MIB BROADCAST STREAM SPECIFICATION

November 5, 2002, Version 1.0

This document contains a specification for the MIB broadcast stream. It will be specified in a language independent manner. It is intended that this data stream will be a mechanism for crossing the "language barrier" between system components written in different languages, for instance, C, C++, and Java.

The proposed method for distributing monitor data from the antennas is to publish or broadcast the data via UDP datagrams, with best effort delivery. The broadcasts will be done by software Devices residing in the MIBs. For the sake of convenience, this document may sometimes refer to broadcasts by the MIB.

The initial model for the communications portion of a software Device is that it will have two ports – a broadcast or data port and a service port. The broadcast port will be used for the dissemination of monitor data and the service port will be used both for the delivery of commands to a Device and for information exchange between a Device and clients. The information exchanged over the service port would be items such as a description of the control points and monitor points present on a device, a description of a particular control point, a particular monitor point, etc.

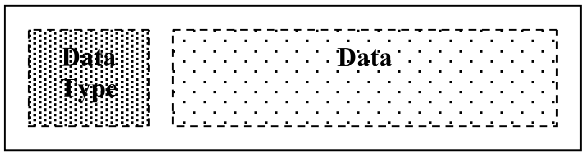
This specification may be used for both the broadcast port and the service port, or it may be used for the broadcast (data) port only. The possibility of using a standardized method of communication on the service port, such as SOAP, is being explored, but the investigations are not yet complete.

The MIB will broadcast monitor data, which will be received by software processes on the network. The MIB will take no cognizance of what these software processes do with the data or where they are located on the network. The MIB broadcast system will use the UDP datagram broadcast protocol to connect to level 3 in the ISO model. The protocol defined below is layered on top of UDP to define communication and data exchange at the higher levels in the ISO model. The content of the UDP datagram will be the Device Data Record (DDR) that is described at the end of this document. The DDR will be composed of the data elements whose descriptions precede the description of the DDR.

This document does not define the port number usage for the datagrams. That will be defined and specified elsewhere.

DATA ELEMENTS

The lowest level entity in the packet is the "data element". Each data element appears in the packet as a byte array that defines and contains the data for a supported data type. The supported data types include the primitive data types, *byte, short, integer, long, float, double, boolean, timestamp* and *string*, and the composite data types, *array* and *struct*. Primitive data types are irreducible, not composed of simpler types. Composite data types are constructed of elements each of which may be a primitive type or another composite type.



The Data Element

The data element has two parts or fields, a *type* field and *data* field. The type field is a single byte in length and defines the type of data. Its purpose is to provide information to the decoding software that allows it to extract the data from the data field. Each data type will be mapped to a numeric value. This value is what will appear in the data type field of the data element. The table below describes the mapping.

DATA TYPE	VALUE
BYTE	1
SHORT	2
INTEGER	3
LONG	4
FLOAT	5
DOUBLE	6
BOOLEAN	7
TIMESTAMP	8
STRING	9
ARRAY	10
STRUCT	11
MONITORPOINT	12
DEVICE	13

The data field contains the actual data. For a composite data type, such as an array, each element of the array will be a data element. Thus, the data field for an array is composed of nested data elements.

For all multi-byte data types, "big-endian" order is used, i.e. the leftmost bytes (those with lower addresses) are the most significant. To be tediously clear, an example is given:

The number 1025 stored as a 4-byte integer:

binary representation of 1025: 00000000 00000000 00000100 00000001

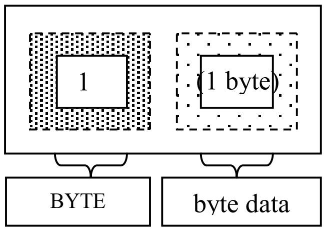
Address	Big-Endian	Little-Endian
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

Big-endian was chosen because it is the traditional network ordering. The TC11IB is little-endian so byte-swapping routines will be needed in the MIB.

All floating point representations will adhere to the IEEE 754-1985 standard.

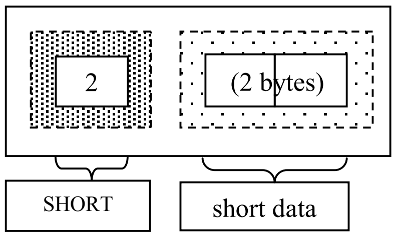
The supported data types are defined below.

BYTE:



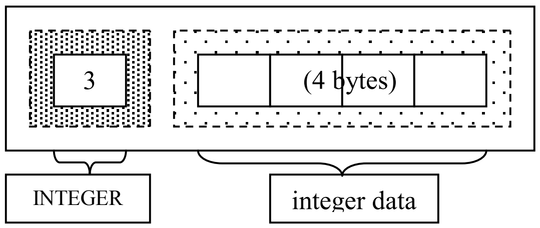
The contents of the data field may be interpreted as an eight bit, two's complement, signed number. Range: -2^7 to $+2^7-1$, or -128 to +127.

SHORT:



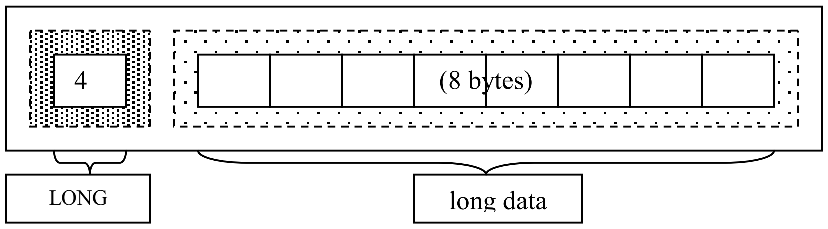
The contents of the data field are the bytes representing the bit pattern of a two's complement, signed, 16-bit integer in "big-endian " order. Range: -2^{15} to $+2^{15}-1$, or -32,768 to +32,767

INTEGER:



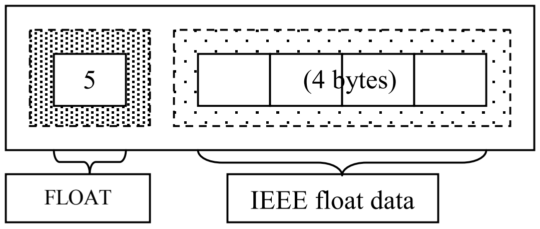
The contents of the data field are the bytes representing the bit pattern of a two's complement, signed, 32-bit integer, in "big-endian" order. Range: -2^{31} to $+2^{31}-1$, or -2,147,483,648 to +2,147,483,647

LONG:



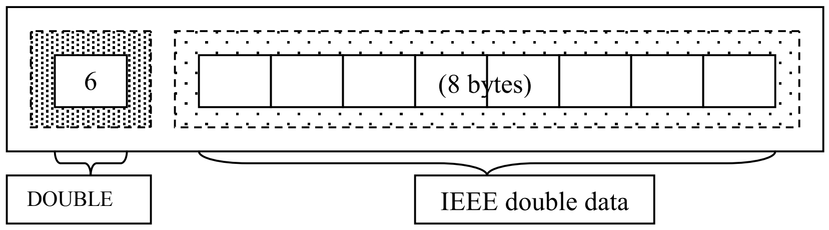
The contents of the data field are the bytes representing the bit pattern of a two's complement, signed, 64-bit integer in "big-endian " order. Range: -2^{63} to $+2^{63}-1$

FLOAT:



The contents of the data field are the bytes representing the bit pattern of an IEEE 754-1985 float in "big-endian" order. Range (normalized): +/- $(2-2^{-23})^{127}$, ~ +/- $10^{38.53}$

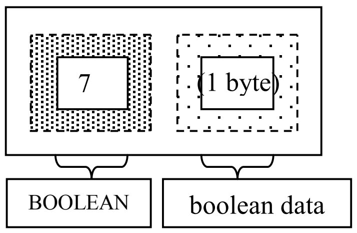
DOUBLE:



The contents of the data field are the bytes representing the bit pattern of an IEEE 754-1985 double in "big-endian" order. Range (normalized): +/- $(2\text{-}2^{-52})^{1023}$, ~ +/- $10^{308.25}$

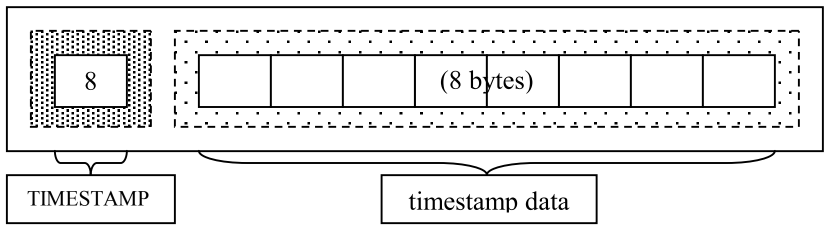
The precision of an IEEE 754-1985 floating point number is 24 bits or approximately 7 decimal digits. The precision of an IEEE 754-1985 double is 53 bits or approximately 16 decimal digits. For both cases, normalized numbers using the "hidden bit" are assumed.

BOOLEAN:



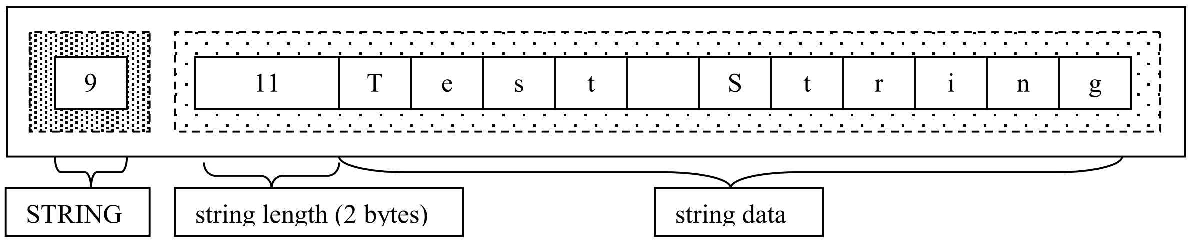
The data field will contain 1 (00000001) for TRUE or 0 (00000000) for FALSE.

TIMESTAMP:



A TIMESTAMP is physically identical to a DOUBLE. The big-endian, double precision floating point datum is the representation of the date and time as a Modified Julian Day (for example, September 27, 2002 at 0000h UT is MJD 52544.000).

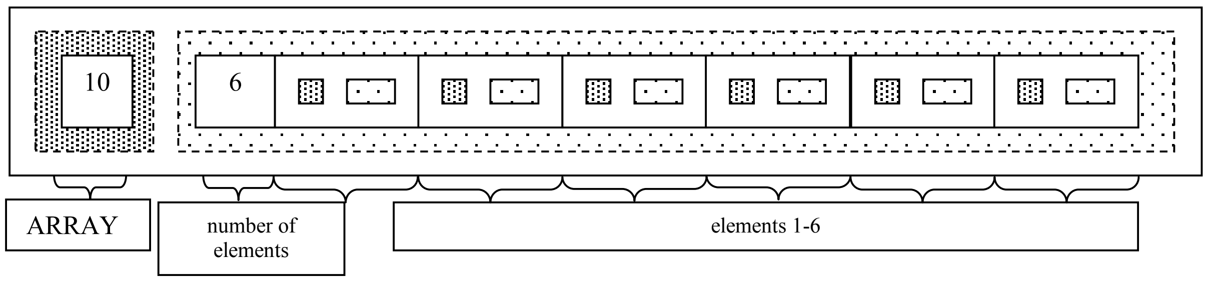
STRING:



The first two bytes of the data field contain the length of the string. Note that these bytes (in contradistinction to the SHORT data type) are to be interpreted as a 16 bit positive number (unsigned short), implying a maximum string length of 65535 characters. The remaining bytes of the data field contain the ASCII representation of the string (not Unicode). A string could have been treated as an array of bytes. It was felt that strings are sufficiently unique that they deserved their own type.

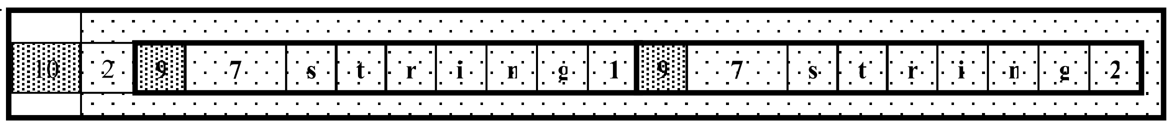
A STRING is not null terminated. It is noted and understood that this definition of a STRING does not conform to the definition of a string as used in the C programming language.

ARRAY:



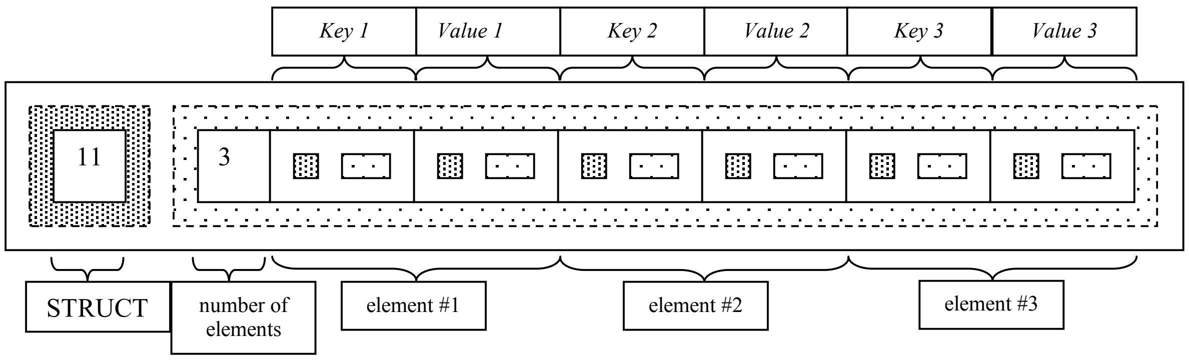
An array is composed of a data type field followed by a single byte, interpreted as an 8-bit unsigned integer (max value = 255), that gives the number of elements in the array, followed by the data elements. The data elements may be primitive or composite data types. Unlike conventional arrays, the MIB broadcast stream ARRAY is not limited to a set of objects of identical types. Different elements of the ARRAY type may hold different types of objects.

A simple example of an ARRAY data element that contains two strings, “string1” and “string2” is as follows:



In the example, each of the small boxes represents a byte of data. The boldfaced outlines represent a data element and the shaded blocks within the data elements represent the data type field. Looking at the first byte of data we see that the outermost element is an ARRAY data type (i.e., data type field = 10). The data field of the outermost element includes all remaining bytes. The second byte from the left – the first byte of the outermost data element - indicates that the ARRAY has 2 elements; these are outlined in boldface. The first byte of each of the two inner data elements indicates that each data element is of type string (i.e., data type field = 9). The second and third byte of each element indicates that each string is 7 bytes in length. The remaining bytes within each data element contain the actual string data. Note that the length of this data element is only determined by summing the lengths of each component element. In theory this can get very complicated, but probably is not a problem in practice.

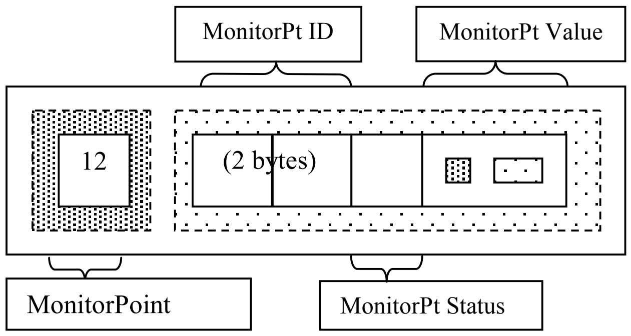
STRUCT:



A struct is a special case of an ARRAY, in which the sub-elements are paired. As with the ARRAY, the first element is a data type field followed by a single byte, interpreted as an 8-bit unsigned integer (max value = 255), that gives the number of elements in the array, followed by the data elements. For the data elements, the first element of the pair must be a string, which will be interpreted as a name associated with the second element. The MIB Broadcast Stream STRUCT is somewhat like a Python associative array or a Java Hashtable.

MONITORPOINT:

A monitor point is represented within the data packet as a monitor point header, followed by one or more data type elements. It is a special type of ARRAY as defined above.



The MonitorPt ID is a short (two byte) integer. As with the STRING type, these two bytes are to be interpreted as a 16-bit unsigned positive integer, with a maximum value of 65535. The ID provides a unique identification of the monitor point within the device. Its use is to provide a key to a more lengthy description of the monitor point that the Device in the MIB will send by another mechanism. MonitorPt Ids will never be reused, making old data always interpretable.

The question of whether the MonitorPt ID is assigned at programming time, compile time, or initialization time is as yet undecided. Therefore, programs handling this broadcast data must take care that the monitor point description originates from the same Device object as the broadcast data.

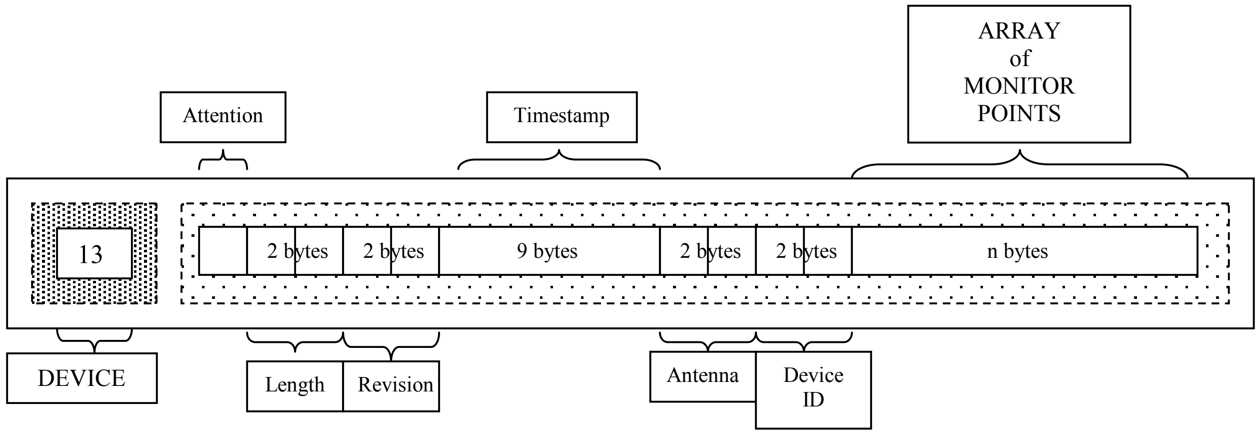
The MonitorPt Status is a byte. It will be interpreted as a bit pattern, not as a signed integer. A value of zero indicates that there are no warnings associated with this MonitorPoint. Meanings for other values are yet to be defined.

The MonitorPt Value is one or more of the data types defined above. The type and quantity of the data portion of the MonitorPoint element is defined by the particular MonitorPoint. The descriptions of the data values are provided by another mechanism.

DEVICE DATA RECORD:

A UDP datagram will contain exactly one device data record (DDR). Each DDR will contain only monitor points that have been sampled at the same time and at the same rate. Additionally, a DDR will contain only 1 value for each monitor point in the record. For example, if a given device contains 5 monitor points that are each sampled at a 1 second interval and 3 monitor points that are sampled at a 5 second interval, then any one DDR for that device is prohibited from intermixing values for the 1 second and 5 second interval monitor points, and the values it does contain will have all been sampled at the same second or 5 second tick (within some epsilon). These conventions eliminate possible ambiguities w.r.t. the meaning of the timestamp.

We require that each DDR be complete within itself – that is, neither a MONITORPOINT nor a DDR shall be divided between datagram packets. We also require that the DDR must fit within a single MTU payload. In order to guarantee that it fits within a single MTU payload, allowing ample room for possible network system options or even an intervening protocol, we shall here require that the total size of a DDR be less than or equal to 1280.



The first item (one byte) of each packet will contain the constant identifier indicating that this is a Device Data Record.

The second item (also one byte) is the "Attention" field. This byte is to be interpreted as a bit field, not as a signed integer. The meanings currently defined for this byte are:

0	Intended for screen use – archiver and observing layer should ignore this message.
1	For the archiver, data should be stored in the monitor archive.
2	For the observing layer. The archiver and screen clients should ignore this record.

The third item is a 16-bit unsigned integer (max value = 65535) giving the length, in bytes, of the data record. This item is redundant, in the sense that this number can be extracted from the datagram itself; it is included here to facilitate putting this packet in temporary or permanent storage without having to add additional data.

The fourth item is a 16-bit field giving a revision identifier for the software comprising the Device. The exact format and interpretation of this field has not yet been defined. The mechanism for generating this number has not yet been specified. However, we here require that all Devices of the same type with the same software revision number shall provide the same monitor point descriptions.

The fifth item is a TIMESTAMP (as defined above; occupying 9 bytes total), being the time that the Device generated the DDR. There is an assumption in this statement concerning when the DDR is generated. It is currently assumed that the DDR will be generated immediately after the monitor point values have been obtained. The timestamp in the DDR, under this assumption, is taken to be, within some reasonable epsilon, the time at which the monitor point values in the DDR were obtained from the device.

The sixth item is a 16-bit unsigned integer (max value = 65535) giving the ID of the antenna in which the MIB resides.

The seventh item is a 16-bit unsigned integer (max value = 65535) giving the ID of the Device. The programmer will assign Device IDs, and will assure that they are unique within an antenna.

The above items are then followed by a data item that will be an ARRAY of MONITORPOINT records. The number of elements in the ARRAY, which is a part of the specification of the ARRAY data type, will give the number of monitor points in the device data record.

Total Power

At this time, total power measurements are not yet adequately defined to specify a format. However, a convincing case can be made that the MIB Broadcast Stream Specification is sufficiently rich and flexible to deal with total power measurements. One possible format would be to return the total power values as an ARRAY with the first element of the ARRAY giving the time interval between the values in the subsequent elements, and the timestamp in the Device Data Record understood to apply to the last value in the sequence. Another possibility is to return total power as an ARRAY with alternate elements giving a time tag and a value.