

- [NAME](#)
- [SYNOPSIS](#)
- [DESCRIPTION](#)
 - [Time and Date Usage](#)
 - [Data Representation](#)
 - [Bit Rules](#)
 - [Byte Rules](#)
 - [Endian Rules](#)
- [UTILITY SUBROUTINES](#)
 - [about\(\\$program, \\$version\)](#)
 - [alert\(\\$keyword, @text\)](#)
 - [\\$boolean = bitand\(\\$int1, \\$int2\)](#)
 - [\\$hex = binToHexStr\(\\$bin\)](#)
 - [\\$seconds = dateTimeToSeconds\(\\$datetime\)](#)
 - [error\(@text\)](#)
 - [fatal\(@text\)](#)
 - [\\$bin = hexToBinStr\(\\$hex\)](#)
 - [\\$boolean = isAtdfEndianBig\(\)](#)
 - [\\$boolean = isAtdfEndianLittle\(\)](#)
 - [\\$boolean = isOSEndianBig\(\)](#)
 - [\\$boolean = isOSEndianLittle\(\)](#)
 - [\\$boolean = isStdFEndianBig\(\)](#)
 - [\\$boolean = isStdFEndianLittle\(\)](#)
 - [loggText\(@text\)](#)
 - [osCpuType\(\)](#)
 - [\\$singleQuotedText = q1\(@text\)](#)
 - [reverseEndian\(\x\)](#)
 - [\\$scaleFactor = scaleFactor\(\\$units\)](#)
 - [\\$datetime = secondsToDateTime\(\\$seconds\)](#)
 - [setBitFlag\(\sstr, \\$map, \\$flg\)](#)
 - [stderr\(@text\)](#)
 - [stdout\(@text\)](#)
 - [trim\(\stext\)](#)
- [ATDF UTILITY SUBROUTINES](#)
 - [\\$atdfFieldName = atdfFieldName\(\\$atdfRecordName, \\$atdfFieldIndex\)](#)
 - [\\$atdfFieldNameIndexLast = atdfFieldNameIndexLast\(\\$atdfRecordName\)](#)
 - [atdfFormatReal\(\sreal\)](#)
 - [@atdfFieldNames = atdfRecordFieldNames\(\\$atdfRecordName\)](#)
 - [atdfRecordWrap\(\\$atdfRecordName, \@atdfRecord\)](#)
- [ATDF TO STDF DATA TYPE SUBROUTINES](#)
 - [\\$result = A2S_do\(\\$stdfDataTypeCode, @arguments\)](#)
 - [\\$binary = A2S_B0\(\)](#)
 - [\\$binary = A2S_B1\(\\$bitNumber\)](#)
 - [\\$binary = A2S_Bn\(\\$hexNumber\)](#)
 - [\\$binary = A2S_C1\(\\$ascii\)](#)
 - [\\$binary = A2S_Cn\(\\$ascii\)](#)

- [\\$binary = A2S_Dn\(@positiveIntegerNumbers\)](#)
- [\\$binary = A2S_I1\(\\$signedByte\)](#)
- [\\$binary = A2S_I2\(\\$signedShort\)](#)
- [\\$binary = A2S_I4\(\\$signedLong\)](#)
- [\\$binary = A2S_N1\(\\$hexchr\)](#)
- [\\$binary = A2S_Nn\(\\$hexnum\)](#)
- [\\$binary = A2S_R4\(\\$realSingle\)](#)
- [\\$binary = A2S_R8\(\\$realDouble\)](#)
- [\\$binary = A2S_U1\(\\$unsignedByte\)](#)
- [\\$binary = A2S_U2\(\\$unsignedShort\)](#)
- [\\$binary = A2S_U4\(\\$unsignedLong\)](#)
- [\\$binary = A2S_xTYPE\(\&type, \@data\)](#)
- [STDF UTILITY SUBROUTINES](#)
 - [\\$stdfEndian = stdfEndian\(\[stdfCpuType\]\)](#)
 - [\\$stdfFieldName = stdfFieldName\(\\$stdfRecordName, \\$stdfFieldIndex\)](#)
 - [\\$stdfFieldNameIndexLast = stdfFieldNameIndexLast\(\\$stdfRecordName\)](#)
 - [@stdfFieldNames = stdfRecordFieldNames\(\\$stdfRecordName\)](#)
 - [\(\\$stdfRecTypeCode, \\$stdfRecSubTypeCode\) = stdfRecordNameToTypeCodes\(\\$stdfRecName\)](#)
 - [\\$stdfRecName = stdfRecordTypeCodesToName\(\\$stdfRecTyp, \\$stdfRecSub\)](#)
 - [\\$stdfRecordTypeToDataTypeCode = stdfRecordTypeToDataTypeCode\(\\$stdfRecordType\)](#)
- [STDF TO ATDF DATA TYPE SUBROUTINES](#)
 - [\\$result = S2A_do\(\\$stdfDataTypeCode, @arguments\)](#)
 - [\\$hexnum = S2A_B1\(\\$binary\)](#)
 - [\\$hexnum = S2A_Bn\(\\$binary\)](#)
 - [\\$char = S2A_C1\(\\$binary\)](#)
 - [\\$text = S2A_Cn\(\\$binary\)](#)
 - [@intnum = S2A_Dn\(\\$binary\)](#)
 - [\\$signedByte = S2A_I1\(\\$binary\)](#)
 - [\\$signedShort = S2A_I2\(\\$binary\)](#)
 - [\\$sint = S2A_I4\(\\$binary\)](#)
 - [\\$hex = S2A_N1\(\\$binary\)](#)
 - [\\$hexnum = S2A_Nn\(\\$binary\)](#)
 - [\\$real = S2A_R4\(\\$binary\)](#)
 - [\\$real = S2A_R8\(\\$binary\)](#)
 - [\\$unsignedByte = S2A_U1\(\\$binary\)](#)
 - [\\$unsignedShort = S2A_U2\(\\$binary\)](#)
 - [\\$uint = S2A_U4\(\\$binary\)](#)
 - [S2A_Vn\(\)](#)
 - [@txt-dat = S2A_xCn\(\\$x, \\$bindat\)](#)
 - [@value = S2A_xTYPE\(\\$x, \\$dataType, \\$bindat\)](#)
- [DEBUG SUBROUTINES](#)
 - [debugData\(\@varRefs, \@varNames\)](#)
 - [debugDataBin\(\\$datnam, \\$bindat\)](#)
 - [debugDataPurdy\(\@varRefs, \@varNames\)](#)
 - [debugSub\(\\$subnam, @_\)](#)
 - [debugText\(@text\)](#)
 - [@fmtbin = debugFmtBin\(\\$bindat\)](#)

- [dumpData\(\@varRefs, \@varNames\)](#)
 - [SUPPORTED PLATFORMS](#)
 - [REQUIRES](#)
 - [SEE ALSO](#)
 - [AUTHOR](#)
 - [COPYRIGHT](#)
 - [ACKNOWLEDGEMENT](#)
 - [KNOWN BUGS](#)
 - [HISTORY](#)
-
-

NAME

TDF - A library of subroutines for STDF version 4 and ATDF version 2 files.

SYNOPSIS

1. `use TDF;`
 2. Call subroutines.
-

DESCRIPTION

Contains standard subroutines for manipulating STDF version 4 and ATDF version 2 files and their contents.

This module is required by the `atdf2_to_std4.pl` and `std4_to_atdf2.pl` programs.

Time and Date Usage

(Quoted from Teradyne STDF specification)

The date and time field used in this specification is defined as a four byte (32 bit) unsigned integer field measuring the number of seconds since midnight on January 1st, 1970, in the local time zone. This is the UNIX standard base time, adjusted to the local time zone.

Data Representation

(Quoted from Teradyne STDF specification)

When data is shared among systems with unlike central processors, the problem arises that there is little or no standardization of data representation (that is, the bit ordering of various data types) among the various processors of the world. For example, the data representations for DEC, Motorola, Intel, and IBM computers are all different, even though at least two of them adhere to the IEEE floating point standard. Moreover, different processors made by the same company sometimes store data in incompatible ways. To address this problem, the STDF specification uses a field called CPU_TYPE in the File Attributes Record (FAR). This field indicates the type of processor that wrote the data (for example, Sun series or DEC-11 series). The field is used as follows:

- When writing an STDF file, a system uses its own native data representation. The type of the writing processor is stored in the CPU_TYPE field.
- When reading an STDF file, a system must convert the records to its own native data representation as it reads them, if necessary. To do so, it checks the value of the CPU_TYPE field in the FAR, which is the first record in the file. Then, if the writing CPU data representation is incompatible with its own, it uses a subroutine that reads the next (or selected) record and converts the records to its own data representation as it reads them.

Bit Rules

This module internally operates on all numeric and text bit encoded values in left (MSB) to right (LSB) order.

Numeric	MSB -> LSB
Binary	0b00000000
Hex	0x00
Octal	0000
Text	MSB -> LSB
Binary	'00000000'
Hex	'00'
Octal	'0000'

Byte Rules

This module internally operates on all numeric and text byte encoded values in left (MSB) to right (LSB) order.

Endian Rules

``Little Endian" means that the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. (The little end comes first.) For example, a 4 byte LongInt

```
Byte3 Byte2 Byte1 Byte0
```

will be arranged in memory as follows:

```
Base Address+0   Byte0
Base Address+1   Byte1
Base Address+2   Byte2
Base Address+3   Byte3
```

Intel processors (those used in PC's) use ``Little Endian" byte order.

``Big Endian" means that the high-order byte of the number is stored in memory at the lowest address, and the low-order byte at the highest address. (The big end comes first.) Our LongInt, would then be stored as:

```
Base Address+0   Byte3
Base Address+1   Byte2
Base Address+2   Byte1
Base Address+3   Byte0
```

The order of the bytes is referred to as the ``endian". In a little endian CPU the storage of bytes in memory is address 0 is LSB, address +1, is next most significant, and address +n is the most significant byte. In a big endian CPU the storage of bytes in memory is address 0 is MSB, address +1 is the next most significant, and address +n is the least significant byte.

VAX and x86 are little endian CPU and SUN SPARC, HP PARC, and PowerPC are big endian CPU.

```

                MSB LSB
Example:  12  34 hex
-----
| ADDR | LITTLE | BIG |
|=====|=====|=====|
| 0000 |      34 |   12 |
| 0001 |      12 |   34 |
-----

```

Therefore, if the STDF file was written in big endian and the system executing an STDF conversion program is a little endian then the bytes in each multibyte field must be swapped so that conversion of the bytes to numeric values will be correct.

UTILITY SUBROUTINES

about(\$program, \$version)

Outputs an about message to standard output.

alert(\$keyword, @text)

Outputs an alert message to a log file and returns the message. An alert message is formatted as:

```
PROGRAM_NAME *KEYWORD* TEXT
```

\$boolean = bitand(\$int1, \$int2)

Performs a bitwise AND of two integers and returns true if the result contains one or more bits that are set to 1.

\$hex = binToHexStr(\$bin)

Converts a binary string to a hexadecimal string. For example, the binary string ``01001010" would be converted to the

hexadecimal string ``4A".

\$seconds = dateTimeToSeconds(\$datetime)

Converts a string containing a date and time value formatted as ``[h]h:[m]m:[s]s [d]d-MMM-YYYY" to epoch seconds.

error(@text)

Concatenates the values of the elements of the list @text using spaces into a single string formatted as a standard error message and outputs the error message to standard error. The format of the error message is:

```
ERROR MESSAGE
```

fatal(@text)

Concatenates the values of the elements of the list @text using spaces into a single string formatted as a standard fatal message, outputs the fatal message to standard error, closes the open input file, and exits the executing program to the command line. The format of the fatal message is:

```
FATAL MESSAGE
```

\$bin = hexToBinStr(\$hex)

Converts a hexadecimal string to a binary string. For example, the hexadecimal string ``4A" would be converted to the binary string ``01001010".

\$boolean = isAtdfEndianBig()

Tests if the value of \$gEndian{'ATDF'} is 'ENDBIG' and returns the boolean result of the test.

\$boolean = isAtdfEndianLittle()

Tests if the value of \$gEndian{'ATDF'} is 'ENDLIT' and returns the boolean result of the test.

\$boolean = isOSEndianBig()

Tests if the value of \$gOS{'ENDIAN'} is 'ENDBIG' and returns the boolean result of the test.

\$boolean = isOSEndianLittle()

Tests if the value of \$gOS{'ENDIAN'} is 'ENDLIT' and returns the boolean result of the test.

\$boolean = isStdEndianBig()

Tests if the value of \$gEndian{'STDF'} is 'ENDBIG' and returns the boolean result of the test.

\$boolean = isStdEndianLittle()

Tests if the value of \$gEndian{'STDF'} is 'ENDLIT' and returns the boolean result of the test.

loggText(@text)

Concatenates the values of the elements of the list @text using spaces into a single string formatted as a standard log message and then outputs the message to standard out. The log message is formatted as:

```
[ PROGRAM_NAME ] MESSAGE
```

osCpuType()

Returns the CPU type of the operating system as an STDF CPU_TYPE field code.

\$singleQuotedText = q1(@text)

Concatenates the values of the elements of the list @text using spaces into a single string formatted and returns the string surrounded by single quotation characters.

reverseEndian(\\$x)

Tests to determine if the endian of the STDF and the operating system are not equal and reverses the byte sequence of the STDF value to match the endian-ness of the operating system.

This is required to interpret floating point numbers correctly.

\$scaleFactor = scaleFactor(\$units)

\$datetime = secondsToDateTime(\$seconds)

Converts an epoch seconds value to a string containing a date and time value formatted as ``[h]h:[m]m:[s]s [d]d-MMM-YYYY".

setBitFlag(\\$str, \\$map, \\$flg)

stderr(@text)

Concatenates the values of the elements of the list @text using spaces into a single string and prints the string to standard error.

stdout(@text)

Concatenates the values of the elements of the list @text using spaces into a single string and prints the string to standard output.

trim(\\$text)

Deletes the leading and trailing white space characters from the text string.

ATDF UTILITY SUBROUTINES

\$atdfFieldName = atdfFieldName(\$atdfRecordName, \$atdfFieldIndex)

Returns the ATDF field name for the specified ATDF record name and field index.

\$atdfFieldNameIndexLast = atdfFieldNameIndexLast(\$atdfRecordName)

Returns a the index of the last field name in the ordered list of the ATDF field names for the specified ATDF record name.

atdfFormatReal(\\$real)

@atdfFieldNames = atdfRecordFieldNames(\$atdfRecordName)

Returns an ordered list of the ATDF field names for the specified ATDF record name.

atdfRecordWrap(\$atdfRecordName, \@atdfRecord)

Wraps the ATDF record to a maximum of 80 characters per line.

ATDF TO STDF DATA TYPE SUBROUTINES

STDF binary values are always written to the STDF file in little endian format independent of the runtime or target operating systems!

\$result = A2S_do(\$stdfDataTypeCode, @arguments)

Executes the ATDF to STDF data type conversion subroutine for the specified STDF data type code and arguments and returns the result.

\$binary = A2S_B0()

Returns an STDF B*0 null binary byte.

\$binary = A2S_B1(\$bitNumber)

Converts an ATDF numeric value between 0 and 255 to an STDF B*1 data type which is a one byte bit encoded value:

b7 b6 b5 b4 b3 b2 b1 b0

therefore the value can be packed directly without any conversion.

\$binary = A2S_Bn(\$hexNumber)

Converts an ATDF hexadecimal number to an STDF B*n binary value.

\$binary = A2S_C1(\$ascii)

Converts an ATDF character value to an STDF C*1 binary value.

\$binary = A2S_Cn(\$ascii)

Converts an ATDF character string of one or more ASCII characters to an STDF C*n binary value where the first byte is the number of bytes in the string followed by one byte for each character in the string.

\$binary = A2S_Dn(@positiveIntegerNumbers)

Converts a list of ATDF positive non-zero integer numbers to an STDF D*n binary bit field. The bit field is organized as:

Byte:	0	1	2	...
Bit:	7 ... 0	15 ... 8	23 ... 9	...
Integer:	8 ... 1	16 ... 9	24 ... 17	...

Where each bit in a byte corresponds to a positive integer value. The maximum positive integer value is 65535. For example, a list containing 2, 8, 11, and 15 would be encoded into a two byte bit field as:

Bit: 10000010 01000100

Returns a binary string formatted as:

[2:bit count][N:bit field]

Where N = bit count.

\$binary = A2S_I1(\$signedByte)

Converts an ATDF one byte signed integer to an STDF I*1 binary.

\$binary = A2S_I2(\$signedShort)

Converts an ATDF two byte signed integer (i.e., a short) to an STDF I*2 binary.

\$binary = A2S_I4(\$signedLong)

Converts an ATDF four byte signed integer (i.e., a long) to an STDF I*4 binary value.

\$binary = A2S_N1(\$hexchr)

Converts one or two ATDF hexadecimal digits to an STDF N*1 binary value.

The STDF N*1 data type represents an unsigned integer data stored as a nibble (nibble = 4 bits of a byte). N1 argument values are always one or two ASCII hexadecimal digits. The packed binary must be a full byte. Therefore, if there is only one argument digit, then a '0' is prepended to make two digits or one byte. For example, if the argument is 'A' then '0A' is packed into the byte.

\$binary = A2S_Nn(\$hexnum)

Converts a string of one or more hexadecimal digits to an binary value. If the hexadecimal string contains an odd number of digits then a zero is pre-pended to the right of the string to create a binary value with an even number of bytes.

\$binary = A2S_R4(\$realSingle)

Converts an ATDF four byte signed real (i.e., a single) to an STDF R*4 binary value.

\$binary = A2S_R8(\$realDouble)

Converts an ATDF eight byte signed real (i.e., a double) to an STDF R*8 binary value.

\$binary = A2S_U1(\$unsignedByte)

Converts an ATDF one byte unsigned integer value to an STDF U*1 binary value.

\$binary = A2S_U2(\$unsignedShort)

Converts an ATDF two byte unsigned integer (i.e., a short) to an STDF U*2 binary value.

\$binary = A2S_U4(\$unsignedLong)

Converts an ATDF four byte unsigned integer (i.e., a long) to an STDF U*4 binary value.

\$binary = A2S_xTYPE(\&type, \@data)

Converts the value of each element of the array referenced by \@data to binary using the ATDF to STDF data type conversion subroutine reference by \&type and returns a string containing the concatenated binary values.

STDF UTILITY SUBROUTINES

\$stdfEndian = stdfEndian([\$stdfCpuType])

Returns the STDF endian keyword and optionally sets the STDF endian keyword if a valid STDF CPU_TYPE is specified.

\$stdfFieldName = stdfFieldName(\$stdfRecordName, \$stdfFieldIndex)

Returns the STDF field name for the specified STDF record name and field index.

\$stdfFieldNameIndexLast = stdfFieldNameIndexLast(\$stdfRecordName)

Returns a the index of the last field name in the ordered list of the STDF field names for the specified STDF record name.

@stdfFieldNames = stdfRecordFieldNames(\$stdfRecordName)

Returns an ordered list of the STDF field names for the specified STDF record name.

(\$stdfRecTypeCode, \$stdfRecSubTypeCode) = stdfRecordNameToTypeCodes(\$stdfRecName)

\$stdfRecName = stdfRecordTypeCodesToName(\$stdfRecTyp, \$stdfRecSub)

\$stdfRecordTypeToDataTypeCode = stdfRecordTypeToDataTypeCode(\$stdfRecordType)

Returns a the index of the last field name in the ordered list of the STDF field names for the specified STDF record name.

STDF TO ATDF DATA TYPE SUBROUTINES

STDF binary values are always written to the STDF file in little endian format independent of the runtime or target operating systems and are byte swapped for compatibility with the runtime operating system.

\$result = S2A_do(\$stdfDataTypeCode, @arguments)

Executes the STDF to ATDF data type conversion subroutine for the specified STDF data type code and arguments and returns the result.

\$hexnum = S2A_B1(\$binary)

Converts an STDF B*1 binary value to an ATDF bit number value.

\$hexnum = S2A_Bn(\$binary)

Converts an STDF B*n binary value to an ATDF bit number string value.

\$char = S2A_C1(\$binary)

Converts an STDF C*1 binary value to an ATDF character value.

\$text = S2A_Cn(\$binary)

Converts an STDF C*n binary value to an ATDF character string.

@intnum = S2A_Dn(\$binary)

Converts an STDF binary bit field to a list of ATDF positive non-zero integer numbers. (See A2S_Dn for a detailed explanation.)

\$signedByte = S2A_I1(\$binary)

Converts an STDF I*1 binary value to an ATDF signed one byte value.

\$signedShort = S2A_I2(\$binary)

Converts an STDF I*2 binary value to an ATDF signed two byte (i.e., short) value.

\$sint = S2A_I4(\$binary)

Converts an STDF I*4 binary value to an ATDF signed four byte (i.e., long) value.

\$hex = S2A_N1(\$binary)

Converts an STDF N*1 binary value to an ATDF hexadecimal value.

\$hexnum = S2A_Nn(\$binary)

\$real = S2A_R4(\$binary)

Converts and STDF R*4 binary value to an ATDF real value.

\$real = S2A_R8(\$binary)

Converts and STDF R*8 binary value to an ATDF real value.

\$unsignedByte = S2A_U1(\$binary)

Converts an STDF U*1 binary value to an ATDF unsigned one byte value.

\$unsignedShort = S2A_U2(\$binary)

Converts an STDF U*2 binary value to an ATDF unsigned two byte (i.e., short) value.

\$uint = S2A_U4(\$binary)

Converts an STDF U*4 binary value to an ATDF unsigned four byte (i.e., long) value.

S2A_Vn()

@txtdat = S2A_xCn(\$x, \$bindat)

Converts an STDF binary value containing one or more STDF C*n data type values to an array of ATDF values.

@value = S2A_xTYPE(\$x, \$dataType, \$bindat)

Converts an STDF binary value containing one or more STDF data type values to an array of ATDF values.

NOTE: DO NOT USE FOR *n DATA TYPES!

DEBUG SUBROUTINES

debugData(\@varRefs, \@varNames)

debugDataBin(\$datnam, \$bindat)

Outputs a binary value to standard output formatted as:

```
VARIABLE_NAME =
INDEX BINARY HEX DECIMAL CHARACTER
```

For example:

If the variable \$fabName contains the binary then ...

```
$fabName =
0000 00000101 05 5    α
0001 01001101 4D 77  M
0002 01001111 4F 79  O
0003 01010011 53 83  S
0004 00101101 2D 45  -
0005 00110100 34 52  4
```

debugDataPurdy(\@varRefs, \@varNames)

Outputs one or more variables to standard output in ``purdy" (i.e., pretty) format.

- Perl core and module documentation.
 - STDF Specification V4 published by Teradyne, Inc.
 - ATDF Specification V2 published by Teradyne, Inc.
-

AUTHOR

Michael Hackerott, michael.hackerott@mrhackerott.org

COPYRIGHT

Copyright © 2004-2005 Michael Hackerott. All rights reserved.

This program is free software; you can redistribute it and/or modify it under the terms of either the GNU General Public License or the Artistic License as specified in the Perl README file.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

ACKNOWLEDGEMENT

The Standard Test Data Format (STDF) and ASCII Test Data Format (ATDF) specifications are the original works of Teradyne Inc.

An Essay on Endian Order, Dr. William T. Verts, April 19, 1996. <http://www.cs.umass.edu/~verts/cs32/ endian.html>

KNOWN BUGS

- S2A_Bn and S2A_Vn is not implemented.

HISTORY

1.0.0 (200501221020) Michael Hackerott

- Created module.

1.0.1 (200402080725) Michael Hackerott

- Added `about ()` subroutine.

1.1.0 (200512012020) Michael Hackerott

- Fixed endian logic.

1.1.1 (200512030659) Michael Hackerott

- Updated documentation.