input and output and the shell interface's input and output, so programs or users could use either one. However, software client process that do not support a human interface are discouraged from using the shell connection.

9.6 Service Port Test Program

The SendMIBCmd test program on the Linux platform supports both the telnet shell interface and the service port server UDP interface to the MIB. This test program will read each line from a given text file and send it to the MIB given via the interface specified, waiting up to 10 seconds for a response before it reads the next line from the text file and repeats the process. Anything received from the MIB is output to stdout; if the timeout occurs instead a timeout message is output.

This program is written in C and should compile on different operating systems without too much modification.

The syntax of the command is:

```
sendmibcmd <MIBaddress>  {UDP|TCP} <filename>
```

An older version of the SendMIBCmd utility also exists on the Windows platforms; it is currently capable of handling the shell interface only and will output the same command every so many seconds to the MIB given on its parameter line.

10 MIB Device Points

The logical MIB Device supports the following CPs and MPs at the revision level of 0.17. Other CPs and MPs may be added in the future; generally none will be deleted. These are provided by the MIB Framework code separately from the MIB's configuration information and so are available on all MIBs.

Table 12. Digital Control Points

Name	Default Value	Description
`xmlLoader`	0	Controls XML file loading and shows status from a load. Setting this to 1 enables XML configuration file loading into the MIB. Once the XML file load completes, this has to be set to 1 again for another XML file load to happen. The `msg` attribute will have a textual description of the status of the XML loader; 20 seconds after a XML load completes, the message will revert to either "waiting to start" or "accepting socket".
`reboot`	0	If a set command is performed on this point a soft reset of the MIB will be performed; all current memory contents will be reloaded from flash.
`wantArchive`	1	If 1 then archive data will be sent for those points that are configured to do so.
`wantScreen`	1	If 1 then screen data will be sent for those points that are configured to do so.
`wantObserve`	0	If 1 then observe data will be sent for those points that are configured to do so.

Table 13. Analog Monitor Points

Name	Default Value	Description
`MIBVERSION`	0.17	This is the version number for the MIB Framework code. It is set by data inserted into the MIB software build. The `msg` attribute will have a text version of this, which includes the build date and time.
`MODULEVERSION`	?	This is the version number for the module's specific software. It is set by data inserted into the module's software build by the module's build file. The `msg` attribute will have a text version of this, which includes the version number and whatever else the module software desires up to the attribute's limits.
`SYSMEM`	0	This is set to the current available system memory in bytes when the `get` command is performed.

Name	Default Value	Description
TELNET_S	0	This is set to the current available stack of the TELNET server when the get command is performed.
BugfixCount	0	An internal MIB Framework debugging aide.
HeartInterval	0	This is the heartbeat event interval; the heartbeat is the EVLA system timing pulse and is not present on all module hardware. If this is 0 there is no heartbeat on this MIB.
HeartTime	0	The time in MJD of the last heartbeat event.
HeartReset	0	A count of the number of times the heartbeat rate calculation has reset due to errors.
SeqMissCmds	0	A count of the number of sequencer engine commands missed due to time having elapsed already; this is not used at present.
codeLoader	0	This will contain a status value from the MIB code loading program. The program is used to send new MIB code to a running MIB and burn it into the Flash on the MIB. The msg attribute will have a textual description of the status codes seen here. In particular a MIB should not be rebooted until the msg attribute reads either nothing or "Load Completed".

11 Appendix

MIB Software Design
Version 0.17
August 10, 2004
C. Frank Helvey

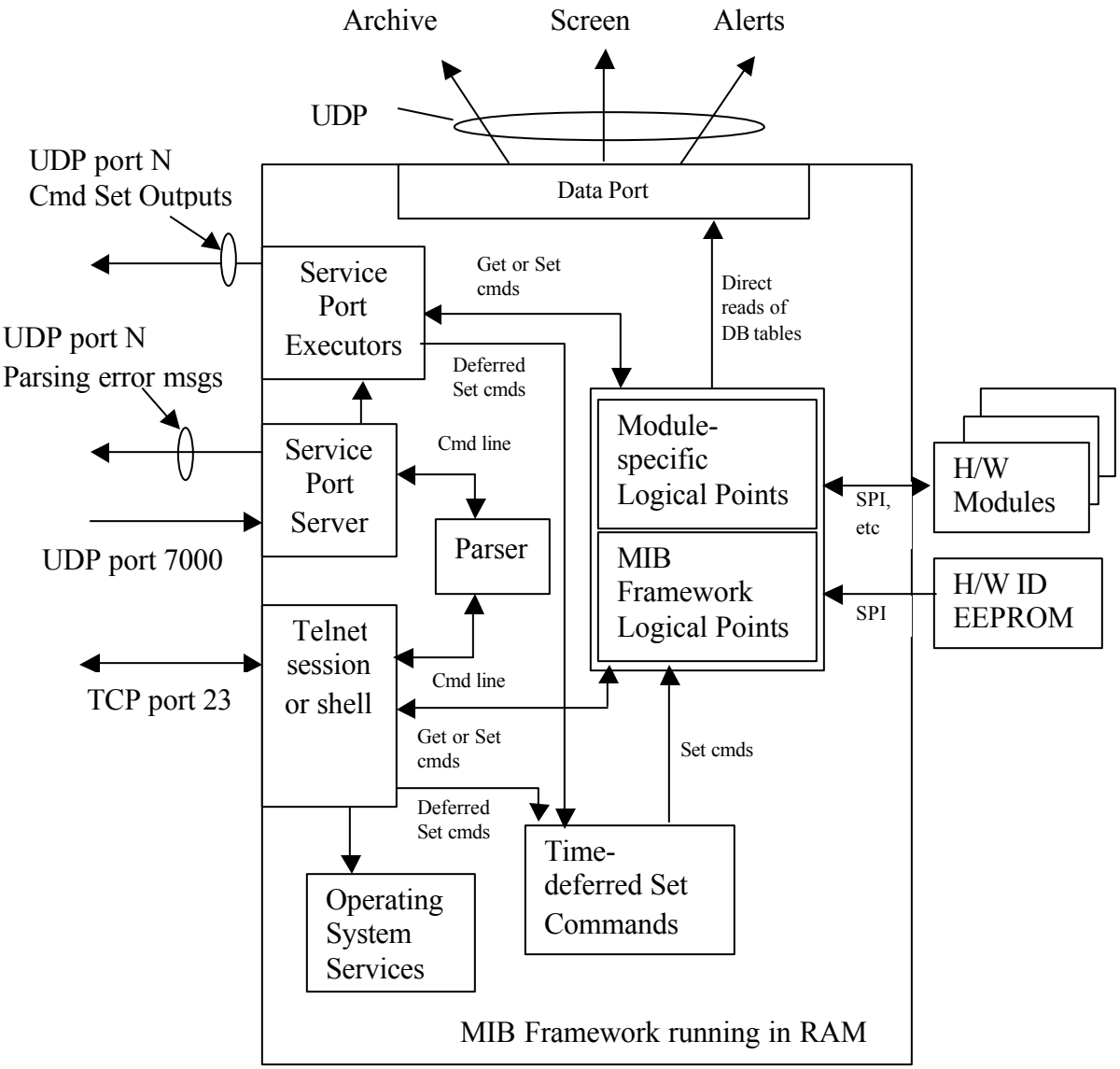

Figure 1. MIB Software Diagram

MIB Software Design
Flash Memory Layout
Revision 0.17
August 10, 2004
C. Frank Helvey

Table 14. Flash Memory Layout

Address Range	Description
0xA0000000 to 0xA0070000	Boot loader program (boot_bootloader 23-Apr-2004 or later)
0xA0080000 to 0xA00F0000	Current image, module software + MIB framework software.
0xA0100000 to 0xA0140000	Spare
0xA0150000 to 0xA0170000	Module specific configuration information in XML format
0xA0180000 to 0xA01F0000	Spare
0xA0200000 to 0xA020FFFF	MAC address