Register Transfer Protocol

A Much Better Idea : Logical : Unique

AmbiLogique

Electronic Controllers

Revisions

1.0	2009-03-05	RJW	Existing documents gathered into this one.
2.0	2011-07-08	RJW	Provision for Checkword added.
2.1	2012-05-02	RJW	New Company name & logo.

Introduction

The AmbiLogic Register Transfer Protocol (ARTP) is used to communicate between PLCs, PLC modules and host computers. It can be used on either a point-to-point connection such as RS-232 or multipoint master-slave connections such as RS-485.

This specification is defines the way in which data is encoded and does not attempt to define the physical medium.

Because the protocol is based on standard asynchronous serial data transmission, any UART-type circuit can be used as the communications port to a device.

The protocol is designed to transmit numeric data between interface registers of different devices, and provides a compact, efficient and reliable method which is easy to monitor and debug.

Data is transmitted in packets: any packet may contain information about one or many registers. This enables blocks of registers to be transferred efficiently.

Means are provided to encode boolean, integer and floating-point (real) values efficiently whilst avoiding the technical difficulties associated with the transmission of binary data directly over serial communications links.

Certain characters are assigned special significance as packet sentinels or numeric sentinels. The use of these specific characters provides for very tight checking on packet integrity and simple bounds checking on packets.
Because of this, communications links using this protocol are self-healing and pick up synchronisation very quickly after system restoration.

All characters used are printable ASCII, and the packet terminator is always the CR character. This means that bus traffic can be viewed by any simple terminal program, every packet occupying a single line on the printout. This feature is a huge aid to diagnostic work with this protocol.

Null characters are never transmitted. This feature simplifies treatment of packet strings in software.

To reduce error rates even further, the ARTPC version has been introduced. This adds an optional check word to each packet.
The addition is designed to be compatible with earlier versions of ARTP.

AmbiLogique Ltd.
1812 Opunake Road, RD29, Hawera 4679, New Zealand
+64 6 764 6567 ph **tech@ambilogic.com.au**

Register Transfer Protocol

A Much Better Idea : Logical : Unique

AmbiLogique

Electronic Controllers

Specification

1 Encoding method.

1.1 Data is transmitted as printable ASCII characters.

1.2 Data is transmitted asynchronously, in serial form.

1.3 Each character is encoded as 8 data bits, no parity, one stop bit (8, N, 1).

2 Data Transmission Speed.

2.1 The default speed is 115,200 bits per second. Other speeds may be used at the discretion of the systems designer.

3 Device Address Hierarchy.

3.1 Devices are addressed hierarchically as a series of PLC's which are networked. Each PLC has a number of slave devices (such as I/O modules, motor drives, etc) plugged into its backplane system. Each slave device may have a number of functional partitions. Finally each functional partition has a number of control and/or status registers associated with it.

3.2 Each PLC (called a "Box") has a boX address associated with it. This is signified by the 'X' symbol which appears, for example, in TERMI input terminals and TERMO output terminals in PLC control diagrams.

3.3 Each slave device plugged into a PLC backplane has a Slot address associated with it. This is signified by the 'S' symbol.

3.4 Each functional partition within a device has a sUbslot address associated with it. This is signified by the 'U' symbol.

3.5 Each interface register in a functional partition has a Register address associated with it. This is signified by the 'R' symbol.

3.6 Within a PLC, provision is made for the individual selection of binary bits from or to an interface register. This selection is carried out by means of a binary Mask, signified by the 'M' symbol. Masks play no part in serial transmission: register values are always transported as complete entities.

4 Encoding of Numerics.

4.1 Numbers are encoded as a special sentinel character followed by a variable-length value field. The sentinel character defines the "size" (data length) of the field.

4.2 The following sizes are defined:-

Size 0 :no additional values are transmitted
Size 1 :8 bits (2 hexadecimal characters) follow the sentinel.
Size 2 :16 bits (4 hexadecimal characters) follow the sentinel.
Size 3 :28 bits (7 hexadecimal characters) follow the sentinel.

4.3 Hexadecimal Characters.

Hexadecimal characters are used to represent numbers from 0 through 15.
Numbers '0' through '9' represent their decimal values.
'A' or 'a' represents 10, 'B' or 'b' represents 11 and so on to 'F' or 'f' which represents 15.
Numbers greater than 15 are represented by strings of hexadecimal characters transmitted with most significant characters first. For example 1000 decimal is transmitted as "3E8" .
On transmission, only upper case characters are used; on reception either upper or lower case is recognised.

4.4 Construction of Numeric Sentinels.

Numeric Sentinels start with a set of 4 Flag bits:-

Bit 3 weight 8 is the 'M(inus)' flag. This signifies that the number is negative.
Bit 2 weight 4 is the 'Y(es)' flag. This signifies that the number is non-zero.
Bits 1 and 0, combined value 0 through 3 are the Size bits.

The flags combine, MYSS, to form a value of 0 through 15. This value is added to the character 'j' (lower case) to give one of the characters 'j' through 'y'. This character is the Numeric Sentinel.

Some examples of Numeric Sentinels:-

```
      MYSS
j     0000  Positive, Zero, size 0, no characters follow, value 0.
n     0100  Positive, Nonzero, size 0, no chars follow, value 1.
v     1100  Negative, Nonzero, no chars follow, value -1.
o1A   0101  Positive, Nonzero, 2 chars, value 26.
w1A   1101  Negative, Nonzero, 2 chars, value -26.
p03E8 0110  Positive, Nonzero, 4 chars, value 1000.
x03E8 1110  Negative, Nonzero, 4 chars, value -1000.
```

4.4.1 Size 3 Numerics.

Size 3 numerics are special: the first hex character which follows is a subsidiary flag field, value 0 through 14, and the remaining 6 characters represent either a 24-bit unsigned integer or a floating-point (real) value with an 8-bit signed exponent and a 16-bit unsigned mantissa.

The subsidiary flag field is represented as FOXx :-

'F' weight 8 is the Floating-point flag.
'O' weight 4 is the Overflow flag, normally used to represent infinite or out-of-range values.
'X' weight 2 is used internally by PLCs to encode the value of the 'Y' flag on the previous scan. This is used to edge-trigger events.
'x' weight 1 is reserved and is always 0.

If the F flag is 0, the following 6 characters represent a 24-bit unsigned integer (maximum value 16,777,215 decimal). Negative integers of this magnitude are encoded by setting the M flag in the main sentinel to make the sentinel character 'y'.

Size 3 Numerics (cont)

If the F flag is 1, the following 2 characters represent a 7-bit plus sign exponent, value -128 through +127. The last 4 characters represent a 16-bit unsigned mantissa (maximum 65535).
The range of floating point values which can be represented in each polarity is from 2.94E-39 to 1.115E+43. This works well for most real-world values.

Note that some slave devices are integer-only. In this case a floating-point value may be misinterpreted.

5 Packet Construction.

5.1 Messages are transmitted in packet form: there are 4 principal packets in use on a master-slave system. These are: Block Request, Block Assert, Block Command, Block Acknowledge.
Each packet consists of a special (unique) sentinel character which defines the packet type, address fields which define the exact registers being transmitted, and a series of value fields.
It is this uniqueness of the packet sentinel and the numeric sentinels which gives ARTP its strength. A very high proportion of transmission errors are detected via format failures, either by corruption of sentinels or by the numeric characters being forced out of the acceptable hexadecimal range, or by failures in the size format indicated by the numeric sentinels. Also, if there is a loss of synchronism due to the transmission medium being interrupted, sync is restored with the first complete packet received after the interruption.

Each field is a numeric value encoded as described in 4. above.
The packet is terminated by the end-of-line character CR (carriage return, 0x0D).

Note that in the examples which follow, the characters are separated by spaces to ease reading. In serial transmission the characters are normally transmitted contiguously.

5.2 Block Request

This packet is issued by a Master device and requests a block of values from a Slave device. The packet is of the form:-

```
- x s u r n c
```

where

- is the sentinel for a Block Request packet;
x is the boX address;
s is the target Slot address;
u is the target sUbslot address;
r is the address of the first Register requested;
n is the Number of registers requested;
c is the carriage return character if no Checkword follows,
or the linefeed character if a Checkword is appended.

5.3 Block Assert

This packet is sent by a Slave device in response to a Block Request received from a Master:-

```
! x s u r n d..d c
```

where
! is the sentinel for a Block Assert packet;
x is the boX address;
s is the source Slot address;
u is the source sUbslot address;
r is the address of the first Register sent;
n is the Number of registers sent;
d..d is the signal Data;
c is the carriage return character if no Checkword follows, or the linefeed character if a Checkword is appended.

5.4 Block Command

This packet is sent by a Master device and pushes a block of values into a Slave device's registers:-

```
+ x t u r n d..d c
```

where
+ is the sentinel for a Block Command packet encoded;
x is the boX address;
t is the target Slot address;
u is the target sUbslot address;
r is the address of the first Register sent;
n is the Number of registers sent;
d..d are the register Data values;
c is the carriage return character if no Checkword follows, or the linefeed character if a Checkword is appended.

5.5 Block Acknowledge

This packet is sent by a Slave device and advises its Master how well the Block Command went :-

```
* x s u r e c
```

where
* is the sentinel defining a Block Acknowledge packet;
x is the boX address;
s is the slave Slot address;
u is the slave sUb-slot address;
r is the address of the first Register in error;
e is the error code: 0 is no error, other errors as defined by the device;
c is the carriage return character if no Checkword follows, or the linefeed character if a Checkword is appended.

Note that in the case of no error, the Error code is 0 and the Register value is one more than the last Register value received.
For example, if the base register was 2 and 3 registers were received, the last register received is 4.
The Register value returned in the Block Acknowledge packet is therefore 5.

5.6 Dialogues.

There are two possible dialogues in a master-slave setup. Each dialogue is initiated by the Master, and the addressed Slave responds.

- Master initiaties with Block Request : Slave replies with Block Assert.
  Data is transferred from Slave to Master.

- Master initiates with Block Command : Slave replies with Block Acknowledge.
  Data is transferred from Master to Slave.

[Specification continues...]

6 Addition of Checkword

Introduced in Rev 2.0
Note that the whole of this section 6 should be regarded as highlighted in Rev 2.0

The natural strength of the protocol (in terms of error detection) can be subtantially enhanced by the transmission and checking of a checkword.
The checkword method set out here is much stronger than a simple checksum but not as strong as a bitwise CRC. The ARTP method has the advantage of consuming very little computing resource whilst providing strong error detection.

6.1 Checkword in Principle

The Checkword is a 16-bit (unsigned) word which is calculated on the entire transmitted packet.
In previous implementations of this standard, it was permissible to terminate a packet with either the carriage return character (CR, 0x0D) or the line feed character (LF, 0x0A).
In practice within AmbiLogique, its associated companies, and throughout the customer base, the CR character was invariably used.
In this revision of the standard, and in future, the use of the CR character where a packet has no checkword is mandatory. The CR character is the sole and unique packet terminator sentinel.
Where a packet carries a checkword, the normal part of the packet is terminated in the LF character. The checkword is then appended as 4 hexadecimal characters, most significant first, then the packet terminates in CR.

The implications of this are that devices implementing this standard are compatible with older Rev 0/1 devices as follows:-

- Rev 2+ Master talks to Rev 2+ Slave:-

  Master sends packet with Checkword to Slave.
  Slave checks packet including Checkword.
  If OK, Slave sends response with Checkword.
  Master checks response including Checkword.
  If error, Master repeats dialogue.

- Rev 2+ Master talks to Rev 0/1 Slave:-

  Master sends packet with Checkword to Slave.
  Slave accepts packet up to LF and checks packet without Checkword.
  If OK, Slave sends response without Checkword, but ending in CR.
  Slave dumps Checkword and CR as garbage.
  Master receives response ending in CR with no Checkword.
  Master checks response but assumes that non-existent Checkword is OK.
  If non-checkword tests show up an error, Master repeats dialogue.

- Rev 0/1 Master talks to Rev 2+ Slave:-

  Master sends packet with no Checkword, ending in CR to Slave.
  Slave sees packet end in CR with no Checkword, knows that it is Rev 0/1.
  Slave checks the packet, assumes non-existent Checkword is OK.
  If OK, Slave responds either with Rev 0/1 packet ending in CR with no Checkword, or with Rev 2+ packet with Checkword.
  If Master receives Rev 0/1 packet, processes it as normal.
  If Master receives Rev 2+ packet, assumes that it ends in LF and dumps Checkword and CR. Checks packet as Rev 0/1 as normal.

- Rev 0/1 Master talks to Rev 0/1 Slave:-

  Dialog is standard Rev 0/1 with no Checkwords.

6.2 Method of Checkword Calculation

The Checkword is generated as a 16-bit unsigned number.
The register in which the calculation is carried out is called the "Checkword Register."
In the following explanation, it is assumed that the Checkword and its Register are laid out bitwise horizontally with the most significant bit on the left.

```
Bit16 bit15 ..... bit1 bit0.
```

The Checkword calculation is carried out on the encoded packet, i.e. on the sequence of characters actually transmitted on the communications channel.

The Checkword is calculated as follows:-

6.2.1 Either (a) the packet sentinel is placed in the least significant 8 bits of the register, the most significant bits cleared to 0, and the entire register inverted, or (b) the register is preset to 0xFFFF (all 1's) and the packet sentinel XOR'd into the least significant bits.

6.2.2 For each subsequent character in the packet, the Register is rotated left by 3 bits, then the new character is XOR'd into the least significant bits.

6.2.3 A method of carrying out the rotation which is very computationally efficient is to implement the Register as a union:-

```
typedef union
{
      U32 w;
      U16 r[2];
}
CWReg;
```

Indices L and H need to be defined with values 0 and 1 so that r[L] aligns with the least significant 16 bits of w, and r[H] aligns with the most significant 16 bits of w.

The Register is initialised by setting CWReg.w to 0x0000FFFF;

For each character:-

(a) CWReg.r[H] is cleared to 0;
(b) CWReg.w is shifted left by 3 bits;
(c) CWReg.r[H] is OR'd into CWReg.r[L];
(d) the character is cast to unsigned 16-bit then XOR'd into CWReg.r[L]

Note that steps (a) through (c) are irrelevant in the case of the first character (the packet sentinel). However, the method above requires less code if no special case is made for the packet sentinel.

After every character in the main body of the packet has been included, including the LF character which defines the transition between the data body and the Checkword, the Checkword is complete and ready for transmission.

6.3 Method of Checkword Encoding

The Checkword is encoded in the same way as other integers, i.e. as 4 hexadecimal characters with the most significant 4 bits transmitted first. **Upper case characters only must be used for Checkword digits between 'A' and 'F'.**

6.4 Method of Checkword Checking

The packet is checked for integrity at the receiving end by
(a) calculating the Checkword for the received packet up to and including the LF character,
(b) converting it to an **upper case hex string** and
(c) comparing it with the received Checkword string.

Note that if both upper and lower case characters are allowed in the Checkword string, errors which simply change the case of the transmitted Checkword will bypass the error check.

6.5 Strength Tests

The Checkword method has been
tested for strength by setting up a typical packet including a Checkword in accordance with this specification.
Clearly any and all errors within the Checkword itself will be detected.

The strength tests were carried out as follows:-

- A single bit error was introduced at every position in the packet body up to and including all bits of the Checkword. The terminating CR was not subject to corruption because this would not affect the Checkword test. In practice, corruption of the terminating CR will provoke a format failure and therefore flag an error.
- For each position, a Checkword was calculated, and the test number incremented.
- If the Checkword calculated in this way matched the reference Checkword, the fail count was incremented.
- When all bit positions had been considered, the result was recorded.
- The test was then repeated for 2 adjacent bits in error, 3.. and so on.
- Each size of error run yields [p-r-7] results where p is the number of bits in the packet and r is the size of error run. The 7 factor avoids testing the terminating CR.
- The test was repeated for runs of bits forced to 0.
- The test was repeated for runs of bits forced to 1.

Here is the typical packet which we tested. It is a Block Assert from an EXDA-01 in Slot 1 in response to a Data Request for the values in Registers 2 and 3, the analogue inputs. It is assumed that the values in each of these registers is 0.5 (mid-range). This causes significant strings of 0's to be encoded.

This packet is among the longest which are encountered in normal systems operation, and is therefore among the most vulnerable.

Note that spaces have been artificially inserted to aid legibility:-

```
  x s u  r   n    d        d       LF   CW   CR
! j n j o02 o02 qAF08000 qAF08000 \x0A 48BF \x0D
```

This gives a byte count of 32, or 256 bits. There are therefore 248 single-bit burst conditions, 247 2-bit bursts ... etc.

Here is the calculation of the Checkword:-

op	hex	r[H]	r[L]	----	----	r[L]	hex
Init		000	1111	1111	1111	1111	FFFF
!	21				0010	0001	0021
xor			1111	1111	1101	1110	FFDE
<<3		111	1111	1110	1111	0	
L |= H		111	1111	1110	1111	0111	FEF7
j	6A				0110	1010	006A
xor			1111	1110	1001	1101	FE9D
<<3		111	1111	0100	1110	1	
L |= H		111	1111	0100	1110	1111	F4EF
n	6E				0110	1110	006E
xor			1111	0100	1000	0001	F481
<<3		111	1010	0100	0000	1	
L |= H			1010	0100	0000	1111	A40F
j	6A				0110	1010	006A
xor			1010	0100	0110	0101	A465
rot 3l		101	0010	0011	0010	1101	232D
o	6F				0110	1111	006F
xor			0010	0011	0100	0010	2342
rot 3l		001	0001	1010	0001	0001	1A11
0	30				0011	0000	0030
xor			0001	1010	0010	0001	1A21
rot 3l		000	1101	0001	0000	1000	D108
2	32				0011	0010	0032
xor			1101	0001	0011	1010	D13A
rot 3l		110	1000	1001	1101	0110	89D6
o	6F				0110	1111	006F
xor			1000	1001	1011	1001	89B9
rot 3l		100	0100	1101	1100	1100	4DCC
0	30				0011	0000	0030
xor			0100	1101	1111	1100	4DFC
rot 3l		010	0110	1111	1110	0010	6FE2
2	32				0011	0010	0032
xor			0110	1111	1101	0000	6FD0
rot 3l		011	0111	1110	1000	0011	7E83
q	71				0111	0001	0071
xor			0111	1110	1111	0010	7EF2
rot 3l		011	1111	0111	1001	0011	F793
A	41				0100	0001	0041
xor			1111	0111	1101	0010	F7D2
rot 3l		111	1011	1110	1001	0111	DE97
F	46				0100	0110	0046
xor			1011	1110	1101	0001	BED1
rot 3l		101	1111	0110	1000	1101	F68D
0	30				0011	0000	0030
xor			1111	0110	1011	1101	F6BD
rot 3l		111	1011	0101	1110	1111	B5EF
8	38				0011	1000	0038
xor			1011	0101	1101	0111	B5D7
rot 3l		101	1010	1110	1011	1101	AEBD

```
op   hex   r[H] r[L] ---- ---- r[L]  hex
---------------------------------------
0    30                  0011 0000  0030
xor            1010 1110 1000 1101  AE8D
rot 3l      101 0111 0100 0110 1101  746D
               0111 0100 0110 1101  746D
0    30                  0011 0000  0030
xor            0111 0100 0101 1101  745D
rot 3l      011 1010 0010 1110 1011  A2EB
0    30                  0011 0000  0030
xor            1010 0010 1101 1011  A2DB
rot 3l      101 0001 0110 1101 1101  16DD
q    71                  0111 0001  0071
xor            0001 0110 1010 1100  16AC
rot 3l      000 1011 0101 0110 0000  B560
A    41                  0100 0001  0041
xor            1011 0101 0010 0001  B521
rot 3l      101 1010 1001 0000 1101  A90D
F    46                  0100 0110  0046
xor            1010 1001 0100 1011  A94B
rot 3l      101 0100 1010 0101 1101  4A5D
0    30                  0011 0000  0030
xor            0100 1010 0110 1101  4A6D
rot 3l      010 0101 0011 0110 1010  536A
8    38                  0011 1000  0038
xor            0101 0011 0101 0010  5352
rot 3l      010 1001 1010 1001 0010  9A92
0    30                  0011 0000  0030
xor            1001 1010 1010 0010  9AA2
rot 3l      100 1101 0101 0001 0100  D514
0    30                  0011 0000  0030
xor            1101 0101 0010 0100  D524
rot 3l      110 1010 1001 0010 0110  A926
0    30                  0011 0000  0030
xor            1010 1001 0001 0110  A916
rot 3l      101 0100 1000 1011 0101  48D5
LF   0A                  0000 1010  000A
xor            0100 1000 1011 1111  48BF
CW   48BF
```

The test sequence was carried out for burst errors of 1 to 25 bits long, resulting in up to 4 sequential corrupted characters.
The error modes tested included bit inversion, force to 0, force to 1.
There were no cases of undetected errors.

END of specification.

Appendix 1. Detection Strategy.

This is one of many possible strategies for detection and interpretation of incoming packets.

This strategy is especially suitable for implementing in a drop-through handler because the amount of computation needed at each stage of the state machine is minimal.

It is assumed that incoming packets arrive into a FIFO buffer such as a ring. This means that as characters arrive, they accumulate in the buffer until either (a) a complete packet is waiting in the buffer or (b) they are removed head-first from the buffer.

A1.1. State Machine.

A1.1.1. The incoming packet is analysed according to a state machine with these states:-

- 0 = IDLE
  - The channel is processing garbage and is dumping any character which is not a packet sentinel.
- 1 = GOTSENT
  - The channel has a packet sentinel lodged in its head position, and is accumulating a packet in the FIFO.
- 2 = GOTHDR
  - The channel has 5 numbers (boX, Slot, sUbslot, Register, Number of regs) lodged in its head. The numbers have been checked for integrity, and the number of numbers in the entire packet has been determined as 5 + Numregs.
- 3 = GOTBODY
  - The channel has all of the numbers required plus one character (which can be either CR or LF).
- 4 = BODYOK
  - The channel has had all of its numbers checked for integrity and has determined that the end-of-body character is LF.
  - If the end-of-body character is CR, the state is advanced to CWOK.
- 5 = GOTCW
  - The channel contains the body plus 5 characters.
- 6 = CWOK
  - Either (a) there was no Checkword or (b) the Checkword calculated from the body matches the Checkword in the channel buffer.
  - The packet is now available for processing.

A1.2. Errors.

A1.2.1. The following errors are recognised.

- -1 = NOREP
  - No response was seen within the allowed timeout after a Master had sent a packet to a Slave (all Master packets should produce a response).
- -2 = TIMEOUT
  - A packet sentinel was received within the permitted time after a Master had sent a packet, but the packet was not completed within the allotted time.
- -3 = SENTERR
  - The packet sentinel is not one of the expected types for the device and channel. For example, a slave device does not expect to see a Block Assert or a Block Acknowledge to be input other than as an echo of something it has sent.
- -4 = FORMERR
  - There is a format error. This can be generated by:-
    - A numeric sentinel not appearing where it should;
    - Characters following a numeric sentinel not being the correct number of hex digits;
    - The end-of -body character not being LF or CR;
    - The 4 characters after LF not being upper-case hex characters;
    - The 5th character after LF not being CR.
- -5 = CWERR
  - This is generated by the calculated Checkword, expressed as a 4-digit upper-case hex string not matching the 4 hex digits received between LF and CR.

A1.2.2. Any of these errors cause the packet to be discarded, and Garbage Disposal action taken.

A1.2.3. An error value of 0 is regarded as INDET which means that the packet error status has not been determined. This value is set when the channel is cleared.

A1.2.4. An error value of 1 or more is regarded as NOERR and the packet is assumed to be OK.

A1.3. Garbage Disposal (Dumping).

A1.3.1. Garbage Disposal or Dumping is the process of clearing unwanted characters out of the receive FIFO.

A1.3.2. This is best achieved by scanning all of the characters in the FIFO and stopping either

a) when all of the characters have been checked or

b) when a packet sentinel is detected.

A1.3.3. The unwanted characters can then be deleted from the FIFO in a block.

A1.3.4. If a packet sentinel has been found, it now resides at the head of the FIFO. The state machine can now be advanced to GOTSENT.

A1.3.5. If a packet which is building in the FIFO is found to have any error, or if it is no longer needed, it can be removed by deleting the packet sentinel from the head of the FIFO. The Garbage Disposal strategy will then remove the remainder of the packet along with any other garbage on the communications channel, until the next packet sentinel arrives.

A1.4. The NumNums Function

A1.4.1. This function is used when the state machine is at GOTSENT or higher.

A1.4.2. Starting at FIFO position 1 (i.e. the first character after the packet sentinel), the size of the first field is determined.

A1.4.3. The position of the next numeric sentinel can then be determined, and the number of numbers stored in the FIFO is incremented.

A1.4.4. The process is repeated until the end of the FIFO is reached.

A1.4.5. An incomplete number at the end of the FIFO is not regarded as an error.

A1.4.6. The number of numbers is returned.

A1.4.7. If a character is found which is not a numeric sentinel:-

a) the error status is set to FORMERR;

b) the packet sentinel is deleted from the head of the FIFO.

c) the state machine is set to IDLE.

A1.4.8. The skip distances for the different sentinels (arranged as for a 'case' statement) are:-

s	MYSS	Dist	Meaning
j	**0000**	**1**	**0**
n	**0100**	**1**	**1**
r	1000	1	-0*
v	**1100**	**1**	**-1**
k	0001	3	8-bit zero*
o	**0101**	**3**	**8-bit positive integer**
s	1001	3	8-bit negative zero*
w	**1101**	**3**	**8-bit negative integer**
l	0010	5	16-bit zero*
p	**0110**	**5**	**16-bit positive integer**
t	1010	5	16-bit negative zero*
x	**1110**	**5**	**16-bit negative integer**
m	0011	8	32-bit zero*
q	**0111**	**8**	**32-bit positive integer or float**
u	1011	8	32-bit negative zero*
y	**1111**	**8**	**32-bit negative integer or float**

- **9 Bold** items are in common use.
- * 7 asterisked items unlikely to be encountered.

A1.5. Post-Check Actions

A1.5.1. Block Request.

a) Slave Device assembles and transmits Block Assert.

A1.5.2. Block Assert.

a) Master Device places register values in appropriate storage.

A1.5.3. Block Command.

a) Slave Device checks for command errors, assembles and transmits Block Acknowledge.

b) Slave Device carries out command.

A1.5.4. Block Acknowledge.

a) Master Device checks error code and takes corrective action if necessary.

END of Appendix A.