.

SRecord

Reference Manual

Scott Finneran <scottfinneran@yahoo.com.au>
Peter Miller <pmiller@opensource.org.au>

.

This document describes SRecord version 1.64
and was prepared 2 February 2017.

This document describing the SRecord program, and the SRecord program itself, are
Copyright 2014 Scott Finneran <scottfinneran@yahoo.com.au>
Copyright Peter Miller <pmiller@opensource.org.au>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

NAME

SRecord − manipulate EPROM load files

DESCRIPTION

The *SRecord* package is a collection of powerful tools for manipulating EPROM load files.

I wrote SRecord because when I was looking for programs to manipulate EPROM load files, I could not find very many. The ones that I could find only did a few of the things I needed. SRecord is written in C++ and polymorphism is used to provide the file format flexibility and arbitrary filter chaining. Adding more file formats and filters is relatively simple.

The File Formats

The SRecord package understands a number of file formats:

Ascii-Hex
: The ascii-hex format is understood for both reading and writing. (Also known as the ascii-space-hex format.)

ASM
: It is possible, for output only, to produce a serices of DB statements containing the data. This can be useful for embedding data into assembler programs. This format cannot be read.

Atmel Generic
: This format is produced by the Atmel AVR assembler. It is understood for both reading and writing.

BASIC
: It is possible, for output only, to produce a serices of DATA statements containing the data. This can be useful for embedding data into BASIC programs. This format cannot be read.

Binary
: Binary files can both be read and written.

B-Record
: Files in Freescale Dragonball bootstrap b-record format can be read and written.

C
: It is also possible to write a C array declaration which contains the data. This can be useful when you want to embed download data into C programs. This format cannot be read.

COE
: The Xilinx Coefficient File Format (.coe) is understood for output only.

Cosmac
: The RCA Cosmac Elf format is understood for both reading and writing.

DEC Binary
: The DEC Binary (XXDP) format is understood for both reading and writing.

Elektor Monitor (EMON52)
: The EMON52 format is understood for both reading and writing.

Fairchild Fairbug
: The Fairchild Fairbug format is understood for both reading and writing.

Formatted Binary
: The Formatted Binary format is understood for both reading and writing.

Four Packed Code (FPC)
: The FPC format is understood for both reading and writing.

Hexdump
: It is possible to get a simple hexdump as output.

IDT/sim
: The IDT/sim binary file format is understood for both reading and writing.

Intel
: The Intel hexadecimal format is understood for both reading and writing. (Also known as the Intel MCS-86 Object format.)

Intel AOMF
: The Intel Absolute Object Module Format (AOMF) is understood for both reading and writing.

Intel 16
: The Intel hexadecimal 16 format is understood for both reading and writing. (Also known as the INHX16 file format.)

LSI Logic Fast Load
: The LSI Logic Fast Load format is understood for both reading and writing. Logisim The Logisim format is understood for both reading and writing. See *srec_logisim*(5) for more information.

Memory Initialization Format
: The Memory Initialization Format (.mem) by Lattice Semiconductor is understood for writing only.

MIF
: The Memory Initialization File format by Altera is supported for both reading and writing.

MOS Technology
: The MOS Technology hexadecimal format is understood for both reading and writing.

MIPS-Flash
: The MIPS Flash file format is supported for both reading and writing.

Motorola S-Record
: The Motorola hexadecimal S-Record format is understood for both reading and writing. (Also known as the Exorciser, Exormacs or Exormax format.)

MsBin
: The Windows CE Binary Image Data Format is supported both for reading and writing.

Needham
: The Needham Electronics ASCII file format is understood for both reading and writing.

OS65V
: The Ohio Scientific hexadecimal format is understood for both reading and writing.

PPB
: The Stag Prom Programmer binary format is understood for both reading and writing.

PPX
: The Stag Prom Programmer hexadecimal format is understood for both reading and writing.

Signetics
: The Signetics format is understood for both reading and writing.

SPASM
: The SPASM format is used by a variety of PIC programmers; it is understood for both reading and writing.

Spectrum
: The Spectrum format is understood for both reading and writing.

Tektronix (Extended)
: The Tektronix hexadecimal format and the Tektronix Extended hexadecimal format are both understood for both reading and writing.

Texas Instruments Tagged
: The Texas Instruments Tagged format is understood for both reading and writing (both 8 and 16 bit). Also known as the TI-tagged or TI-SDSMAC format.

Texas Instruments ti-txt
: The TI-TXT format is understood for reading and writing. This format is used with the bootstrap loader of the Texas Instruments MSP430 family of processors.

TRS-80
: The Radio Shack TRS-80 object file format is understood for reading and writing.

VHDL
: It is possible to write VHDL file. This is only supported for output.

Verilog VMEM
: It is possible to write a Verilog VMEM file suitable for loading with `$readmemh()`. This format is supported for reading and writing.

Wilson
: The Wilson format is understood for both reading and writing. This mystery format was added for a mysterious type of EPROM writer.

The Tools

The primary tools of the package are *srec_cat* and *srec_cmp*. All of the tools understand all of the file formats, and all of the filters.

srec_cat The *srec_cat* program may be used to catenate (join) EPROM load files, or portions of EPROM load files, together. Because it understands all of the input and output formats, it can also be used to convert files from one format to another.

srec_cmp
The *srec_cmp* program may be use to compare EPROM load files, or portions of EPROM load files, for equality.

srec_info
The *srec_info* program may be used to print summary information about EPROM load files.

The Filters

The *SRecord* package is made more powerful by the concept of *input filters*. Wherever an input file may be specified, filters may also be applied to that input file. The following filters are available:

bit reverse
The *bit-reverse* filter may be used to reverse the order of bits in each data byte.

byte swap
The *byte swap* filter may be used to swap pairs of add and even bytes.

CRC The various *crc* filters may be used to insert a CRC into the data.

checksum
The *checksum* filters may be used to insert a checksum into the data. Positive, negative and bit-not checksums are available, as well as big-endian and little-endian byte orders.

crop The *crop* filter may be used to isolate an input address range, or ranges, and discard the rest.

exclude The *exclude* filter may be used to exclude an input address range, or ranges, and keep the rest.

fill The *fill* filter may be used to fill any holes in the data with a nominated value.

length The *length* filter may be used to insert the data length into the data.

maximum
The *maximum* filter may be used to insert the maximum data address into the data.

minimum
The *minimum* filter may be used to insert the minimum data address into the data.

offset The *offset* filter may be used to offset the address of data records, both forwards and backwards.

random fill
The *random fill* filter may be used to fill holes in the data with random byte values.

split The *split* filter may be used to split EPROM images for wide data buses or other memory striping schemes.

unfill The *unfill* filter may be used to make holes in the data at bytes with a nominated value.

unsplit The *unsplit* filter may be reverse the effects of the split filter.

More than one filter may be applied to each input file. Different filters may be applied to each input file. All filters may be applied to all file formats.

ARCHIVE SITE

The latest version of *SRecord* is available on the Web from:

```
URL:    http://srecord.sourceforge.net/
File:   index.html              # the SRecord page
File:   srecord-1.64.README     # Description, from the tar file
File:   srecord-1.64.lsm        # Description, LSM format
```

```
File:   srecord−1.64.spec       # RedHat package specification
File:   srecord−1.64.tar.gz     # the complete source
File:   srecord−1.64.pdf        # Reference Manual
```

BUILDING SRECORD

Full instructions for building *SRecord* may be found in the *BUILDING* file included in this distribution.

It is also possible to build *SRecord* on Windows using the Cygwin (www.cygwin.com) or DJGPP (www.delorie.com/djgpp) environments. Instructions are in the *BUILDING* file, including how to get native Windows binaries.

COPYRIGHT

srecord version 1.64
Copyright © Scott Finneran <scottfinneran@yahoo.com.au>
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller <pmiller@opensource.org.au>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

It should be in the *LICENSE* file included with this distribution.

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

RELEASE NOTES

This section details the various features and bug fixes of the various releases. For excruciating and complete detail, and also credits for those of you who have generously sent me suggestions and bug reports, see the *etc/CHANGES.** files.

Version 1.64 (2014-Jun-22)

• Cleaned up a few references to the maintainer's name.

• Fixed some warnings in test 38

• Fixed bugs discovered by Mike <russiane39@gmail.com> regarding the formatting of some of the examples where lines were wrapping the wrong way.

• Added the ability to compile without libgcrypt if the user specifies the option --without-gcrypt to configure. If gcrypt is missing, the user is prompted to either install it or explicitly compile with it disabled.

• Cleaned up a few recent compiler and doxygen warnings.

• Added Sourceforge Patch #4 contributed by Stas Sergeev <stsp@sourceforge.net> which contributed get_upper_bound() and makes it and the lower equivalent public (for use from libsrecord). Also fixed a bug in srec_memory::find_next_chunk() where it would fail to reset the find_next_chunk_index once it became equal to nchunks. After that, find_next_chunk() would always fail.

• Added prefix and postfix strings to the C Array format which are applied at each end of the data array declaration. This allows the user to add various compiler directives such as location flags or non-standard load address specifiers.

• when converting from AOMF Fixed Sourceforge bug 11 raised by patryks. In the analysis, it was discovered that a start address (of zero) was being generated when converting from AOMF. AOMF does not support the concept of an execution start address.

• The Windows build instructions have been updated by Jens Heilig <jens@familie-heilig.net>.

• Fixed the length field description in extended tektronix documentation. The implementation was already correct.

• Added a regression test for calculated address for CRC (Sourceforge bug 19).

Version 1.63 (2014-Apr-07)

• The srec_tools now understand how to read and write Logisim formt.

• Daniel Anselmi <danselmi@gmx.ch> fixed a bug with generating Lattice Memory init files.

• This change set generalises the code that handles redundant byte settings and contradictory byte settings. The defcon_t type describes what to do: ignore, warn or error. The −−multiple option is no more.

• Daniel Anselmi <danselmi@gmx.ch> discovered that the "mem" output format was malformed. The lines now break every "width" bits.

• Hanspeter Niederstrasser disoverd some false negatives in the test suite, when used on OSX. Thank you for the bug report. This has been fixed.

• Simplification of selection code to choose which CRC16 implmentation is used.

• Juliano MourÃÂ£o Vieira <juliano@utfpr.edu.br> discovered a problem with the srec_mif.5 description. This mistake is not present in the code.

• Liju Prasanth Nivas (RBEI/ECA1) <Liju.PrasanthNivas@in.bosch.com> suggsted another use case for the examples. In the case of "joining" files that are meant to be “stacked in layers”. Contributions for the examples are always appreciated. Thank you.

• Added more comments to the code, so that the use of URL_deode for the command line is better explained. And comment to explain why not to do turl_encode when building header records.

• Added URL quoting to the command line. There are times you need to be able to insert unprintable characters on the command line. The immediate use case prompting this was a user wanting to put a NUL in the header string. So now you can, as "%00". The other choice was quoted printable encoding, but that was a bit obscure.

• Fixed new warnings when building with g++ 4.8.1

• Added more links to the windows files on SourceForge, maybe it will boost download numbers.

Version 1.62 (2013-Jun-05)

- Luc Steynen <LucSteynen@edna.be> discovered that the −hecksum-big-endian opion was a counter-intuitive alias for the the −checksum-bitnot-big-endian option. The −checksum-big-endian option is now deprecated, in favor of the −checksum-bitnot-big-endian option; the code will warn uers of the old option they will need to change. Ditto little-endian variants
- Alin Pilkington <apilkington@moog.com> found that the Tektronics Extended format was calculating the record length incorrectory. Thanks you for the bug report. This has been fixed for both reading and writing.
- Dr. Benedikt Schmitt <Benedikt.Schmitt@safeintrain.de> suggested being able to inject arbitrary data into the file header (such as NUL termination characters). This change set adds URL-style escapes (e.g. %25) to the string on he command line. For example: −header or −generate −string

Version 1.61 (2013-Jan-04)

- Izzet Ozcelik <izzetozcelik@cscope.co.uk> discovered a bug in the Tektronix-Extenden format line checksum calculations. The comparison should have been in 8 bits, not int.
- Daniel Anselmi <danselmi@gmx.ch> contributed a Memory Initialization Format by Lattice Semiconductor, for output only.
- Daniel Anselmi <danselmi@gmx.ch> contributed a Xilinx Coefficient File Format (.coe) output class.

Version 1.60 (2012-May-19)

• There are now several additional CRC-16 polynomials, plus the ability to select a polynomial by name, rather than by value. See http://en.wikipedia.org/wiki/Cyclic_redundancy_check for a table of names and values.

Version 1.59 (2012-Feb-10)

- A number of additional CRC-16 polynomials have been added, as well as the ability to select a polynomial by name, rather than by value. See *srec_input*(1) for more information.

Version 1.58 (2011-Dec-18)

- The **−guess** command line option, for guessing the file format, now also tells you the command line option you could have used instead of **−guess** for the exact format.
- The Intergated Device Technology (IDT) system integration manager (IDT/sim) binary format is now understood for both reading and writing.
- The Stag Prom Programmer binary format is now supported for both reading and writing.
- The Stag Prom Programer hexadecimal format is now understood for both reading and writing.
- The MIPS-Flash fiel format is now supported for both reading and writing.
- Bernhard Weirich `<Bernhard.Weirich@riedel.net>` discovered that a backward compatible option had been omitted when the **−INtel_16** option was renamed **−INtel_HeX_16** to more closely match the usual abbreviation (INHX16) for this format. The backwards compatible option name has been reintroduced.
- The windows build instructions have been greatly imptoved, based on the experiences of Jens Heilig `<jens@familie-heilig.net>` which he has generously shared.

- The documentation in the manual about sequence warnings has been improved. The **−disable-sequence-warnings** option must come before the input file on the command line. My thanks to Emil Gracic <emil_kruki@yahoo.com> for reporting this problem.

Version 1.57 (2011-Jun-09)

- The byte order of the fletcher16 output has been reversed.
- The meaning of the **−address-length** option has been change for the Intel output format. Previously, 2 meant using i16hex 20-bit segmented addressing, and >2 meant using i32hex extended addressing. This has been changed: a value of 2 requests i8hex 16-bit addressing, a value of 3 requests i16hex 20-bit segment addressing, and a value >=4 requests i32hex 32-bit addressing. My thanks to Stephen R. Phillips <srp@CSECorporation.com> for reporting the absence of i8hex support.
- The **−generate −repeat-string** option is now able to take a string that looks like a number as the text to be repeated. My thanks to Stephen R. Phillips <srp@CSECorporation.com> for reporting this problem.
- Luca Giancristofaro <luca.giancristofaro@prosa.com> discovered a WinAVR linker that is a sandwich short of a picnic: it generated non-conforming Intel hex end-of-file records. This is no longer an error, but only a warning.
- There were some problems with the RPM spec file, these have been improved. My thanks to Galen Seitz <galens@seitzassoc.com> for reporting this problem.

Version 1.56 (2010-Sep-15)

- A bug has been fixed in the MsBin output, it now concatenates records correctly, and calaulate checksums appropriately.
- It is now possible to ask the Fletcher 16 filter to give you a specific answer, and adjusting the checksum to achieve that result. It is also possible to specify different seed values for the sums.
- There is a new srec_cat **−enable=optional-address** option to cause output formats capable of omitting addresses, to omit a leading zero address, as those formats usually default the address to zero, if no address information is seen before the first data record. Defaults to false (disabled).
- There is a new *srec_cat*(1) **−output-block-packing** option, that may be used to pack output records even when they cross internal SRecord boundaries.
- There is a new *srec_cat*(1) **−output-block-size** so that you can specify the block size exactly, rather than implying it with the line length option.

Version 1.55 (2010-Feb-10)

- The Makefile.in has been improved, it now copes with non-standard `--prefix` options.
- The rpm.spec file has been improved, it now separates the commands, shared libraries and development files.

Version 1.54 (2010-Jan-29)

- There is now a shared library installed, including the necessary header files so that you can use all of the file formats and filters in your own projects.
- The license on the shared library code is GNU Lesser General Public License, version 3.0 or later.
- The code can cope with older versions of GNU Libgcrypt. In the case of very old versions, by ignoring it.
- A number of build problems have been fixed.

Version 1.53 (2009-Nov-10)

- There is a new MsBin (Windows CE Binary Image Data) file format, supported for both reading and writing.

- The lintian(1) warning about hyphen in the manual pages has been silenced, by careful use of −, - and – as appropriate. Sure makes some of the sources ugly, tho. The lintian(1) warning about the undefined .XX macro has been silenced, by making it conditional.
- The code will build without libgcrypt.

Version 1.52 (2009-Sep-17)

- There is a new *srec_cat −generator −l-e-constant* data generator (and also *−b-e-const*) that may be used to insert multi-byte constants into your data. See *srec_input*(1) for more information.

Version 1.51 (2009-Sep-13)

- A number of gcc 4.4 build problems have been fixed.
- A bugs has been fixed in the Intel output format. When using the segemented format (address-length=2) records that span the end of segment boundary are tricky. The code now carefully splits such output records, to ensure the two parts are explicitly placed into separate segments.

Version 1.50 (2009-Jul-09)

- The CRC16 code has been enhanced to provide low-to-high bit order, in addition to the previous high-to-low bit order. It is also possible to specify the polynomial, with the default the CCITT standard polynomial, as was in the previous code. See *srec_input*(1) for more information.
- The MD5, RipeMD-160, SHA1, SHA224, SHA256, SHA384, SHA 512 and Whirlpool message digests are now supported. See *srec_input*(1) for more information.
- There is a new *srec_cat −bit-reverse* filter, that may be used to reverse the bits in each data byte. See *srec_input*(1) for more information.

Version 1.49 (2009-May-17)

- A typo in the srec_input(1) man page has been fixed.

Version 1.48 (2009-Apr-19)

- There are new Fletcher Checksum filters, both 32-bits and 16-bits, both little-endian and big-endian.
- There are new Adler Checksum filters, both 32-bits and 16-bits, both little-endian and big-endian.

Version 1.47 (2009-Feb-19)

- Memory Initialzation File (MIF) format by Altera is now supported for reading and writing.

Version 1.46 (2009-Jan-13)

- There is a new option for the *−−x*-e-length filters, they can now accept a width, and this is divided into the byte lenght, so that you can insert the length in units of words (2) or longs (4).
- Some small corrections have been made to the documentation.
- The −minimum and −maximum options have been renamed −minimum-address and −maximum-address, to avoid a command line grammar syntax problem.

Version 1.45 (2008-Sep-30)

- A bug has been fixed in the *srec_cat*(1) command. You are now able to specify several inputs within parentheses, instead of just one. This allows filters to be applied to the concatenation of several inputs.
- The *srec_cat*(1) command is now able to write FORTH output.

Version 1.44 (2008-Aug-29)

- Some compilers issue a warning when const appears before extern. "warning: storage class is not first". The C output has been updated to conform to this expectation.
- The manual page for *srec_cat*(1) has been enhanced to describe the in-memory data model, and the resulting output data order.
- The **−motorola** optional width argument now produces a better error message when it is out of range.

- The **−fill** filter now checks the size, and fails for absurdly large fills, with a **−big** override if they really want >1GB fills.
- A bug in the .spec file for rpmbuild has been fixed, it now takes notice of $RPM_BUILD_ROOT
- There is a new **−line-termination** option, which may be used to select the desired line termination of output text files.

Version 1.43 (2008-Jul-06)

- The *srec-cat −data-only* option has been broken down into four separate controls. It is now possible to **−enable** and **−disable** individual features, such as “header”, “data-count”, “execution-start-address” and “footer”. See *srec_cat*(1) for more information.
- The *srec_cat −start-address* option has been renamed **−execution-start-address** to remove any confusion with the **−offset** filter. The documentation now explicitly explains the difference between the two.
- Examples of converting to and from binary files have been added to the *srec_examples*(1) man page.
- A bug has been fixed in the MOS Tech format, it now emits an end record even when there is no execution start address passed in.

Version 1.42 (2008-Jun-01)

- The MOS Technology format was not reading and writing end records correctly, this has been fixed. The name of the company has been corrected.
- Some examples of how to insert constant or scripted data into your EPROM load files have been added to the *srec_examples*(1) man page.

Version 1.41 (2008-May-12)

- False negative being reported by tests on Cygwin have been fixed.
- There are six new filters (−be-exclusive-length, −le-exclusive-length, −be-exclusive-maximum, −le-exclusive-maximum, −be-exclusive-minimum and −le-exclusive-minimum) which are very similar to their non-exclusive equivalents, except that they do not include the adress range covered by their output in their output.
- A bug has been fixed in the C word-array output. It was getting offsets and lengths wrong in some cases.
- A bug has been fixed in the generated C array header file, it no longer omits the section descriptor arrays.
- A problem with building RPM packages with the names of the executables in the .spec file has been fixed, and the BuildRequires has been updated.

Version 1.40 (2008-Mar-13)

- An RPM build problem has been fixed.
- The dependency on the Boost library is now documented in the BUILDING file.
- Some build problems with g++ 4.3 have been fixed
- A bug has been fixed in the calculation of ranges on the command line, it no longer goes into an infinite loop for "−fill 0xFF −over { foo.hex −exclude −within foo.hex }" construct, which should have been calculating an empty fill set, but was instead calculating a 4GB fill set.
- The CRC32 filters now take an −xmodem option, to use an xmodem-like (all bit zero) initial state, rather than the default CCITT (all bits on) initial state.

Version 1.39 (2008-Feb-04)

- A bug has been fixed in the use of parentheses to group filters and override the default precedences.

Version 1.38 (2008-Jan-14)

- The CRC16 filters now support a −Broken option, to perform a common-but-broken CRC16 calculation, in addition to the CCITT and XMODEM calculations.
- A link has been added to the CRC16 man page section to the www.joegeluso.com/software/articles/ccitt.htm web page, to explain the difficulties in seeding CRC16 calculations.
- A buglet has been fixed in the *srec_motorola*(5) man page, it now includes `S6` in the list of things that can appear in the type field.
- The ability to negate expressions is now mentioned in the *srec_examples*(1) man page.

Version 1.37 (2007-Oct-29)

- It is now possible to have negative expressions on the command line, to facilitate "−−offset −−minimum foo" usages.
- The *srec_cat*(1) command now has a simple hexadecimal dump output format.
- The use of *uudecode*(1) in the tests has been removed, so *sharutils* is no longer a build dependency.

Version 1.36 (2007-Aug-07)

- A bug has been fixed in the CRC-16 CCITT calculation; the algorithm was correct but the start value was incorrect, leading to incorrect results.
- The CRC16 filters have a new −−no-augment option, to omit the 16 zero bits augmenting the message. This is not CCITT standard conforming, but some implementations do this.
- A problem has been fixed in the generated Makefile.in file found in the tarball.
- The license has been changed to GNU GPL version 3.

Version 1.35 (2007-Jun-23)

- A major build problem with the generated makefile has been fixed.

Version 1.34 (2007-Jun-22)

- The C and ASM output formats have been improved in the word mode.
- Several build problems have been fixed.

Version 1.33 (2007-May-18)

- More examples have been added to the documentation.
- It is now possible to perform set intersection and set difference on address ranges on the command line.
- There is a new category of data source: generators. You can generate constant data, random data and repeating data.
- The assembler and C-Array outputs now support additional options to facilitate MSP430 systems. They can also optionally write shorts rather than bytes.
- You can now round address ranges on the command line to be whole multiples of a number of bytes.

Version 1.32 (2007-Apr-24)

- The TI-TXT format output has been improved; it is less spec conforming but more reality conforming. It now allows odd alignment without padding. It also ends with a `q` instead of a `Q`.
- The warning for odd input addresses has been dropped. The spec didn't like them, but the MSP430 handles them without a hiccup.

Version 1.31 (2007-Apr-03)

- The Verilog format now suppresses comments when you specify the −−data-only option.
- The Texas Instruments ti-txt (MSP430) format is now understood for reading and writing.

Version 1.30 (2007-Mar-21)

- The ascii-hex output format has been improved.
- The ti-tagged 16-bit format is now understood for reading and writing.
- The Intel format no longer warns about missing optional records.
- A bug in the ti-tagged format has been fixed, it now understands the '0' tag.

Version 1.29 (2007-Mar-13)

- A serious bug has been fixed in the generated Makefile.

Version 1.28 (2007-Mar-08)

- It is now possible to read and write files in the Freescale MC68EZ328 Dragonball bootstrap b-record format

Version 1.27 (2006-Dec-21)

- [SourceForge Feature Request 1597637] There is a new warning issued when input data records are not in strictly ascending address order. There is a new command line option to silence the warning.
- [SourceForge Feature Request 1592348] The command line processing of all srecord commands now understands *@file* command line options, filled with additional space separated strings witch will be treated as of they were command line options. This gets around absurdly short command line length limits in some operating systems.

Version 1.26 (2006-May-26)

- It is now possible to place parentheses on the command line in more places to clarify your intent.
- This change prepares SRecord for the next public release.

Version 1.25 (2006-May-18)

- The assembler output has been enhanced to produce ORG directives, if necessary, to change the data address.
- The *srec_cat*(1) command now only writes an execution start address into the output if there was an execution start address present in the input.

Version 1.24 (2006-Mar-08)

- Additional information has been added to the lseek error when they try to seek to addresses >= 2**31
- The CRC 16 filters have been enhanced to accept an argument to specify whether CCITT or XMODEM calculations are to be performed.

Version 1.23 (2005-Sep-23)

- A segfault has been fixed on x86_64 when running the regression test suite.
- A compile problem with the lib/srec/output/file/c.cc file has been fixed.

Version 1.22 (2005-Aug-12)

- The **−byte-swap** filter now has an optional *width* argument, to specify the address width to swap. The default is two bytes.
- The motorola file format now accepts an additional 'width' command line argument, so you can have 16-bit and 32-bit address multiples.
- A bug has been fixed in the VMEM output format. It was failing to correctly set the next address in some cases. This fixes SourceForge bug 1119786.
- The −C-Array output format now uses the `const` keyword by default, you can turn it off with the −no-const option. The −C-Array output format can now generate an additional include file if you use the −INClude option. This answers SourceForge feature request 942132.

- A fix for the "undefined symbols" problem when using g++ 3.x on Cygwin and MacOsX has been added to the ./configure script.
- There is a new −ignore-checksum command line option. The −ignore-checksums option may be used to disable checksum validation of input files, for those formats which have checksums at all. Note that the checksum values are still read in and parsed (so it is still an error if they are missing) but their values are not checked.

Version 1.21 (2005-Feb-07)

- More Doxygen comments have been added to the class header files.
- There is a new *srec_cat −−crlf* option, which may be used for force CRLF output on operating systems which don't use that style of line termination.
- A number of problems with GCC, particularly with the early 3.*x* series.
- There is a new "Stewie" format, an undocumented format loosely based on the Motorola S-Record format, apparently used in mobile phones. More information would be most welcome.
- A number of build problems have been fixed.

Version 1.20 (2004-Feb-08)

- The AOMF format now accepts (and ignores) more record types.

Version 1.19 (2004-Jan-03)

- It is now possible to set the execution start address in the output using the *srec_cat −Execution_Start_Address* command line option.
- The Intel Absolute Object Module Format (AOMF) is now supported for reading and writing.
- There is a new *srec_cat −Random_Fill* filter, like the *srec_cat −Fill* filter except that it uses random values.

Version 1.18 (2004-Jan-01)

- The VMEM format is now able to output data for 64 and 128 bits wide memories.
- A bug in the SRecord reference manuals has been fixed; the CRCxx had a copy-and-paste glitch and always said big-endian where little endian was intended half the time.

Version 1.17 (2003-Oct-12)

- There is now support for Intel Extended Segment addressing output, via the −−address-length=2 option.
- There is now support for output of Verilog VMEM format. See *srec_vmem*(5) for more information.
- There is now support for reading and writing the INHX16 format, used in various PIC programmers. It looks just like the Intel Hex format, except that the bytes counts and the addresses refer to words (hi,lo) rather than bytes. See *srec_intel16*(5) for more information.

Version 1.16 (2003-Jul-28)

- Some updates have been made to cope with GCC 3.2

Version 1.15 (2003-Jun-16)

- The ASCII-Hex implementation is now slightly more complete. I still haven't found a definitive description.
- The Fairchild Fairbug format has been added for reading and writing. See *srec_fairchild*(5) for more information.
- The Spectrum format has been added for reading and writing. See *srec_spectrum*(5) for more information.
- The Formatted Binary format has been added for reading and writing. See *srec_formatted_binary*(5) for more information.

- The RCA Cosmac Elf format has been added for reading and writing. See *srec_cosmac*(5) for more information.
- The Needham EMP programmer format has been added for reading and writing. See *srec_needham*(5) for more information.

Version 1.14 (2003-Mar-11)

- Numerous fixes have been made to header handling. It is now possible to specify an empty header with the `-header` command line option.
- Some more GCC 3.2 build problems have been fixed.

Version 1.13 (2003-Feb-05)

- Bugs have been fixed in the Texas Instruments Tagged and VHDL formats, which produced inconsistent output.
- A couple of build problems have been fixed.
- There are two new output formats for ASM and BASIC.

Version 1.12 (2002-Dec-06)

- It is now possible to put **-minimum** *input.spec* (also **-maximum** and **-length**) almost anywhere on the command line that you can put a number. It allows, for example, the -offset value to be calculated from the maximum of the previous file. The values calculated by **-Minimum**, **-Maximum** and **-Length** may also be rounded to arbitrary boundaries, using **-Round_Down**, **-Round_Nearest** and **-Round_Up**.
- The malformed Motorola S5 records output by the Green Hills tool chain are now understood.

Version 1.11 (2002-Oct-21)

- The Ohio Scientific OS65V audio tape format has been added for reading and writing. See *srec_os65v*(5) for more information.
- Some build problems have been fixed.

Version 1.10 (2002-Jun-14)

- The Intel format now emits the redundant extended linear address record at the start of the file; some loaders couldn't cope without it.
- The Binary format now copes with writing to pipes.
- The Motorola format now understands the S6 (24-bit data record count) records for reading and writing.
- The DEC Binary format now works correctly on Windows machines.
- The LSI Logic Fast Load format is now understood for both reading and writing. See *srec_fastload*(5) for more information.

Version 1.9 (2001-Nov-27)

- The DEC Binary (XXDP) format is now understood for both reading and writing. See *srec_dec_binary*(5) for more information.
- The Elektor Monitor (EMON52) format is now understood for both reading and writing. See *srec_emon52*(5) for more information.
- The Signetics format is now understood for both reading and writing. See *srec_signetics*(5) for more information.
- The Four Packed Code (FPC) format is now understood for both reading and writing. See *srec_fpc*(5) for more information.
- Wherever possible, header data is now passed through by *srec_cat*(1). There is also a new *srec_cat -header* option, so that you can set the header comment from the command line.
- The Atmel Generic format for Atmel AVR programmers is now understood for both reading and writing. See *srec_atmel_generic*(5) for more information.

- The handling of termination records has been improved. It caused problems for a number of filters, including the −fill filter.
- A bug has been fixed in the checksum calculations for the Tektronix format.
- There is a new SPASM format for PIC programmers. See *srec_spasm*(5) for more information.

Version 1.8 (2001-Apr-20)

- There is a new “unfill” filter, which may be used to perform the reverse effect of the “fill” filter.
- There is a new bit-wise NOT filter, which may be used to invert the data.
- A couple of bugs have been fixed in the CRC filters.

Version 1.7 (2001-Mar-19)

- The documentation is now in PDF format. This was in order to make it more accessible to a wider range of people.
- There is a new *srec_cat* −−*address-length* option, so that you can set the length of the address fields in the output file. For example, if you always want S3 data records in a Motorola hex file, use the `--address-length=4` option. This helps when talking to brain-dead EPROM programmers which do not fully implement the format specification.
- There is a new −−*multiple* option to the commands, which permits an input file to contain multiple (contradictory) values for some memory locations. The last value in the file will be used.
- A problem has been fixed which stopped SRecord from building under Cygwin.
- A bug has been fixed in the C array output. It used to generate invalid output when the input had holes in the data.

Version 1.6 (2000-Dec-03)

- A bug has been fixed in the C array output. (Holes in the input caused an invalid C file to be produced.)
- There is are new CRC input filters, both 16-bit and 32-bit, both big and little endian. See *srec_cat*(1) for more information.
- There is a new VHDL output format.
- There are new checksum filters: in addition to the existing one’s complement (bit not) checksum filter, there are now negative and positive checksum filters. See *srec_cat*(1) for more information.
- The checksum filters are now able to sum over 16-bit and 32-bit values, in addition to the existing byte sums.
- The *srec_cmp* program now has a −−**verbose** option, which gives more information about how the two inputs differ. See *srec_cmp*(1) for more information.

Version 1.5 (2000-Mar-06)

- There is now a command line option to guess the input file format; all of the tools understand this option.
- The “MOS Technologies” file format is now understood for reading and writing. See *srec_mos_tech*(5) for more information.
- The “Tektronix Extended” file format is now understood for reading and writing. See *srec_tektronix_extended*(5) for more information.
- The “Texas Instruments Tagged” file format is now understood for reading and writing. (Also known as the TI-Tagged or SDSMAC format.) See *srec_ti_tagged*(5) for more information.
- The “ascii-hex” file format is now understood for reading and writing. (Also known as the ascii-space-hex format.) See *srec_ascii_hex*(5) for more information.

- There is a new *byte swap* input filter, allowing pairs of odd and even input bytes to be swapped. See *srec_cat*(1) for more information.
- The “wilson” file format is now understood for reading and writing. This mystery format was added for a mysterious type of EPROM writer. See *srec_wilson*(5) for more information.
- The *srec_cat* program now has a **−data-only** option, which suppresses all output except for the data records. This helps when talking to brain-dead EPROM programmers which barf at anything but data. See *srec_cat*(1) for more information.
- There is a new *−Line-Length* option for the *srec_cat* program, allowing you to specify the maximum width of output lines. See *srec_cat*(1) for more information.

Version 1.4 (2000-Jan-13)

- SRecord can now cope with CRLF sequences in Unix files. This was unfortunately common where the file was generated on a PC, but SRecord was being used on Unix.

Version 1.3 (1999-May-12)

- A bug has been fixed which would cause the crop and exclude filters to dump core sometimes.
- A bug has been fixed where binary files were handled incorrectly on Windows NT (actually, any system in which text files aren’t the same as binary files).
- There are three new data filters. The −−OR filter, which may be used to bit-wise OR a value to each data byte; the −−AND filter, which may be used to bit-wise AND a value to each data byte; and the −−eXclusive-OR filter, which may be used to bit-wise XOR a value to each data byte. See *srec_cat*(1) for more information.

Version 1.2 (1998-Nov-04)

- This release includes file format man pages. The web page also includes a PostScript reference manual, containing all of the man pages.
- The Intel hex format now has full 32-bit support. See *srec_intel*(5) for more information.
- The Tektronix hex format is now supported (only the 16-bit version, Extended Tektronix hex is not yet supported). See *srec_tektronix*(5) for more information.
- There is a new *split* filter, useful for wide data buses and memory striping, and a complementary *unsplit* filter to reverse it. See *srec_cat*(1) for more information.

Version 1.1 (1998-Mar-22)

First public release.

NAME

How to build SRecord

SPACE REQUIREMENTS

You will need about 3MB to unpack and build the *SRecord* package. Your milage may vary.

BEFORE YOU START

There are a few pieces of software you may want to fetch and install before you proceed with your installation of SRecord.

Boost Library

> You will need the C++ Boost Library. If you are using a package based system, you will need the libboost-devel package, or one named something very similar.
> http://boost.org/

Libgcrypt Library

> You will need the GNU Crypt library. If you are using a package based system, you will need the libgcrypt-devel package, or one named something very similar.
> http://directory.fsf.org/project/libgcrypt/

GNU Libtool

> You will need the GNU Libtool software, used to build shared libraries on a variety of systems.
> http://www.gnu.org/software/libtool/

GNU Groff

> The documentation for the *SRecord* package was prepared using the GNU Groff package (version 1.14 or later). This distribution includes full documentation, which may be processed into PostScript or DVI files at install time − if GNU Groff has been installed.

GCC You may also want to consider fetching and installing the GNU C Compiler if you have not done so already. This is not essential. SRecord was developed using the GNU C++ compiler, and the GNU C++ libraries.

The GNU FTP archives may be found at `ftp.gnu.org`, and are mirrored around the world.

SITE CONFIGURATION

The **SRecord** package is configured using the *configure* program included in this distribution.

The *configure* shell script attempts to guess correct values for various system-dependent variables used during compilation, and creates the *Makefile* and *lib/config.h* files. It also creates a shell script *config.status* that you can run in the future to recreate the current configuration.

Normally, you just *cd* to the directory containing *SRecord*'s source code and then type

```
% ./configure
...lots of output...
%
```

If you're using *csh* on an old version of System V, you might need to type

```
% sh configure
...lots of output...
%
```

instead to prevent *csh* from trying to execute *configure* itself.

Running *configure* takes a minute or two. While it is running, it prints some messages that tell what it is doing. If you don't want to see the messages, run *configure* using the quiet option; for example,

```
% ./configure --quiet
%
```

To compile the **SRecord** package in a different directory from the one containing the source code, you must use a version of *make* that supports the *VPATH variable*, such as *GNU make*. *cd* to the directory where you want the object files and executables to go and run the *configure* script. *configure* automatically checks for the source code in the directory that *configure* is in and in .. (the parent directory). If for some reason *configure* is not in the source code directory that you are configuring, then it will report that it can't find the

source code. In that case, run *configure* with the option `--srcdir=`*DIR*, where *DIR* is the directory that contains the source code.

By default, *configure* will arrange for the *make install* command to install the **SRecord** package's files in */usr/local/bin*, and */usr/local/man*. There are options which allow you to control the placement of these files.

`--prefix=`*PATH*
: This specifies the path prefix to be used in the installation. Defaults to */usr/local* unless otherwise specified.

`--exec-prefix=`*PATH*
: You can specify separate installation prefixes for architecture-specific files files. Defaults to *${prefix}* unless otherwise specified.

`--bindir=`*PATH*
: This directory contains executable programs. On a network, this directory may be shared between machines with identical hardware and operating systems; it may be mounted read-only. Defaults to *${exec_prefix}/bin* unless otherwise specified.

`--mandir=`*PATH*
: This directory contains the on-line manual entries. On a network, this directory may be shared between all machines; it may be mounted read-only. Defaults to *${prefix}/man* unless otherwise specified.

configure ignores most other arguments that you give it; use the `--help` option for a complete list.

On systems that require unusual options for compilation or linking that the *SRecord* package's *configure* script does not know about, you can give *configure* initial values for variables by setting them in the environment. In Bourne-compatible shells, you can do that on the command line like this:

```
$ CXX='g++ -traditional' LIBS=-lposix ./configure
...lots of output...
$
```

Here are the *make* variables that you might want to override with environment variables when running *configure*.

Variable: CXX
: C++ compiler program. The default is *c++*.

Variable: CPPFLAGS
: Preprocessor flags, commonly defines and include search paths. Defaults to empty. It is common to use `CPPFLAGS=-I/usr/local/include` to access other installed packages.

Variable: INSTALL
: Program to use to install files. The default is *install* if you have it, *cp* otherwise.

Variable: LIBS
: Libraries to link with, in the form `-l`*foo* `-l`*bar*. The *configure* script will append to this, rather than replace it. It is common to use `LIBS=-L/usr/local/lib` to access other installed packages.

If you need to do unusual things to compile the package, the author encourages you to figure out how *configure* could check whether to do them, and mail diffs or instructions to the author so that they can be included in the next release.

BUILDING SRECORD

All you should need to do is use the

```
% make
...lots of output...
%
```

command and wait. When this finishes you should see a directory called *bin* containing three files: *srec_cat*, *srec_cmp* and *srec_info*.

srec_cat *srec_cat* program is used to manipulate and convert EPROM load files. For more information, see *srec_cat*(1).

srec_cmp
The *srec_cmp* program is used to compare EPROM load files. For more information, see *srec_cmp*(1).

srec_info
The *srec_info* program is used to print information about EPROM load files. For more information, see *srec_info*(1).

If you have GNU Groff installed, the build will also create a *etc/reference.ps* file. This contains the README file, this BUILDING file, and all of the man pages.

You can remove the program binaries and object files from the source directory by using the

```
% make clean
...lots of output...
%
```

command. To remove all of the above files, and also remove the *Makefile* and *lib/config.h* and *config.status* files, use the

```
% make distclean
...lots of output...
%
```

command.

The file *etc/configure.in* is used to create *configure* by a GNU program called *autoconf*. You only need to know this if you want to regenerate *configure* using a newer version of *autoconf*.

Windows NT

It is possible to build SRecord on MS Windows platforms, using the Cygwin (see `www.cygwin.com`) or DJGPP (see `www.delorie.com/djgpp`) environments. This provides the "porting layer" necessary to run Unix programs on Windows. The build process is exactly as described above.

You may need to pass in the include path to the Boost library. This is most simply done as

```
CC='gcc -no-cygwin' \
CXX='g++ -mno-cygwin -I/usr/include/boost-1_33_1' \
```

DJGPP always produces native binaries, however if you want to make native binaries with Cygwin (*i.e.* ones which work outside Cygwin) there is one extra step you need after running `./configure` and before you run `make`. You need to edit the *Makefile* file, and add `-mno-cygwin` to the end of the `CXX=g++` line.

Once built (using either tool set) Windows binaries should be testable in the same way as described in the next section. However, there may be some CRLF issues in the text file comparisons which give false negatives, depending on the CRLF setting of your Cygwin file system when you unpacked the tarball.

TESTING SRECORD

The *SRecord* package comes with a test suite. To run this test suite, use the command

```
% make sure
...lots of output...
Passed All Tests
%
```

The tests take a few seconds each, with a few very fast, and a couple very slow, but it varies greatly depending on your CPU.

If all went well, the message

```
Passed All Tests
```

should appear at the end of the make.

INSTALLING SRECORD

As explained in the *SITE CONFIGURATION* section, above, the *SRecord* package is installed under the */usr/local* tree by default. Use the `--prefix=`*PATH* option to *configure* if you want some other path. More specific installation locations are assignable, use the `--help` option to *configure* for details.

All that is required to install the *SRecord* package is to use the

```
% make install
...lots of output...
%
```

command. Control of the directories used may be found in the first few lines of the *Makefile* file and the other files written by the *configure* script; it is best to reconfigure using the *configure* script, rather than attempting to do this by hand.

GETTING HELP

If you need assistance with the *SRecord* package, please post to the srecord-users mailing list

```
srecord-users@lists.sourceforge.net
```

For information obout the srecord-users mailing list. http://srecord.sourceforge.net/mailing-list.html

When reporting problems, please include the version number given by the

```
% srec_cat -version
srecord version 1.64.D001
...warranty disclaimer...
%
```

command. Please do not send this example; run the program for the exact version number.

COPYRIGHT

srecord version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 SCott Finneran

The *SRecord* package is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

It should be in the *LICENSE* file included with this distribution.

AUTHOR

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

How to add a new file format

DESCRIPTION

This section describes how to add a new file format. It's mostly a set of reminders for the maintainer. If you want a format added to the distribution, use this method and e-mail the maintainer a patch (generated with `diff -Nur`, usually) and it can be added to the sources if appropriate.

New Files

The directory hierarchy is an echo of the class hierarchy, making it easy to guess the filename of a class, and to work out the appropriate file name of a new class. You get used to it. It is suggested that you simply work in the root of the source tree (exploiting tab-completion in your shell and your editor) rather than continually changing directories up and down the source tree. All of the file names below assume this.

The following files need to be creates for a new format.

srecord/output/file/*name*.cc
: This file is how to write the new format. Take a look at the other files in the same directory for examples. Also check out *srecord/output/file.h* and *srecord/output.h* for various helper methods.

srecord/output/file/*name*.h
: This is the class declaration for the above file.

srecord/input/file/*name*.cc
: This file is how to read the new format. Take a look at the other files in the same directory for examples. Also check out *srecord/input/file.h* and *srecord/input.h* for various helper methods.

srecord/input/file/*name*.h
: This is the class declaration for the above file.

man/man5/srec_*name*.5
: This file describes the format. Take a look at the other files in the same directory for examples.

  If you need to describe something as "stupid", as is all too often the case, use `thesaurus.com` to find a synonym. Use the following command

  ```
  find man/. -type f | xargs grep -i synonym
  ```

  to make sure it hasn't been used yet.

test/*nn*/t*nnmm*a.sh
: You may have noticed that SRecord comes with a lot of tests. You are more likely to get the patch for your new format accepted rapidly if it comes with at least one test for its output class, and at least one test for its input class.

  If your filter has endian-ness, add tests for each endian.

Modified Files

The following files need to be updated to mention the new format.

srecord/srecord.h
: Add the new include file to the list. This file controls what files are installed into the `/usr/include` directory. Not all of them, just the public interface.

etc/README.man
: Mention the new format in the section of this file which describes the supported file formats.

etc/index.html
: Mention the new format in the section of this file which describes the supported file formats.

srecord/arglex/tool.h
: Add the new format to the command line argument type enum.

  If your filter has endian-ness, add one for each endian, using "_be" and "_le" suffixes.

srecord/arglex/tool.cc
: Add the new format to the array of command line arguments types.

  If your filter has endian-ness, add one for each endian, using "_Big_Endian" and "_Little_Endian" suffixes.

srecord/arglex/tool/input.cc
: Add the new format to the code which parses input formats.

srecord/arglex/tool/output.cc
: Add the new format to the code which parses output formats.

srecord/input/file/guess.cc
: Add the new format to the list of formats which are tested.

man/man1/srec_input.1
: Mention the new format in the section of this file which describes the supported input file formats.

man/man1/srec_cat.1
: Mention the new format in the section of this file which describes the supported output file formats.

Makefile
: Actually, the system the maintainer uses automatically generates this file, but if you aren't using Aegis you will need to edit this file for your own use.

Tests

You may have noticed that SRecord comes with a lot of tests. You are more likely to get the patch for your new format accepted rapidly if it comes with at least one test for its output class, and at least one test for its input class.

If your filter has endian-ness, add tests for each endian.

IMPLEMENTATION ISSUES

In implementing a new file format, there are a couple of philosophical issues which affect technical decisions:

Be liberal in what you accept
: Where ever possible, consume the widest possible interpretation of valid data. This includes treating mandatory input fields as optional (*e.g.* file headers and execution start addresses), and coping with input definitions to their logical extremes (*e.g.* 255 byte data records in Motorola format). Checksums should always be checked on input, only ignore them if the −ignore-checksums command line option has been given. Absurd line lengths must be tolerated.

Be conservative in what you produce
: Even when the input is questionable, the output produced by *srec_cat* must always be strictly conforming with the format definition (except as mandated by command line options, see below). Checksums, if the format has them, must always be correct on output. Line lengths should default to something reasonable (about 80 characters or less).

Eat Your Own Dog Food
: You input class must always be able to consume what your output class produces, no matter what combination of command line options (see below) has been selected.

Round Trip

: In general, what went in is what comes out.

  - The data may be re-arranged in order, the line lengths may change, but the same data should go out as came in. (The data should be unchanged even if the format changed, assuming equally capable formats.) The *srec_cmp*(1) command may be used to verify this.
  - If the input has no header record, the output should not have one either (if at all possible). This means not automatically inserting a header record if the output file code sees data as the

first method call. (The −disable=header option affects this, too.)

- If the input has no execution start address record, the output should not have one either (if at all possible). This means not automatically inserting an execution start address record if the output file code does not see one by the time the destructor is called. (The −disable=exec-start-addr flag affects this, too.)
- Write at least one **test** that does a "round trip" of data through the new format and back again, exercising any interesting boundary conditions along the way (*e.g.* data records spanning segment boundaries).

Holes Do not to fill in holes in the data. That said, sometimes you *have* to fill holes in the data. This happens, for example, when a 16-bit format is faced with an 8-bit byte of data for one or other half of a 16-bit word. If there is no other way around it, call the fatal_alignment_error method, which will suggest a suitable input filter.

OPTIONS

There are also some command line arguments you will need to take into account:

−address-length
This options is used to specify the minimum address length, if your new format has a choice about how many bytes of address it produces.

−data-only
This option implies all of the **−disable=header**, **−disable=data-count** **−disable=exec-start-addr** and **−disable=footer** options. Only the essential data records are produced.

−disable=header
If this option is used, no header records are to be produced (or minimal header records). This is available as the `enable_header_flag` class variable in the methods of your derived class.

−disable=data-count
If this option is used, no data record count records are to be produced. This is available as the `enable_data_count_flag` class variable in the methods of your derived class.

−disable=exec-start-addr
If this option is used, no execution start address records are to be produced. This is available as the `enable_goto_addr_flag` class variable in the methods of your derived class.

−disable=footer
If this option is used, no end-of-file records are to be produced. This is available as the `enable_footer_flag` class variable in the methods of your derived class.

−enable=optional-address
If this option is used, in combination with a format that does not have an address on every line, the the first zero address many be omitted. All subsequent addresses are not optional, just the first zero address. Defaults to disabled.

−ignore-checksums
If this flag is set, your file input methods must parse *but not check* checksums, if the format has checksums. You can tell if you need to use checksums by calling the `use_checksums()` method within the implementation of your derived class. This only applies to input; output must always produce correct checksums.

−line-length
Where your output format is text, and there exists the possibility of putting more or less text on each line (*e.g.* the Motorola format allows a variable number of data bytes per record) then this should be controllable. This manifests in the `address_length_set` and `preferred_block_size_get` methods you must implement in your derived class.

CODING STYLE

Please following the coding style of the existing code. It makes your patches and contributions more likely to be accepted if they don't have to be extensively reformatted.

Indent increments are four characters. Do not use tab characters at all, nobody can agree how wide they are supposed to be. Line length is 80 characters or fewer, no exceptions.

Please follow the existing convention of always using Doxygen comments on all your instance variables and methods, even for private methods. Always document all arguments of all methods, even private methods, using `@param` tags; see existing style. Always use whole sentences in your Doxygen documentation, see existing code for examples.

Do not use upper case letters in file names. Do not use white space or shell special characters in file names.

When sending a patch please use “diff −Nur”, as this will include your new files in the patch, and you will not need additional attachments in your email. Patches are preferred over tarballs.

Include tests. It makes your patches and contributions more likely to be accepted if the maintainer doesn’t have to write your tests for you. See sources for examples of existing tests.

CONTRACT RATES

It is possible to have the maintainer write your new file format or new filter for you. However, if you want it done for nothing, you will be put at the end of a (very) long queue of other *gratis* open source work the maintainer has yet to do. You can jump the queue if you want to pay the maintainer to do the work for you.

The maintainer’s rates are AU$100 per hour.

A well document new format typically takes six hours to write and test, this includes both reading and writing the new format. A well documented new filter typically takes three hours to write and test.

Examples make these tasks easier. Poor documentation makes these tasks take longer. A mystery format that requires reverse engineering may take *much* longer; ask again once you have figured it out.

All code written for you will be included in the project source tarball in its next release. All formats and filters written for you will be copyright Scott Finneran;E-Mail:;scottfinneran@yahoo.com.au, and they will be GNU GPL licensed. If you *need* a format or filter written, it has value to you; the issue of freeloaders is irrelevant.

Conversely, integrating complete open source contributions and patches is done *gratis*, and usually done as promptly as time permits.

AUTHOR

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

How to add a new filter

DESCRIPTION

This section describes how to add a new filter. It's mostly a set of reminders for the maintainer. If you want a filter added to the distribution, use this method and e-mail the maintainer a patch (generated with `diff -Nur`, usually) and it can be added to the sources if appropriate.

New Files

The directory hierarchy is an echo of the class hierarchy, making it easy to guess the filename of a class, and to work out the appropriate file name of a new class. You get used to it. It is suggested that you simply work in the root of the source tree (exploiting tab-completion in your shell and your editor) rather than continually changing directories up and down the source tree. All of the file names below assume this.

The following files need to be created for a new filter.

srecord/input/filter/*name*.cc
: This file is how to process the new filter. Take a look at the other files in the same directory for examples. Also read *srecord/input.h* and *srecord/input/filter.h* for various helper methods.

srecord/input/filter/*name*.h
: This is the class declaration for the above file.

srecord/input/filter/message/*name*.cc
: If your filter needs all of the data to be known before it can proceed, or it needs all of the data to appear in ascending address order, derive from the `srec_input_filter_message` class, instead. This takes care of all data handling, you only have to write the method that computes the result from the data. Take a look at the other files in the same directory for examples.

srecord/input/filter/message/*name*.h
: This is the class declaration for the above file.

test/*nn*/t*nnmm*a.sh
: You may have noticed that SRecord comes with a lot of tests. You are more likely to get the patch for your new filter accepted rapidly if it comes with at least one test.

Modified Files

The following files need to be updated to mention the new filter.

srecord/srecord.h
: Add the new include file to the list. This file controls what files are installed into the `/usr/include` directory. Not all of them, just the public interface.

etc/README.man
: Mention the new filter in the section of this file which describes the supported filters.

etc/index.html
: Mention the new filter in the section of this file which describes the supported filters.

srecord/arglex/tool.h
: Add the new filter to the command line argument type enum.

  If your filter has endian-ness, add one for each endian, using "_be" and "_le" suffixes.

srecord/arglex/tool.cc
: Add the new filter to the array of command line arguments types.

  If your filter has endian-ness, add one for each endian, using "_Big_Endian" and "_Little_Endian" suffixes.

srecord/arglex/tool/input.cc
: Add the new filter to the code which parses input filters.

  If your filter has endian-ness, add your command line tokens to the switch in the srecord::arglex_tool::get_endian_by_token method.

man/man1/o_input.so
: Mention the new filter in the section of this file which describes the supported input filters.

Makefile
: Actually, the system the maintainer has Aegis automatically generate this file, but if you aren't using Aegis you will need to edit this file for your own use.

Tests

You may have noticed that SRecord comes with a lot of tests. You are more likely to get the patch for your new filter accepted rapidly if it comes with at least one test.

If your filter has endian-ness, add tests for each endian.

IMPLEMENTATION ISSUES

In implementing a new filter, there are a couple of philosophical issues which affect technical decissions:

- Be liberal in what you accept. Where ever possible, consume the widest possible interpretation of "valid" data. You especially need to cope with data with holes, and data records out of order, and data records not nicely aligned.

If your filter has endian-ness, add tests for each endian.

- Be conservative in what you produce. Even when the input is weird, the output produced by the filter must be conforming. *E.g.* the byte-swap filter still works when it has only one of the two bytes, and the other is a hole; it swaps the byte and the hole.
- If the input has no header record, the output should not have one either.
- If the input has no execution start address record, the output should not have one either.
- Do not to fill in holes in the data, unless you are a writing a "fill" filter. See the `srecord/input/filter/message.cc` file for an example of issuing a warning in the presence of holes.
- If the new filter is supposed to be its own inverse (*e.g.* byte-swap), or a pair of filters are supposed to be inverses (*e.g.* split and unsplit) be sure to write a test to confirm this. The tests should exersize all of the boundary conditions (*e.g.* around the edges of holes, extremes of data ranges).

CODING STYLE

Please following the coding style of the existing code. It makes your patches and contributions more likely to be accepted if they don't have to be extensively reformatted.

Indent increments are four characters. Do not use tab characters at all, nobody can agree how wide they are supposed to be. Line length is 80 characters or fewer, no exceptions.

Please follow the existing convention of always using Doxygen comments on all your instance variables and methods, even for private methods. Always document all arguments of all methods, even private methods, using `@param` tags; see existing style. Always use whole sentences in your Doxygen documentation, see existing code for examples.

Do not use upper case letters in file names. Do not use white space or shell special characters in file names.

When sending a patch please use "diff −Nur", as this will include your new files in the patch, and you will not need additional attachments in your email. Patches are preferred over tarballs.

Include tests. It makes your patches and contributions more likely to be accepted if the maintainer doesn't have to write your tests for you. See sources for examples of existing tests.

CONTRACT RATES

It is possible to have the maintainer write your new file format or new filter for you. However, if you want it done for nothing, you will be put at the end of a (very) long queue of other *gratis* open source work the maintainer has yet to do. You can jump the queue if you want to pay the maintainer to do the work for you.

The maintainer's rates are AU$100 per hour.

A well document new format typically takes six hours to write and test, this includes both reading and

writing the new format. A well documented new filter typically takes three hours to write and test.

Examples make these tasks easier. Poor documentation makes these tasks take longer. A mystery format that requires reverse engineering may take *much* longer; ask again once you have figured it out.

All code written for you will be included in the project source tarball in its next release. All formats and filters written for you will be copyright Scott Finneran;E-Mail:;scottfinneran@yahoo.com.au, and they will be GNU GPL licensed. If you *need* a format or filter written, it has value to you; the issue of freeloaders is irrelevant.

Conversely, integrating complete open source contributions and patches is done *gratis*, and usually done as promptly as time permits.

AUTHOR

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_cat − manipulate EPROM load files

SYNOPSIS

srec_cat [*option*...] *filename*...
srec_cat −Help
srec_cat −VERSion

DESCRIPTION

The *srec_cat* program is used to assemble the given input files into a single output file. The use of filters (see below) allows significant manipulations to be performed by this command.

Data Order

The data from the input files is not immediately written to the output, but is stored in memory until the complete EPROM image has been assembled. Data is then written to the output file in ascending address order. The original ordering of the data (in those formats capable of random record ordering) is *not* preserved.

Data Comparison

Because input record order is not preserved, textual comparison of input and output (such as the *diff*(1) or *tkdiff*(1) commands) can be misleading. Not only can lines appear in different address orders, but line lengths and line termination can differ as well. Use the *srec_cmp*(1) program to compare two EPROM load files. If a text comparison is essential, run both files through the *srec_cat*(1) program to ensure both files to be compared have identical record ordering and line lengths.

Data Conflicts

The storing of data in memory enables the detection of data conflicts, typically caused by linker sections unintentionally overlapping.

- A warning will be issued for each address which is redundantly set to the same value.
- A fatal error will be issued if any address is set with contradictory values. To avoid this error use an **−exclude −within** filter (see *srec_input*(1)) or, to make it a warning, use the **−contradictory-bytes** option (see below).
- A warning will be issued for input files where the data records are not in strictly ascending address order. To suppress this warning, use the **−disable-sequence-warning** option (see below).

These features are designed to detect problems which are difficult to debug, and detects them *before* the data is written to an EPROM and run in your embedded system.

INPUT FILE SPECIFICATIONS

Input may be qualified in two ways: you may specify a data file or a data generator. format and you may specify filters to apply to them. An input file specification looks like this:

data-file [*filter* ...]
data-generator [*filter* ...]

Data Files

Input from data files is specified by file name and format name. An input file specification looks like this:

filename [*format*][−ignore-checksums]

The default format is Motorola S-Record format, but *many* others are also understood.

Data Generators

It is also possible to generate data, rather than read it from a file. You may use a generator anywhere you could use a file. An input generator specification looks like this:

−GENerate *address-range* *−data-source*

Generators include random data and various forms of constant data.

Common Manual Page

See *srec_input*(1) for complete details of input specifiers. This description is in a separate manual page because it is common to more than one SRecord command.

OPTIONS

The following options are understood:

@*filename*

The named text file is read for additional command line arguments. Arguments are separated by white space (space, tab, newline, *etc*). There is no wildcard mechanism. There is no quoting mechanism. Comments, which start with '#' and extend to the end of the line, are ignored. Blank lines are ignored.

−Output *filename* [*format*]

This option may be used to specify the output file to be used. The special file name "−[rq] is understood to mean the standard output. Output defaults to the standard output if this option is not used.

The *format* may be specified as:

−Absolute_Object_Module_Format

An Intel Absolute Object Module Format file will be written. (See *srec_aomf* (5) for a description of this file format.)

−Ascii_Hex

An Ascii-Hex file will be written. (See *srec_ascii_hex*(5) for a description of this file format.)

−ASM [*prefix*][−*option*...]

A series of assembler DB statements will be written.

The optional *prefix* may be specified to change the names of the symbols generated. The defaults to "`eprom`" if not set.

Several options are available to modify the style of output:

−Dot_STyle

Use "dot" style pseudo-ops instead of words. For example `.byte` instead of the `DB` default.

−HEXadecimal_STyle

Use hexadecimal numbers in the output, rather than the default decimal numbers.

−Section_STyle

By default the generated assemble of placed at the correct address using `ORG` pseudo-ops. Section style output emits tables of section addresses and lengths, so the data may be related at runtime.

−A430 Generate output which is compliant to the `a430.exe` compiler as it is used, *e.g.* in IAR Embedded Workbench. This is short-hand for −section-style −hex-style

−CL430 Generate output which is Code Composer Essentials compliant, *i.e.* the compiler of it. This is short-hand for −section-style −hex-style −dot-style

−Output_Word

Generate output which is in two-byte words rather than bytes. This assumes little-endian words; you will need to use the −Byte-Swap filter if your target is big-endian. No attempt is made to align the words onto even address boundaries; use and input filter such as

input-file −fill 0xFF −within *input-file* −range-pad 2

to pad the data to whole words first.

−Atmel_Generic
An Atmel Generic file will be written. (See *srec_atmel_generic*(5) for a description of this file format.)

−BASic A series of BASIC DATA statements will be written.

−B-Record
A Freescale MC68EZ328 Dragonball bootstrap b-record format file will be written. (See *srec_brecord*(5) for a description of this file format.)

−Binary
A raw binary file will be written. If you get unexpected results **please** see the *srec_binary*(5) manual for more information.

−C-Array [*identifier*][*−option...*]
A C array defintion will be written.

The optional *identifier* is the name of the variable to be defined, or bugus if not specified.

−INClude
This option asks for an include file to be generated as well.

−No-CONST
This options asks for the variables to not use the const keyword (they are declared constant be default, so that they are placed into the read-only segment in embedded systems).

−C_COMpressed
These options ask for an compressed c-array whose memory gaps will not be filled.

−Output_Word
This option asks for an output which is in words not in bytes. This is little endian, so you may need to

−PREfix *string*
This option allows a string to be prepended to the array definition. This is commonly used for non-standard options common to cross compilers.

−POSTfix *string*
This option allows a string to be appended to the array definition. This is commonly used for non-standard options common to cross compilers.

−COE This option says to use the Xilinx Coefficient File Format (.coe) for output. (See *srec_coe*(5) for a description of this file format.)

−COsmac
An RCA Cosmac Elf format file will be written. (See *srec_cosmac*(5) for a description of this file format.)

−Dec_Binary
A DEC Binary (XXDP) format file will be written. (See *srec_dec_binary*(5) for a description of this file format.)

−Elektor_Monitor52
This option says to use the EMON52 format file when writing the file. (See *srec_emon52*(5) for a description of this file format.)

−FAIrchild
This option says to use the Fairchild Fairbug format file when writing the file. (See *srec_fairchild*(5) for a description of this file format.)

−Fast_Load
This option says to use the LSI Logic Fast Load format file when writing the file. (See *srec_fastload*(5) for a description of this file format.)

−Formatted_Binary
A Formatted Binary format file will be written. (See *srec_formatted_binary*(5) for a description of this file format.)

−FORTH [*−option*]
A FORTH input file will be written. Each line of output includes a byte value, an address, and a command.

−RAM The store command is C! This is the default.

−EEPROM
The store command is EEC!

−Four_Packed_Code
This option says to use the PFC format file when writing the file. (See *srec_fpd*(5) for a description of this file format.)

−HEX_Dump
A human readable hexadecimal dump (including ASCII) will be printed.

−IDT An IDT System Integration Manager (IDT/sim) binary file will be written. (See *srec_idt*(5) for a description of this file format.)

−Intel An Intel hex format file will be written. (See *srec_intel*(5) for a description of this file format.) The default is to emit “i32hex” 32-bit linear addressing; if you want “i16hex” 20-bit extended segment addressing use the **−address-length=3** option, if you want “i8hex” 16-bit addressing use the **−address-length=2** option.

−Intel_HeX_16
An Intel-16 hex format (INHX16) file will be written. (See *srec_intel16*(5) for a description of this file format.)

−Lattice_Memory_Initialization_Format [*width* **]**
The Memory Initialization Format (.mem) by Lattice Semiconductor is understood for writing only. (A.k.a. **−MEM**) (See *srec_mem*(5) for a description of this file format.)

−LOGisim
LOginsim logic simuator uses the format See **−srec_logisim**(5) form more information.

−Memory_Initialization_File [*width*]
Memory Initialization File (MIF) by Altera format will be written. The *width* defaults to 8 bits. (See *srec_mif*(5) for a description of this file format.)

−Mips_Flash_Big_Endian

−Mips_Flash_Little_Endian
MIPS Flash file format will be written. (See *srec_mips_flash*(5) for a description of this file format.)

−MOS_Technologies
An Mos Technologies format file will be written. (See *srec_mos_tech*(5) for a description of this file format.)

−Motorola [*width*]
A Motorola S-Record file will be written. (See *srec_motorola*(5) for a description of this file format.) This is the default output format. By default, the smallest possible address length is emitted, this will be S19 for data in the first 64KB; if you wish to force S28 use the **−address-length=3** option; if you wish to force S37 use the **−address-length=4** option

The optional *width* argument describes the number of bytes which form each address

multiple. For normal uses the default of one (1) byte is appropriate. Some systems with 16-bit or 32-bit targets mutilate the addresses in the file; this option will imitate that behavior. Unlike most other parameters, this one cannot be guessed.

−MsBin This option says to use the Windows CE Binary Image Data Format to write the file. See *srec_msbin*(5) for a description of this file format.

−Needham_Hexadecimal
This option says to use the Needham Electronics ASCII file format to write the file. See *srec_needham*(5) for a description of this file format.

−Ohio_Scientific
This option says to use the Ohio Scientific hexadecimal format. See *srec_os65v*(5) for a description of this format.

−PPB This option says to use the Stag Prom Programmer binary format. See *srec_ppb*(5) for a description of this format.

−PPX This option says to use the Stag Prom Programmer hexadecimal format. See *srec_ppx*(5) for a description of this format.

−SIGnetics
This option says to use the Signetics hex format. See *srec_signetics*(5) for a description of this format.

−SPAsm
This option says to use the SPASM assembler output format (commonly used by PIC programmers). See *srec_spasm*(5) for a description of this format.

−SPAsm_Little_Endian
This option says to use the SPASM assembler output format (commonly used by PIC programmers). But with the data the other way around.

−STewie
A Stewie binary format file will be written. (See *srec_stewie*(5) for a description of this file format.)

−Tektronix
A Tektronix hex format file will be written. (See *srec_tektronix*(5) for a description of this file format.)

−Tektronix_Extended
A Tektronix extended hex format file will be written. (See *srec_tektronix_extended*(5) for a description of this file format.)

−Texas_Instruments_Tagged
A TI-Tagged format file will be written. (See *srec_ti_tagged*(5) for a description of this file format.)

−Texas_Instruments_Tagged_16
A Texas Instruments SDSMAC 320 format file will be written. (See *srec_ti_tagged_16*(5) for a description of this file format.)

−Texas_Instruments_TeXT
This option says to use the Texas Instruments TXT (MSP430) format to write the file. See *srec_ti_txt*(5) for a description of this file format.

−TRS80
This option says to use the Radio Shack TRS-80 object file format to write the file. See *srec_trs80*(5) for a description of this file format.

−VHdl [*bytes-per-word* [*name*]]
A VHDL format file will be written. The *bytes-per-word* defaults to one, the *name* defaults to `eprom`. The *etc/x_defs_pack.vhd* file in the source distribution contains an

example ROM definitions pack for the type-independent output. You may need to use the −byte-swap filter to get the byte order you want.

−VMem [*memory-width*]
A Verilog VMEM format file will be written. The *memory-width* may be 8, 16, 32, 64 or 128 bits; defaults to 32 if unspecified. (See *srec_vmem*(5) for a description of this file format.) You may need to use the −byte-swap filter to get the byte order you want.

−WILson
A wilson format file will be written. (See *srec_wilson*(5) for a description of this file format.)

−Address_Length *number*
This option many be used to specify the minimum number of bytes to be used in the output to represent an address (padding with leading zeros if necessary). This helps when talking to imbecilic EPROM programmer devices which do not fully implement the format specification.

−CRLF This option is short-hand for the **−line-termination=crlf** option. For use with hare-brained EPROM programmer devices which assume all the world uses Evil Bill's operating system's line termination.

−Data_Only
This option implies the **−disable=header**, **−disable=data-count**, **−disable=exec-start-address** and **−disable=footer** options.

−DISable *feature-name*
This option is used to disable the output of a named feature. See the **−enable** option for a description of the available features.

−ENable *feature-name*
This option is used to enable the output of a named feature.

Header This feature controls the presence of header records, records which appear before the data itself. Headers often, but not always, include descriptive text.

Execution_Start_Address
This feature controls the presence of execution start address records, which is where the monitor will jump to and start executing code once the hex file has finished loading.

Data_Count
This feature controls the presence of data record count records, which appear aftre the data, and state how many data records preceeded them. Usually a data integrity mechanism.

Footer This feature controls the presence of a file termination record, one that *does not* double as an execution start address record.

Optional_Address
In formats that have the address and the data separated or partially separated (as opposed to having a complete address in every record) it is possible to disable emitting the first address where that address would be zero, as these format often default the address to zero if no address is seen beofre the first data record. This is disabled by default, the zero address is always emitted.

Not all formats have all of the above features. Not all formats are able to optionally omit any or all the above features. Feature names may be abbreviated like command line option names.

−Execution_Start_Address *number*
This option may be used to set the execution start address, in those formats which support it. The execution start address is where the monitor will jump to and start executing code once the hex file has finished loading, think of it as a "goto" address. Usually ignored by EPROM programmer devices. This option implies the **−enable=exec-start-addr** option.

Please note: the execution start address is a different concept than the first address in memory of your data. If you want to change where your data starts in memory, use the **−offset** filter.

−HEAder *string*
This option may be used to set the header comment, in those formats which support it. This option implies the **−enable=header** option.

If you need to inject binary data into the header, use the URL encoding that uses % followed by two hexadeimal characters. For example a backspace would be encoded as “%08”.

−IGnore_Checksums
The **−IGnore-Checksums** option may be used to disable checksum validation of input files, for those formats which have checksums at all. Note that the checksum values are still read in and parsed (so it is still an error if they are missing) but their values are not checked. Used after an input file name, the option affects that file alone; used anywhere else on the command line, it applies to all following files.

−Line_Termination *style-name*
This option may be used to specify line termination style for text output. The default is to use the host operating system’s default line termination style (but Cygwin behaves as if it’s Unix). Use this option with caution, because it will also introduce extra (i.e. wrong) CR bytes into binary formats.

Carriage_Return_Line_Feed
Use the CRLF line termination style, typical of DOS and M$ Windows.

NewLine
Use the NL line termination style, typical of Unix and Linux.

Carriage_Return
Use the CR line termination style, typical of Apple Macintosh.

All other line termination style names will produce a fatal error. Style names may be abbreviated like command line option names.

−Line_Length *number*
This option may be used to limit the length of the output lines to at most *number* characters. (Not meaningful for binary file format.) Defaults to something less than 80 characters, depending on the format. If you need to control the maximum number of bytes in each output record, use the **−−Ouput_Block_Size** option.

−Output_Block_Size *number*
This option may be used to specify the exact number of data bytes to appear in each output record. There are format-specific limitations on this value, you will get an error if the value isn’t valid. If you need to control the maximum number of characters on a line of text output, use the **−−Line_Length** option.

−Output_Block_Packing
From time to time, with large files, you may notice that your data records are spit unexpectedly on output. This usually happens where record lengths are not a power of 2. If this bothers you (or your comparison tools) this option may be used to repack the output so that SRecord’s internal block boundaries are not visable in the output.

−Output_Block_Alignment
This option is similar to the **−Output_Block_Packing** option, except that short records are used after holes to cause subsequent records to be placed on a block size boundary.

−Enable_Sequence_Warnings
This option may be used to enable warnings about input files where the data records are not in strictly ascending address order. Only one warning is issued per input file. This is the default.

Note: the output of *srec_cat*(1) is always in this order.

Note: This option must be used *before* the input file. This is because if there are several files on the command line, each may need different settings. The setting remains in force until the next **−Disable_Sequence_Warnings** option.

−Disable_Sequence_Warnings

This option may be used to disable warnings about input files where the data records are not in strictly ascending address order.

Note: This option must be used *before* the offending input file. This is because if there are several files on the command line, each may need different settings. The setting remains in force until the next **−Ensable_Sequence_Warnings** option.

All other options will produce a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−help", "−HEL" and "−h" are all interpreted to mean the **−Help** option. The argument "−hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *srec_cat* are long, this means ignoring the extra leading "−". The "*−−option=value*" convention is also understood.

EXIT STATUS

The *srec_cat* command will exit with a status of 1 on any error. The *srec_cat* command will only exit with a status of 0 if there are no errors.

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_cmp − compare two EPROM load files for equality

SYNOPSIS

srec_cmp [*option*...] *filename*...
srec_cmp −Help
srec_cmp −VERSion

DESCRIPTION

The *srec_cmp* program is used to compare two EPROM load files for equality. This comparison is performed irrespective of the load order of the data in each of the files.

INPUT FILE SPECIFICATIONS

Input may be qualified in two ways: you may specify a data file or a data generator. format and you may specify filters to apply to them. An input file specification looks like this:

data-file [*filter* ...]
data-generator [*filter* ...]

Data Files

Input from data files is specified by file name and format name. An input file specification looks like this:

filename [*format*][−ignore-checksums]

The default format is Motorola S-Record format, but *many* others are also understood.

Data Generators

It is also possible to generate data, rather than read it from a file. You may use a generator anywhere you could use a file. An input generator specification looks like this:

−GENerate *address-range* *−data-source*

Generators include random data and various forms of constant data.

Common Manual Page

See *srec_input*(1) for complete details of input specifiers. This description is in a separate manual page because it is common to more than one SRecord command.

OPTIONS

The following options are understood:

@*filename*

The named text file is read for additional command line arguments. Arguments are separated by white space (space, tab, newline, *etc*). There is no wildcard mechanism. There is no quoting mechanism. Comments, which start with '#' and extend to the end of the line, are ignored. Blank lines are ignored.

−Help

Provide some help with using the *srec_cmp* program.

−IGnore_Checksums

The **−IGnore-Checksums** option may be used to disable checksum validation of input files, for those formats which have checksums at all. Note that the checksum values are still read in and parsed (so it is still an error if they are missing) but their values are not checked. Used after an input file name, the option affects that file alone; used anywhere else on the command line, it applies to all following files.

−Enable_Sequence_Warnings

This option may be used to enable warnings about input files where the data records are not in strictly ascending address order. Only one warning is issued per input file. This is the default.

Note: the output of *srec_cat*(1) is always in this order.

Note: This option must be used *before* the input file. This is because if there are several files on the command line, each may need different settings. The setting remains in force until the next **−Disable_Sequence_Warnings** option.

−Disable_Sequence_Warnings
This option may be used to disable warnings about input files where the data records are not in strictly ascending address order.

Note: This option must be used *before* the offending input file. This is because if there are several files on the command line, each may need different settings. The setting remains in force until the next **−Ensable_Sequence_Warnings** option.

−redundant-bytes=*value*
Use this option to permit a file to contain redundant values for some memory locations. The default is for this condition to be a warning.

ignore
No warning or error is issued whena redundant settings are detected.

warning
A warning is issued when a redundant settings are observed, the warning includes the problematic address.

error
A fatal error is issued when a redundant settings are observed, the fatal error message includes the problematic address and byte value.

−contradictory-bytes=*value*
Use this option to permit a file to contain contradictory values for some memory locations. The last value in the input(s) will be used. The default is for this condition to be a fatal error.

ignore
No warning or error is issued when contradictory setting is detected.

warning
A warning is issued when a vontradictory settings are observed, the warning includes the problematic address, and values.

error
A fatal error is issued when contradictory settings are observed, the fatal error message includes the problematic address and byte values.

−VERSion
Print the version of the *srec_cmp* program being executed.

−Verbose
This option may be used to obtain more information about how and where the two files differ. Please note that this takes longer, and the output can be voluminous.

All other options will produce a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−help”, “−HEL” and “−h” are all interpreted to mean the **−Help** option. The argument “−hlp” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *srec_cmp* are long, this means ignoring the extra leading “−”. The “−−*option*=*value*” convention is also understood.

EXIT STATUS

The *srec_cmp* command will exit with a status of 1 on any error. The *srec_cmp* command will only exit with a status of 0 if there are no errors.

EXAMPLE

A common use for the *srec_cmp* command is to verify that a particular signature is present in the code. In this example, the signature is in a file called “signature[rq], and the EPROM image is in a file called “image[rq]. We assume they are both Motorola S-Record format, although this will work for all formats:

```
srec_cmp signature  image −crop −within signature
```

The signature need not be at the start of memory, nor need it be one single contiguous piece of memory. In the above example, the portions of the image which have the same address range as the signature are compared with the signature.

COPYRIGHT

srec_cmp version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cmp* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cmp −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cmp −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_examples − examples of how to use SRecord

DESCRIPTION

The *srec_cat* command is very powerful, due to the ability to combine the the input filters in almost unlimited ways. This manual page describes a few of them.

This manual page describes how to use the various input files, input filters and input generators. But these are only examples, for more complete details, see the *srec_input*(1) manual page.

The Commands Lines Are Too Long

If you are marooned on an operating system with absurdly short command line length limits, some of the commands which follow may be too long. You can get around this handicap by placing your command line in a file, say *fred.txt*, and then tell *srec_cat*(1) to read this file for the rest of its command line, like this

```
srec_cat @fred.txt
```

This also has the advantage of allowing comments, allowing you to write your command line options over several lines, and even indenting to make the command more clear. Comments start at a "#" and extend to the end of the line. Blank lines are ignored.

Of course, you could always upgrade to Linux, which has been sucking less for over 25 years now.

Your Examples Wanted

If you have a clever way of using SRecord, or have solved a difficult problem with SRecord, you could contribute to this manual page, making it more useful for everyone. Send your example in an email to the email address at the end of this manual page.

CONVERTING FILE FORMATS

The simplest of the things *srec_cat*(1) can do is convert from one EPROM file format to another. Please keep in mind, as you read this section, that you can do many of these things simultaneously in one command. They are only broken out separately to make them easier to understand.

Intel to Motorola

One of the simplest examples is converting files from Intel hex format to Motorola S-Record format:

```
srec_cat intel-file −intel −o srec-file
```

Note that the format specifier immediately follows the name of the file it is describing. Pick any two formats that SRecord understands, and it can convert between all of them. (Except the assembler, BASIC, C and FPGA outputs which are write only.)

Motorola to Intel

Converting the other way is just as simple:

```
srec_cat srec-file −o intel-file −intel
```

The default format is Motorola S-Record format, so it does not need to be specified after the file name.

Different Shapes of the Same Format

It is regrettably common that some addle-pated EPROM programmers only implement a portion of the specification used to represent their hex files. For example, some compilers produce "s19" Motorola data (that is, S1 data records with S9 start records, 16 bit address fields) which would be OK except that some blockhead EPROM programmers insist on "s37" Motorola data (that is, S3 data records with S7 start records, 32 bit address fields).

It is possible to convert from one Motorola shape to another using the **−Address-Length** option:

```
srec_cat short.srec −o long.srec −address-length=4
```

This command says to use four byte (32-bit) addresses on output.

This section also applies to Intel hex files, as they, too, have the ability to select from a variety of address widths. To convert from one Intel shape to another using the same **−Address-Length** option:

```
srec_cat i32.hex -o i16.hex -address-length=3
```

This command says to use “i16hex” 20-bit segmented addresses on output. An address length of 4 is the default (“i32hex” 32-bit linear addressing), and an address length of 2 would request “i8hex” 16-bit addressing.

Line Lengths

From time to time you will come across a feeble-minded EPROM programmer that can’t cope with long text lines, they assume that there will only ever be 46 characters per line and barf when they see the default line lengths that *srec_cat*(1) writes (or worse, get a stack scribble and crash).

The Motorola S-record format definition permits up to 255 bytes of payload, or lines of *514* characters, plus the line termination. All EPROM programmers *should* have sufficiently large line buffers to cope with records this big. Few do.

The −line-length option may be used to specify the maximum line length (not including the newline) to be used on output. For example, 16 byte payloads for Motorola hex

```
srec_cat long.srec -o short.s19 -line-length=46
```

The line length option interacts with the address length option, so some tinkering to optimize for your particular situation many be necessary.

Output Block Size

Every once in a while you will come across an ancient daft EPROM programmer that can’t cope with long data records, they assume that there will only ever be at most 16 bytes of data per record, and barf when they see the default 32 byte payloads that *srec_cat*(1) writes (or worse, the buffer over-run causes a tall grass walk that scribbles on your EPROM).

The Intel hex format definition permits up to 255 bytes of payload data per record. All EPROM programmers *should* have sufficiently large data buffers to cope with records this big. Good luck with that.

The −Output-Block-Size option may be used to specify the record data size to be used on output. For example, Intel hex with 16 byte payloads:

```
srec_cat long.srec -o short.hex -intel -obs=16
```

Be careful not to put the **−obs** option between the output file name and the format specifier.

Just the Data, Please

There are some bonehead EPROM programmers which can only cope with data records, and are unable to cope with header records or execution start address records. If you have this problem, the **−data-only** option can be used to suppress just about everything except the data. The actual effect depends on the format, of course, because some don’t have these features anyway.

The **−data-only** option is short hand. There are four properties which may be **−disabled** or **−enabled** separately. See the *srec_cat*(1) man page for a description of the **−disabled** and **−enabled** options.

For example, your neanderthal EPROM programmer requires Motorola hex with header records (S0), but without data count (S5) records. Not using the **−data-only** option has it barf on the data count record, but using the **−data-only** option has it barf on the missing header record. Using the **−disable=data-count** option would leave the header record intact while suppressing the data count record.

Data Headers

The *srec_cat*(1) command always tries to pass through header records unchanged, whenever they are present. It even tries preserve them across file format changes, to the limit the file formats are capable of.

If there is no file header record and you would like to add one, or you wish to override an existing file header record, use the **−header**=*string* option. You will need to quote the string (to insulate it from the shell) if it contains spaces or shell meta-characters.

Execution Start Addresses

The *srec_cat*(1) command always tries to pass through execution start addresses (typically occurring at the end of the file), whenever they are present. They are adjusted along with the data records by the **−offset** filter. It even tries preserve them across file format changes, to the limit the file formats are capable of.

If there is no execution start address record and you would like to add one, or you wish to override an existing execution start address record, use the **−execution-start-address**=*number* option.

Please note: the execution start address is a different concept than the first address in memory of your data. Think of it as a “goto” address to be jumped to by the monitor when the hex load is complete. If you want to change where your data starts in memory, use the **−offset** filter.

Fixing Checksums

Some embedded firmware developers are saddled with featherbrained tools which produce incorrect checksums, which the more vigilant models of EPROM programmer will not accept.

To fix the checksums on a file, use the **−ignore-checksums** option. For example:

```
srec_cat broken.srec −ignore-checksums −o fixed.srec
```

The checksums in *broken.srec* are parsed (it is still and error if they are absent) but are not checked. The resulting *fixed.srec* file has correct checksums. The **−ignore-checksums** option only applies to input.

This option may be used on any file format which has checksums, including Intel hex.

Discovering Mystery Formats

See the **What Format Is This?** section, below, for how to discover and convert mystery EPROM load file formats.

BINARY FILES

It is possible to convert to and from binary files. You can even mix binary files and other formats together in the same *srec_cat*(1) command.

Writing Binary Files

The simplest way of reading a hex file and converting it to a binary file looks like this:

```
srec_cat fred.hex −o fred.bin −binary
```

This reads the Motorola hex file *fred.srec* and writes it out to the *fred.bin* as raw binary.

Note that the data is placed into the binary file at the byte offset specified by the addresses in the hex file. If there are holes in the data they are filled with zero. This is, of course, common with linker output where the code is placed starting at a particular place in memory. For example, when you have an image that starts at 0x100000, the first 1MB of the output binary file will be zero.

You can automatically cancel this offset using a command like

```
srec_cat fred.hex −offset – −minimum-addr fred.hex −o fred.bin
```

The above command works by offsetting the *fred.hex* file lower in memory by the least address in the *fred.hex* file’s data.

See also the *srec_binary*(5) man page for additional detail.

Reading Binary Files

The simplest way of reading a binary file and converting it looks like this

```
srec_cat fred.bin −binary −o fred.srec
```

This reads the binary file *fred.bin* and writes all of its data back out again as a Motorola S-Record file.

Often, this binary isn’t exactly where you want it in the address space, because it is assumed to reside at address zero. If you need to move it around use the **−offset** filter.

```
srec_cat fred.bin −binary −offset 0x10000 −o fred.srec
```

You also need to avoid file “holes” which are filled with zero. You can use the **−crop** filter, of you could use the **−unfill** filter if you don’t know exactly where the data is.

```
srec_cat fred.bin −binary −unfill 0x00 512 −o fred.srec
```

The above command removes runs of zero bytes that are 512 bytes long or longer. If your file contains 1GB of leading zero bytes, this is going to be slow, it may be better to use the *dd*(1) command to slice and dice first.

JOINING FILES TOGETHER

The *srec_cat* command takes its name from the UNIX *cat*(1) command, which is short for “catenate” or “to join”. The *srec_cat* command joins EPROM load files together.

All In One

Joining EPROM load files together into a single file is simple, just name as many files on the command line as you need:

```
srec_cat infile1 infile2 -o outfile
```

This example is all Motorola S-Record files, because that’s the default format. You can have multiple formats in the one command, and *srec_cat*(1) will still work. You don’t even have to output the same format:

```
srec_cat infile1 -spectrum infile2 -needham \
    -o outfile -signetics
```

These are all ancient formats, however it isn’t uncommon to have to mix and match Intel and Motorola formats in the one project.

Overlaying two data files

It is common to want to “join” two hex files together, without any changes of address. on the assumption neither file intersects with the other. This is a simple “layers”, it is quite common for linkers to output the main code, and then a whole bunch of relocation and jump destination, by writing a two layered files.

```
srec_cat one.he two.hex -o three.hex
```

Almost always you see an error

```
srec_cat: two.srec: 49282: contradictory 0x00000000 value (previous = 0x00, this one = 0x80)
```

This means that the files actually intersect, they try to set the same location. You can turn the error into a warning, using the **-contradictory-bytes=warning** command line option. But this will probably generate a bazillion warnings.

The necessary step is to crop the first file, to avoid the regions the second file is going o be overwriting.

```
srec_cat                                  \
    one.srec -exclude -within two.srec \
    two.srec -exclude -within one.srec \
    -o three.hex
```

Depending on your linker this will have no errors (but if it wants another layer, more jiggery-pokery is required).

Filtering After Joining

There are times when you want to join two sets of data together, and then apply a filter to the joined result. To do this you use parentheses.

```
srec_cat                                                  \
    '('                                                   \
        infile -exclude 0xFFF0 0x10000                   \
        -generate 0xFFF0 0xFFF8 -repeat-string 'Bananas ' \
    ')'                                                   \
    -length-b-e 0xFFF8 4                                  \
    -checksum-neg-b-e 0xFFFC 4 4                          \
    -o outfile
```

The above example command catenate an input file (with the generated data area excluded) with a constant string. This catenated input is then filtered to add a 4-byte length, and a 4-byte checksum.

Joining End-to-End

All too often the address ranges in the EPROM load files will overlap. You will get an error if they do. If both files start from address zero, because each goes into a separate EPROM, you may need to use the offset filter:

```
srec_cat infile1 \
    infile2 -offset 0x80000 \
    -o outfile
```

Sometimes you want the two files to follow each other exactly, but you don't know the offset in advance:

```
srec_cat infile1 \
    infile2 -offset -maximum-addr infile1 \
    -o outfile
```

Notice that where the was a number (0x80000) before, there is now a calculation (−maximum-addr *infile1*). This is possible most places a number may be used (also −minimum-addr and −range).

CROPPING THE DATA

It is possible to copy an EPROM load file, selecting addresses to keep and addresses to discard.

What To Keep

A common activity is to crop your data to match your EPROM location. Your linker may add other junk that you are not interested in, *e.g.* at the RAM location. In this example, there is a 1MB EPROM at the 2MB boundary:

```
srec_cat infile -crop 0x200000 0x300000 \
    -o outfile
```

The lower bound for all address ranges is inclusive, the upper bound is exclusive. If you subtract them, you get the number of bytes.

Address Offset

Just possibly, you have a moronic EPROM programmer, and it barfs if the EPROM image doesn't start at zero. To find out just where is *does* start in memory, use the *srec_info*(1) command:

```
$ srec_info example.srec
Format: Motorola S-Record
Header: extra-whizz tool chain linker
Execution Start Address: 0x00200000
Data:   0x200000 - 0x32AAEF
$
```

Rather than butcher the linker command file, just offset the addresses:

```
srec_cat infile -crop 0x200000 0x300000 -offset –0x200000 \
    -o outfile
```

Note that the offset given is *negative*, it has the effect of subtracting that value from all addresses in the input records, to form the output record addresses. In this case, shifting the image back to zero.

This example also demonstrates how the input filters may be chained together: first the crop and then the offset, all in one command, without the need for temporary files.

If all you want to do is offset the data to start from address zero, this can be automated, so you don't have to know the minimum address in advance, by using *srec_cat*'s ability to calculate some things on the command line:

```
srec_cat infile -offset - -minimum-addr infile \
    -o outfile
```

Note the spaces either side of the minus sign, they are mandatory.

What To Throw Away

There are times when you need to exclude an small address range from an EPROM load file, rather than wanting to keep a small address range. The **−exclude** filter may be used for this purpose.

For example, if you wish to exclude the address range where the serial number of an embedded device is kept, say 0x20 bytes at 0x100, you would use a command like this:

```
srec_cat input.srec -exclude 0x100 0x120 -o output.srec
```

The *output.srec* file will have a hole in the data at the necessary locations.

Note that you can have both **−crop** and **−exclude** on the same command line, whichever works more naturally for your situation.

Discontinuous Address Ranges

Address ranges don't have to be a single range, you can build up an address range using more than a single pair.

```
srec_cat infile −crop 0x100 0x200 0x1000 0x1200 \
    −o outfile
```

This filter results in data from 0x100..0x1FF and data from 0x1000..0x1200 to pass through, the rest is dropped. This is is more efficient than chaining a **−crop** and an **−exclude** filter together.

MOVING THINGS AROUND

It is also possible to change the address of data records, both forwards and backwards. It is also possible rearrange where data records are placed in memory.

Offset Filter

The **−offset=***number* filter operates on the addresses of records. If the number is positive the addresses move that many bytes higher in memory, negative values move lower.

```
srec_cat infile −crop 0x200000 0x300000 −offset −0x200000 \
    −o outfile
```

The above example moves the 1MB block of data at 0x200000 down to zero (the offset is *negative*) and discards the rest of the data.

Byte Swapping

There are times when the bytes in the data need to be swapped, converting between big-endian and little-endian data usually.

```
srec_cat infile −byte-swap 4 −o outfile
```

This reverses bytes in 32 bit values (4 bytes). The default, if you don't supply a width, is to reverse bytes in 16 bit values (2 bytes). You can actually use any weird value you like, it doesn't even have to be a power of 2. Perhaps 64 bits (8 bytes) may be useful one day.

Binary Output

You need to watch out for binary files on output, because the holes are filled with zeros. Your 100kB program at the top of 32-bit addressed memory will make a 4GB file. See *srec_binary*(5) for how understand and avoid this problem, usually with the **−offset** filter.

Splitting an Image

If you have a 16-bit data bus, but you are using two 8-bit EPROMs to hold your firmware, you can generate the even and odd images by using the **−SPlit** filter. Assuming your firmware is in the *firmware.hex* file, use the following:

```
srec_cat firmware.hex −split 2 0 −o firmware.even.hex
srec_cat firmware.hex −split 2 1 −o firmware.odd.hex
```

This will result in the two necessary EPROM images. Note that the output addresses are divided by the split multiple, so if your EPROM images are at a particular offset (say 0x10000, in the following example), you need to remove the offset, and then replace it...

```
srec_cat firmware.hex \
    −offset −0x10000 −split 2 0 \
    −offset 0x10000 −o firmware.even.hex
srec_cat firmware.hex \
    −offset −0x10000 −split 2 1 \
    −offset 0x10000 −o firmware.odd.hex
```

Note how the ability to apply multiple filters simplifies what would otherwise be a much longer script.

Striping

A second use for the **−SPlit** filter is memory striping.

You don't have to split into byte-wide parts, you can choose other sizes. It is common to want to convert 32-bit wide data into two set of 16-bit wide data.

```
srec_cat firmware.hex −split 4 0 2 −o firmware.01.hex
srec_cat firmware.hex −split 4 2 2 −o firmware.23.hex
```

This is relatively simple to understand, but you can use even wider stripes.

In this next example, the hardware requires that 512-byte blocks alternate between 4 EPROMs. Generating the 4 images would be done as follows:

```
srec_cat firmware.hex −split 0x800 0x000 0x200 −o firmware.0.hex
srec_cat firmware.hex −split 0x800 0x200 0x200 −o firmware.1.hex
srec_cat firmware.hex −split 0x800 0x400 0x200 −o firmware.2.hex
srec_cat firmware.hex −split 0x800 0x600 0x200 −o firmware.3.hex
```

Asymmetric Striping

A more peculiar example of striping is the Microchip dsPIC33F microcontroller, that has a weird memory storage pattern and they are able to store 3 bytes in an address that should only contain 2 bytes. The result is a hex file that has zero-filled the top byte (little-endian), and all addresses are doubled from what they are in the chip. Here is an example:

```
S1130000000102000405060008090A000C0D0E0098
S1130010101112001415160018191A001C1D1E00C8
S1130020202122002425260028292A002C2D2E00F8
S1130030303132003435360038393A003C3D3E0028
```

To get rid of the 00 padding bytes, leaving only the 3/4 significant bytes, you also use the split filter, with its additional *width* argument, like this:

```
srec_cat example.srec −split 4 0 3 −o no_dross.srec
```

This results in a file with the 00 padding bytes removed. It looks like this:

```
S1130000000102040506080 90A0C0D0E1011121451
S1130010151618191A1C1D1E2021222425262829EC
S11300202A2C2D2E30313234353638393A3C3D3E87
```

Notice how the addresses are 3/4 the size, as well. You can reverse this using the **−unsplit** and **−fill=0** filters.

Unsplit ING Images

The unsplit filter may be used to reverse the effects of the split filter. Note that the address range is expanded leaving holes between the stripes. By using all the stripes, the complete input is reassembled, without any holes.

```
srec_cat −o firmware.hex \
    firmware.even.hex −unsplit 2 0 \
    firmware.odd.hex  −unsplit 2 1
```

The above example reverses the previous 16-bit data bus example. In general, you unsplit with the same parameters that you split with.

FILLING THE BLANKS

Often EPROM load files will have "holes" in them, places where the compiler and linker did not put anything. For some purposes this is OK, and for other purposes something has to be done about the holes.

The Fill Filter

It is possible to fill the blanks where your data does not lie. The simplest example of this fills the entire EPROM:

```
srec_cat infile −fill 0x00 0x200000 0x300000 −o outfile
```

This example fills the holes, if any, with zeros. You must specify a range − with a 32-bit address space, filling everything generates *huge* load files.

If you only want to fill the gaps in your data, and don't want to fill the entire EPROM, try:

```
srec_cat infile -fill 0x00 -over infile -o outfile
```

This example demonstrates the fact that wherever an address range may be specified, the **−over** and **−within** options may be used.

Unfilling the Blanks

It is common to need to "unfill" an EPROM image after you read it out of a chip. Usually, it will have had all the holes filled with 0xFF (areas of the EPROM you don't program show as 0xFF when you read them back).

To get rid of all the 0xFF bytes in the data, use this filter:

```
srec_cat infile -unfill 0xFF -o outfile
```

This will get rid of *all* the 0xFF bytes, including the ones you actually wanted in there. There are two ways to deal with this. First, you can specify a minimum run length to the un-fill:

```
srec_cat infile -unfill 0xFF 5 -o outfile
```

This says that runs of 1 to 4 bytes of 0xFF are OK, and that a hole should only be created for runs of 5 or more 0xFF bytes in a row. The second method is to re-fill over the intermediate gaps:

```
srec_cat outfile -fill 0xFF -over outfile \
    -o outfile2
```

Which method you choose depends on your needs, and the shape of the data in your EPROM. You may need to combine both techniques.

Address Range Padding

Some data formats are 16 bits wide, and automatically fill with 0xFF bytes if it is necessary to fill out the other half of a word which is not in the data. If you need to fill with a different value, you can use a command like this:

```
srec_cat infile -fill 0x0A \
    -within infile -range-padding 2 \
    -o outfile
```

This gives the fill filter an address range calculated from details of the input file. The address range is all the address ranges covered by data in the *infile*, extended downwards (if necessary) at the start of each sub-range to a 2 byte multiple and extended upwards (if necessary) at the end of each sub-range to a 2 byte multiple. This also works for larger multiples, like 1kB page boundaries of flash chips. This address range padding works anywhere an address range is required.

Fill with Copyright

It is possible to fill unused portions of your EPROM with a repeating copyright message. Anyone trying to reverse engineer your EPROMs is going to see the copyright notice in their hex editor.

This is accomplished with two input sources, one from a data file, and one which is generated on-the-fly.

```
srec_cat infile \
    -generate '(' 0 0x100000 -minus -within infile ')' \
        -repeat-string 'Copyright (C) 1812 Tchaikovsky.  ' \
    -o outfile
```

Notice the address range for the data generation: it takes the address range of your EPROM, in this case 1MB starting from 0, and subtracts from it the address ranges used by the input file.

If you want to script this with the current year (because 1812 is a bit out of date) use the shell's output substitution (back ticks) ability:

```
srec_cat infile \
```

```
-generate '(' 0 0x100000 -minus -within infile ')' \
    -repeat-string "Copyright (C) `date +%Y` Tchaikovsky.  " \
-o outfile
```

The string specified is repeated over and over again, until it has filled all the holes.

Obfuscating with Noise

Sometimes you want to fill your EPROM images with noise, to conceal where the real data stops and starts. You can do this with the **-random-fill** filter.

```
srec_cat infile -random-fill 0x200000 0x300000 \
    -o outfile
```

It works just like the **-fill** filter, but uses random numbers instead of a constant byte value.

Fill With 16-bit Words

When filling the image with a constant byte value doesn't work, and you need a constant 16-bit word value instead, use the **-repeat-data** generator, which takes an arbitrarily long sequence of bytes to use as the fill pattern:

```
srec_cat infile \
    -generator '(' 0x200000 0x300000 -minus -within infile ')' \
        -repeat-data 0x1B 0x08 \
    -o outfile
```

Notice how the generator's address range once again avoids the address ranges occupied by the *infile*'s data. You have to get the endian-ness right yourself.

INSERTING CONSTANT DATA

From time to time you will want to insert constant data, or data not produced by your compiler or assembler, into your EPROM load images.

Binary Means Literal

One simple way is to have the desired information in a file. To insert the file's contents literally, with no format interpretation, use the *binary* input format:

```
srec_cat infile -binary -o outfile
```

It will probably be necessary to use an *offset* filter to move the data to where you actually want it within the image:

```
srec_cat infile -binary -offset 0x1234 -o outfile
```

It is also possible to use the standard input as a data source, which lends itself to being scripted. For example, to insert the current date and time into an EPROM load file, you could use a pipe:

```
date | srec_cat - -bin -offset 0xFFE3 -o outfile
```

The special file name "-" means to read from the standard input. The output of the *date* command is always 29 characters long, and the offset shown will place it at the top of a 64KB EPROM image.

Repeating Once

The **Fill with Copyright** section, above, shows how to repeat a string over and over. We can use a single repeat to insert a string just once.

```
srec_cat -generate 0xFFE3 0x10000 -repeat-string "`date`" \
    -o outfile
```

Notice how the address range for the data generation exactly matches the length of the *date*(1) output size. You can, of course, add your input file to the above *srec_cat*(1) command to catenate your EPROM image together with the date and time.

Inserting A Long

Another possibility is to add the Subversion commit number to your EPROM image. In this example, we are inserting it a a 4-byte little-endian value at address 0x0008. The Subversion commit number is in the *$version* shell variable in this example:

```
srec_cat -generate 0x0008 0x000C -constant-l-e $version 4 \
    infile -exclude 0x0008 0x000C \
    -o outfile
```

Note that we use a filter to ensure there is a hole in the input where the version number goes, just in case the linker put something there.

DATA ABOUT THE DATA

It is possible to add a variety of data about the data to the output.

Checksums

The **-checksum-negative-big-endian** filter may be used to sum the data, and then insert the negative of the sum into the data. This has the effect of summing to zero when the checksum itself is summed across, provided the sum width matches the inserted value width.

```
srec_cat infile \
        -crop 0 0xFFFFFC \
        -random-fill 0 0xFFFFFC \
        -checksum-neg-b-e 0xFFFFFC 4 4 \
    -o outfile
```

In this example, we have an EPROM in the lowest megabyte of memory. The -crop filter ensures we are only summing the data within the EPROM, and not anywhere else. The -random-fill filter fills any holes left in the data with random values. Finally, the -checksum-neg-b-e filter inserts a 32 bit (4 byte) checksum in big-endian format in the last 4 bytes of the EPROM image. Naturally, there is a little-endian version of this filter as well.

Your embedded code can check the EPROM using C code similar to the following:

```
unsigned long *begin = (unsigned long *)0;
unsigned long *end = (unsigned long *)0x100000;
unsigned long sum = 0;
while (begin < end)
    sum += *begin++;
if (sum != 0)
{
    Oops
}
```

The **-checksum-bitnot-big-endian** filter is similar, except that summing over the checksum should yield a value of all-one-bits (–1). For example, using shorts rather than longs:

```
srec_cat infile \
        -crop 0 0xFFFFFE \
        -fill 0xCC 0x00000 0xFFFFFE \
        -checksum-neg-b-e 0xFFFFFE 2 2 \
    -o outfile
```

Assuming you chose the correct endian-ness filter, your embedded code can check the EPROM using C code similar to the following:

```
unsigned short *begin = (unsigned short *)0;
unsigned short *end = (unsigned short *)0x100000;
unsigned short sum = 0;
while (begin < end)
    sum += *begin++;
if (sum != 0xFFFF)
{
    Oops
}
```

There is also a **-checksum-positive-b-e** filter, and a matching little-endian filter, which inserts the simple

sum, and which would be checked in C using an equality test.

```
srec_cat infile \
        -crop 0 0xFFFFFF \
        -fill 0x00 0x00000 0xFFFFFF \
        -checksum-neg-b-e 0xFFFFFF 1 1 \
    -o outfile
```

Assuming you chose the correct endian-ness filter, your embedded code can check the EPROM using C code similar to the following:

```
unsigned char *begin = (unsigned char *)0;
unsigned char *end = (unsigned char *)0xFFFFF;
unsigned char sum = 0;
while (begin < end)
    sum += *begin++;
if (sum != *end)
{
    Oops
}
```

In the 8-bit case, it doesn't matter whether you use the big-endian or little-endian filter.

Quick Hex-Dump

You can look at the checksum of your data, by using the "hex-dump" output format. This is useful for looking at calculated values, or for debugging an *srec_cat*(1) command before immortalizing it in a script.

```
srec_cat infile                          \
        -crop 0 0x10000                  \
        -fill 0xFF 0x0000 0x10000        \
        -checksum-neg-b-e 0x10000 4      \
        -crop 0x10000 0x10004            \
    -o - -hex-dump
```

This command reads in the file, checksums the data and places the checksum at 0x10000, crops the result to contain only the checksum, and then prints the checksum on the standard output in a classical hexadecimal dump format. The special file name "-" means "the standard output" in this context.

Cyclic Redundancy Checks

The simple additive checksums have a number of theoretical limitations, to do with errors they can and can't detect. The CRC methods have fewer problems.

```
srec_cat infile                          \
        -crop 0 0xFFFFFC                 \
        -fill 0x00 0x00000 0xFFFFFC      \
        -crc32-b-e 0xFFFFFC              \
    -o outfile
```

In the above example, we have an EPROM in the lowest megabyte of memory. The **-crop** filter ensures we are only summing the data within the EPROM, and not anywhere else. The **-fill** filter fills any holes left in the data. Finally, the **-checksum-neg-b-e** filter inserts a 32 bit (4 byte) checksum in big-endian format in the last 4 bytes of the EPROM image. Naturally, there is a little-endian version of this filter as well.

The checksum is calculated using the industry standard 32-bit CRC. Because SRecord is open source, you can always read the source code to see how it works. There are many non-GPL versions of this code available on the Internet, and suitable for embedding in proprietary firmware.

There is also a 16-bit CRC available.

```
srec_cat infile                          \
        -crop 0 0xFFFFFE                 \
        -fill 0x00 0x00000 0xFFFFFE \
```

```
        -crc16-b-e 0xFFFFFE         \
    -o outfile
```

The checksum is calculated using the CCITT formula. Because SRecord is open source, you can always read the source code to see how it works. There are many non-GPL version of this code available on the Internet, and suitable for embedding in proprietary firmware.

You can look at the CRC of your data, by using the “hex-dump” output format.

```
srec_cat infile                      \
        -crop 0 0x10000              \
        -fill 0xFF 0x0000 0x10000    \
        -crc16-b-e 0x10000           \
        -crop 0x10000 0x10002        \
    -o - -hex-dump
```

This command reads in the file, calculates the CRC of the data and places the CRC at 0x10000, crops the result to contain only the CRC, and then prints the checksum on the standard output in a classical hexadecimal dump format.

Where Is My Data?

There are several properties of your EPROM image that you may wish to insert into the data.

```
srec_cat infile -minimum-b-e 0xFFFE 2 -o outfile
```

The above example inserts the minimum address of the data (*low water*) into the data, as two bytes in big-endian order at address 0xFFFE. This includes the minimum itself. If the data already contains bytes at the given address, you need to use an exclude filter. The number of bytes defaults to 4.

There is also a **-minimum-l-e** filter for inserting little-endian values, and two more filters called **-exclusive-minimum-b-e** and **-exclusive-minimum-l-e** that do not include the minimum itself in the calculation of the minimum data address.

```
srec_cat infile -maximum-b-e 0xFFFFFC 4 -o outfile
```

The above example inserts the maximum address of the data (*high water + 1*, just like address ranges) into the data, as four bytes in big-endian order at address 0xFFFFFC. This includes the maximum itself. If the data already contains bytes at the given address, you need to use an **-exclude** filter. The number of bytes defaults to 4.

There is also a **-maximum-l-e** filter for inserting little-endian values, and two more filters called **-exclusive-maximum-b-e** and **-exclusive-maximum-l-e** that do not include the maximum itself in the calculation of the maximum data address.

```
srec_cat infile -length-b-e 0xFFFFFC 4 -o outfile
```

The above example inserts the length of the data (*high water* + 1 – *low water*) into the data, as four bytes in big-endian order at address 0xFFFFFC. This includes the length itself. If the data already contains bytes at the length location, you need to use an **-exclude** filter. The number of bytes defaults to 4.

There is also a **-length-l-e** filter for inserting a little-endian length, and the **-exclusive-length-b-e** and **-exclusive-length-l-e** filters that do not include the length itself in the calculation.

What Format Is This?

You can obtain a variety of information about an EPROM load file by using the *srec_info*(1) command. For example:

```
$ srec_info example.srec
Format: Motorola S-Record
Header: "http://srecord.sourceforge.net/"
Execution Start Address: 00000000
Data:   0000 - 0122
        0456 - 0FFF
$
```

This example shows that the file is a Motorola S-Record. The text in the file header is printed, along with the execution start address. The final section shows the address ranges containing data (the upper bound of each subrange is *in*clusive, rather than the *ex*clusive form used on the command line.

```
$ srec_info some-weird-file.hex -guess
Format: Signetics
Data:   0000 - 0122
        0456 - 0FFF
$
```

The above example guesses the EPROM load file format. It isn't infallible but it usually gets it right. You can use **-guess** anywhere you would give an explicit format, but it tends to be slower and for that reason is not recommended. Also, for automated build systems, you want hard errors as early as possible; if a file isn't in the expected format, you want it to barf.

MANGLING THE DATA

It is possible to change the values of the data bytes in several ways.

`srec_cat` *infile* `-and 0xF0 -o` *outfile*

The above example performs a bit-wise AND of the data bytes with the 0xF0 mask. The addresses of records are unchanged. I can't actually think of a use for this filter.

`srec_cat` *infile* `-or 0x0F -o` *outfile*

The above example performs a bit-wise OR of the data bytes with the 0x0F bits. The addresses of records are unchanged. I can't actually think of a use for this filter.

`srec_cat` *infile* `-xor 0xA5 -o` *outfile*

The above example performs a bit-wise exclusive OR of the data bytes with the 0xA5 bits. The addresses of records are unchanged. You could use this to obfuscate the contents of your EPROM.

`srec_cat` *infile* `-not -o` *outfile*

The above example performs a bit-wise NOT of the data bytes. The addresses of records are unchanged. Security by obscurity?

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat -VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat -VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_info − information about EPROM load files

SYNOPSIS

srec_info [*option*...] *filename*...
srec_info −Help
srec_info −VERSion

DESCRIPTION

The *srec_info* program is used to obtain input about EPROM load files. It reads the files specified, and then presents statistics about them. These statistics include: the file header if any, the execution start address if any, and the address ranges covered by the data if any.

If there is binary data the header, it will be presented using the URL encoding that uses % followed by two hexadeimal characters. For example a backspace would be encoded as “%08”. (This is symmetric with the srec_cat --header opion).

INPUT FILE SPECIFICATIONS

Input may be qualified in two ways: you may specify a data file or a data generator. format and you may specify filters to apply to them. An input file specification looks like this:

data-file [*filter* ...]
data-generator [*filter* ...]

Data Files

Input from data files is specified by file name and format name. An input file specification looks like this:

filename [*format*][−ignore-checksums]

The default format is Motorola S-Record format, but *many* others are also understood.

Data Generators

It is also possible to generate data, rather than read it from a file. You may use a generator anywhere you could use a file. An input generator specification looks like this:

−GENerate *address-range* *−data-source*

Generators include random data and various forms of constant data.

Common Manual Page

See *srec_input*(1) for complete details of input specifiers. This description is in a separate manual page because it is common to more than one SRecord command.

OPTIONS

The following options are understood:

@*filename*

The named text file is read for additional command line arguments. Arguments are separated by white space (space, tab, newline, *etc*). There is no wildcard mechanism. There is no quoting mechanism. Comments, which start with ’**#**’ and extend to the end of the line, are ignored. Blank lines are ignored.

−Help

Provide some help with using the *srec_info* program.

−IGnore_Checksums

The **−IGnore-Checksums** option may be used to disable checksum validation of input files, for those formats which have checksums at all. Note that the checksum values are still read in and parsed (so it is still an error if they are missing) but their values are not checked. Used after an input file name, the option affects that file alone; used anywhere else on the command line, it applies to all following files.

−Enable_Sequence_Warnings

This option may be used to enable warnings about input files where the data records are not in strictly ascending address order. Only one warning is issued per input file. This is the default.

Note: the output of *srec_cat*(1) is always in this order.

Note: This option must be used *before* the input file. This is because if there are several files on the command line, each may need different settings. The setting remains in force until the next **−Disable_Sequence_Warnings** option.

−Disable_Sequence_Warnings
This option may be used to disable warnings about input files where the data records are not in strictly ascending address order.

Note: This option must be used *before* the offending input file. This is because if there are several files on the command line, each may need different settings. The setting remains in force until the next **−Ensable_Sequence_Warnings** option.

−redundant-bytes=*value*
Use this option to permit a file to contain redundant values for some memory locations. The default is for this condition to be a warning.

ignore
No warning or error is issued whena redundant settings are detected.

warning
A warning is issued when a redundant settings are observed, the warning includes the problematic address.

error
A fatal error is issued when a redundant settings are observed, the fatal error message includes the problematic address and byte value.

−contradictory-bytes=*value*
Use this option to permit a file to contain contradictory values for some memory locations. The last value in the input(s) will be used. The default is for this condition to be a fatal error.

ignore
No warning or error is issued when contradictory setting is detected.

warning
A warning is issued when a vontradictory settings are observed, the warning includes the problematic address, and values.

error
A fatal error is issued when contradictory settings are observed, the fatal error message includes the problematic address and byte values.

−VERSion
Print the version of the *srec_info* program being executed.

All other options will produce a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−help”, “−HEL” and “−h” are all interpreted to mean the **−Help** option. The argument “−hlp” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *srec_info* are long, this means ignoring the extra leading “−”. The “*−−option=value*” convention is also understood.

EXIT STATUS

The *srec_info* command will exit with a status of 1 on any error. The *srec_info* command will only exit with a status of 0 if there are no errors.

COPYRIGHT

srec_info version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_info* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_info −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_info −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_input − input file specifications

SYNOPSIS

srec_* *filename* [*format*]

DESCRIPTION

This manual page describes the input file specifications for the *srec_cat*(1), *srec_cmp*(1) and *srec_info*(1) commands.

Input files may be qualified in a number of ways: you may specify their format and you may specify filters to apply to them. An input file specification looks like this:

filename [*format*][−ignore-checksums][*filter* ...]

The *filename* may be specified as a file name, or the special name "−" which is understood to mean the standard input.

Grouping with Parentheses

There are some cases where operator precedence of the filters can be ambiguous. Input specifications may also be enclosed by **(** parentheses **)** to make grouping explicit. Remember that the parentheses must be separate words, *i.e.* surrounded by spaces, and they will need to be quoted to get them past the shell's interpretation of parentheses.

Those Option Names Sure Are Long

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−help", "−HEL" and "−h" are all interpreted to mean the **−Help** option. The argument "−hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *srec_input* are long, this means ignoring the extra leading "−". The "**−−***option***=***value*" convention is also understood.

File Formats

The *format* is specified by the argument *after* the file name. The format defaults to Motorola S-Record if not specified. The format specifiers are:

−Absolute_Object_Module_Format

This option says to use the Intel Absolute Object Module Format (AOMF) to read the file. (See *srec_aomf*(5) for a description of this file format.)

−Ascii_Hex

This option says to use the Ascii-Hex format to read the file. See *srec_ascii_hex*(5) for a description of this file format.

−Atmel_Generic

This option says to use the Atmel Generic format to read the file. See *srec_atmel_genetic*(5) for a description of this file format.

−Binary

This option says the file is a raw binary file, and should be read literally. (This option may also be written −Raw.) See *srec_binary*(5) for more information.

−B-Record

This option says to use the Freescale MC68EZ328 Dragonball bootstrap b-record format to read the file. See *srec_brecord*(5) for a description of this file format.

−COsmac
This option says to use the RCA Cosmac Elf format to read the file. See *srec_cosmac*(5) for a description of this file format.

−Dec_Binary
This option says to use the DEC Binary (XXDP) format to read the file. See *srec_dec_binary*(5) for a description of this file format.

−Elektor_Monitor52
This option says to use the EMON52 format to read the file. See *srec_emon52*(5) for a description of this file format.

−FAIrchild
This option says to use the Fairchild Fairbug format to read the file. See *srec_fairchild*(5) for a description of this file format.

−Fast_Load
This option says to use the LSI Logic Fast Load format to read the file. See *srec_fastload*(5) for a description of this file format.

−Formatted_Binary
This option says to use the Formatted Binary format to read the file. See *srec_formatted_binary*(5) for a description of this file format.

−Four_Packed_Code
This option says to use the FPC format to read the file. See *srec_fpc*(5) for a description of this file format.

−Guess This option may be used to ask the command to guess the input format. This is slower than specifying an explicit format, as it may open and scan and close the file a number of times.

−HEX_Dump
This option says to try to read a hexadecimal dump file, more or less in the style output by the same option. This is not an exact reverse mapping, because if there are ASCII equivalents on the right hand side, these may be confused for data bytes. Also, it doesn't understand white space representing holes in the data in the line.

−IDT This option says to the the IDT/sim binary format to read the file. See *srec_idt*(5) for a description of this file format.

−Intel This option says to use the Intel hex format to read the file. See *srec_intel*(5) for a description of this file format.

−INtel_HeX_16
This option says to use the Intel hex 16 (INHX16) format to read the file. See *srec_intel16*(5) for a description of this file format.

−LOGIsim
This format is read and written by the open source Logisim program. See *srec_logisim*(5) for more informatuion.

−Memory_Initialization_File
This option says to use the Memory Initialization File (MIF) format by Altera to read the file. See *srec_mif* (5) for a description of this file format.

−Mips_Flash_Big_Endian

−Mips_Flash_Little_Endian
These options say to use the MIPS Flash file format to read the file. See *srec_mips_flash* (5) for a description of this file format.

−MOS_Technologies
This option says to use the Mos Technologies format to read the file. See *srec_mos_tech*(5) for a description of this file format.

−Motorola [*width*]
This option says to use the Motorola S-Record format to read the file. (May be written **−S-Record** as well.) See *srec_motorola*(5) for a description of this file format.

The optional *width* argument describes the number of bytes which form each address multiple. For normal uses the default of one (1) byte is appropriate. Some systems with 16-bit or 32-bit targets mutilate the addresses in the file; this option will correct for that. Unlike most other parameters, this one cannot be guessed.

−MsBin This option says to use the Windows CE Binary Image Data Format to read the file. See *srec_msbin*(5) for a description of this file format.

−Needham_Hexadecimal
This option says to use the Needham Electronics ASCII file format to read the file. See *srec_needham*(5) for a description of this file format.

−Ohio_Scientific
This option says to use the Ohio Scientific format. See *srec_os65v*(5) for a description of this file format.

−PPB This option says to use the Stag Prom Programmer binary format. See *srec_ppb*(5) for a description of this file format.

−PPX This option says to use the Stag Prom Programmer hexadecimal format. See *srec_ppx*(5) for a description of this file format.

−SIGnetics
This option says to use the Signetics format. See *srec_spasm*(5) for a description of this file format.

−SPAsm
This is a synonym for the **−SPAsm_Big_Endian** option.

−SPAsm_Big_Endian
This option says to use the SPASM assembler output format (commonly used by PIC programmers). See *srec_spasm*(5) for a description of this file format.

−SPAsm_Little_Endian
This option says to use the SPASM assembler output format, but with the data the other way around.

−STewie
This option says to use the Stewie binary format to read the file. See *srec_stewie*(5) for a description of this file format.

−Tektronix
This option says to use the Tektronix hex format to read the file. See *srec_tektronix*(5) for a description of this file format.

−Tektronix_Extended
This option says to use the Tektronix extended hex format to read the file. See *srec_tektronix_extended*(5) for a description of this file format.

−Texas_Instruments_Tagged
This option says to use the Texas Instruments Tagged format to read the file. See *srec_ti_tagged*(5) for a description of this file format.

−Texas_Instruments_Tagged_16
This option says to use the Texas Instruments SDSMAC 320 format to read the file. See *srec_ti_tagged_16*(5) for a description of this file format.

−Texas_Instruments_TeXT
This option says to use the Texas Instruments TXT (MSP430) format to read the file. See *srec_ti_txt*(5) for a description of this file format.

−TRS80
: This option says to use the Radio Shack TRS-80 object file format to read the file. See *srec_trs80*(5) for a description of this file format.

−VMem
: This option says to use the Verilog VMEM format to read the file. See *srec_vmem*(5) for a description of this file format.

−WILson
: This option says to use the wilson format to read the file. See *srec_wilson*(5) for a description of this file format.

Ignore Checksums

The **−IGnore-Checksums** option may be used to disable checksum validation of input files, for those formats which have checksums at all. Note that the checksum values are still read in and parsed (so it is still an error if they are missing) but their values are not checked. Used after an input file name, the option affects that file alone; used anywhere else on the command line, it applies to all following files.

−redundant-bytes=*value*
: Use this option to permit a file to contain redundant values for some memory locations. The default is for this condition to be a warning.

  ignore
  : No warning or error is issued whena redundant settings are detected.

  warning
  : A warning is issued when a redundant settings are observed, the warning includes the problematic address.

  error
  : A fatal error is issued when a redundant settings are observed, the fatal error message includes the problematic address and byte value.

−contradictory-bytes=*value*
: Use this option to permit a file to contain contradictory values for some memory locations. The last value in the input(s) will be used. The default is for this condition to be a fatal error.

  ignore
  : No warning or error is issued when contradictory setting is detected.

  warning
  : A warning is issued when a vontradictory settings are observed, the warning includes the problematic address, and values.

  error
  : A fatal error is issued when contradictory settings are observed, the fatal error message includes the problematic address and byte values.

Generators

It is also possible to generate data, rather than read it from a file. You may use a generator anywhere you could use a file. An input generator specification looks like this:

−GENerate *address-range* **−***data-source*

The **−***data-source* may be one of the following:

−CONSTant *byte-value*
: This generator manufactures data with the given byte value of the the given address range. It is an error if the byte-value is not in the range 0..255.

  For example, to fill memory addresses 100..199 with newlines (0x0A), you could use a command like

```
srec_cat −generate 100 200 −constant 10 −o newlines.srec
```

This can, of course, be combined with data from files.

−REPeat_Data *byte-value...*
This generator manufactures data with the given byte values repeating over the the given address range. It is an error if any of the the byte-values are not in the range 0..255.

For example, to create a data region with 0xDE in the even bytes and 0xAD in the odd bytes, use a generator like this:

```
srec_cat −generate 0x1000 0x2000 −repeat-data 0xDE 0xAD
```

The repeat boundaries are aligned with the base of the address range, modulo the number of bytes.

−REPeat_String *text*
This generator is almost identical to −repeat-data except that the data to be repeated is the text of the given string.

For example, to fill the holes in an EPROM image *eprom.srec* with the text “Copyright (C) 1812 Tchaikovsky”, combine a generator and an −exclude filter, such as the command

If you need to inject binary data into the string (e.g. a terminating NUL character), use the URL encoding that uses % followed by two hexadeimal characters. For example a backspace would be encoded as “%08”.

```
srec_cat eprom.srec \
    −generate 0 0x100000 \
        −repeat-string 'Copyright (C) 1812 Tchaikovsky. ' \
        −exclude −within eprom.srec \
    −o eprom.filled.srec
```

The thing to note is that we have two data sources: the *eprom.srec* file, and generated data over an address range which covers first megabyte of memory but excluding areas covered by the *eprom.srec* data.

−CONSTant_Little_Endian *value width*
This generator manufactures data with the given numeric value, of a given byte width, in little-endian byte order. It is an error if the given value does not fit into the given byte width. It will repeat over and over within the address range range.

For example, to insert a subversion commit number into 4 bytes at 0x0008..0x000B you would use a command like

```
srec_cat −generate 8 12 −constant-l-e $VERSION 4 \
    −o version.srec
```

This generator is a convenience wrapper around the **−REPeat_Data** generator. It can, of course, be combined with data from files.

−CONSTant_Big_Endian *value width*
As above, but using big-endian byte ordering.

Anything else will result in an error.

Input Filters
You may specify zero or more *filters* to be applied. Filters are applied in the order the user specifies.

−Adler_16_Big_Endian *address*
This filter may be used to insert an “Adler” 16-bit checksum of the data into the data. Two bytes, big-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

Note: If you have holes in your data, you will get a different Adler checksum than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the Adler checksum filters. You will

receive a warning if the data presented for Adler checksum has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

http://en.wikipedia.org/wiki/Adler-32

−Adler_16_Little_Endian *address*

This filter may be used to insert an Adler 16-bit checksum of the data into the data. Two bytes, in little-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

Note: If you have holes in your data, you will get a different Adler checksum than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the Adler filters. You will receive a warning if the data presented for Adler checksum has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

http://en.wikipedia.org/wiki/Adler-32

−Adler_32_Big_Endian *address*

This filter may be used to insert a Adler 32-bit checksum of the data into the data. Four bytes, big-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

Note: If you have holes in your data, you will get a different Adler checksum than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the Adler checksum filters. You will receive a warning if the data presented for Adler checksum has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

http://en.wikipedia.org/wiki/Adler-32

−Adler_32_Little_Endian *address*

This filter may be used to insert a Adler 32-bit checksum of the data into the data. Four bytes, in little-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

Note: If you have holes in your data, you will get a different Adler checksum than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the Adler checksum filters. You will receive a warning if the data presented for Adler checksum has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

http://en.wikipedia.org/wiki/Adler-32

−AND *value*

This filter may be used to bit-wise AND a *value* to every data byte. This is useful if you need to clear bits. Only existing data is altered, no holes are filled.

−Bit_Reverse [*width*]

This filter may be used to reverse the order of the bits in each data byte. By specifying a width (in bytes) it is possible to reverse the order multi-byte values; this is implemented using the byte-swap filter.

−Byte_Swap [*width*]
This filter may be used to swap pairs of odd and even bytes. By specifying a width (in bytes) it is possible to reverse the order of 4 and 8 bytes, the default is 2 bytes. (Widths in excess of 8 are assumed to be number of bits.) It is not possible to swap non-power-of-two addresses. To change the alignment, use the offset filter before and after.

−Checksum_BitNot_Big_Endian *address* [*nbytes* [*width*]]
This filter may be used to insert the one's complement checksum of the data into the data, most significant byte first. The data is literally summed; if there are duplicate bytes, this will produce an incorrect result, if there are holes, it will be as if they were filled with zeros. If the data already contains bytes at the checksum location, you need to use an exclude filter, or this will generate errors. You need to apply and crop or fill filters before this filter. The value will be written with the most significant byte first. The number of bytes of resulting checksum defaults to 4. The width (the width in bytes of the values being summed) defaults to 1.

−Checksum_BitNot_Little_Endian *address* [*nbytes* [*width*]]
This filter may be used to insert the one's complement (bitnot) checksum of the data into the data, least significant byte first. Otherwise similar to the above.

−Checksum_Negative_Big_Endian *address* [*nbytes* [*width*]]
This filter may be used to insert the two's complement (negative) checksum of the data into the data. Otherwise similar to the above.

−Checksum_Negative_Little_Endian *address* [*nbytes* [*width*]]
This filter may be used to insert the two's complement (negative) checksum of the data into the data. Otherwise similar to the above.

−Checksum_Positive_Big_Endian *address* [*nbytes* [*width*]]
This filter may be used to insert the simple checksum of the data into the data. Otherwise similar to the above.

−Checksum_Positive_Little_Endian *address* [*nbytes* [*width*]]
This filter may be used to insert the simple checksum of the data into the data. Otherwise similar to the above.

−CRC16_Big_Endian *address* [*modifier*...]
This filter may be used to insert an industry standard 16-bit CRC checksum of the data into the data. Two bytes, big-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

The following additional modifiers are understood:

number Set the polynomial to be used to the given number.

−POLYnomial *name*
This option may be used to set the CRC polynomial to be used, by name. The known names include:

ibm	0x8005
ansi	0x8005
ccitt	0x1021
t10-dif	0x8bb7
dnp	0x3d65
dect	0x0589

See http://en.wikipedia.org/wiki/Cyclic_redundancy_check for a table of names and values.

−Most_To_Least
The CRC calculation is performed with the most significant bit in each byte processed first, and then proceeding towards the least significant bit. This is the default.

−Least_To_Most
The CRC calculation is performed with the least significant bit in each byte processed first, and then proceeding towards the most significant bit.

−CCITT
The CCITT calculation is performed. The initial seed is 0xFFFF. This is the default.

−XMODEM
The alternate XMODEM calculation is performed. The initial seed is 0x0000.

−BROKEN
A common-but-broken calculation is performed (see note 2 below). The initial seed is 0x84CF.

−AUGment
The CRC is augmented by sixteen zero bits at the end of the calculation. This is the default.

−No-AUGment
The CRC is not augmented at the end of the calculation. This is less standard conforming, but some implementations do this.

Note: If you have holes in your data, you will get a different CRC than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the CRC filters. You will receive a warning if the data presented for CRC has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

Note 2: there are a great many CRC16 implementations out there, see http://www.joegeluso.com-/software/articles/ccitt.htm (now gone, reproduced at http://srecord.sourceforge.net-/crc16−ccitt.html) and “A painless guide to CRC error detection algorithms” http://www.repairfaq.org/filipg/LINK/F_crc_v3.html for more information. If all else fails, SRecord is open source software: read the SRecord source code. The CRC16 source code (found in the `srecord/crc16.cc` file of the distribution tarball) has a great many explanatory comments.

Please try all twelve combinations of the above options before reporting a bug in the CRC16 calculation.

−CRC16_Little_Endian *address* [*modifier*...]
The same as the **−CRC16_Big_Endian** filter, except in little-endian byte order.

−CRC32_Big_Endian *address* [*modifier*...]
This filter may be used to insert an industry standard 32-bit CRC checksum of the data into the data. Four bytes, big-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input). See also the note about holes, above.

The following additional modifiers are understood:

−CCITT
The CCITT calculation is performed. The initial seed is all one bits. This is the default.

−XMODEM
An alternate XMODEM-style calculation is performed. The initial seed is all zero bits.

−CRC32_Little_Endian *address*
The same as the **−CRC32_Big_Endian** filter, except in little-endian byte order.

−Crop *address-range*
This filter may be used to isolate a section of data, and discard the rest.

−Exclude *address-range*
This filter may be used to exclude a section of data, and keep the rest. The is the logical complement of the **−Crop** filter.

−Exclusive_Length_Big_Endian *address* [*nbytes* [*width*]]
The same as the **−Length_Big_Endian** filter, except that the result does **not** include the length itself.

−Exclusive_Length_Little_Endian *address* [*nbytes* [*width*]]
The same as the **−Length_Little_Endian** filter, except that the result does **not** include the length itself.

−Exclusive_MAXimum_Big_Endian *address* [*nbytes*]
The same as the **−MAXimum_Big_Endian** filter, except that the result does **not** include the maximum itself.

−Exclusive_MAXimum_Little_Endian *address* [*nbytes*]
The same as the **−MAXimum_Little_Endian** filter, except that the result does **not** include the maximum itself.

−Exclusive_MINimum_Big_Endian *address* [*nbytes*]
The same as the **−MINimum_Big_Endian** filter, except that the result does **not** include the minimum itself.

−Exclusive_MINimum_Little_Endian *address* [*nbytes*]
The same as the **−MINimum_Little_Endian** filter, except that the result does **not** include the minimum itself.

−eXclusive-OR *value*
This filter may be used to bit-wise XOR a *value* to every data byte. This is useful if you need to invert bits. Only existing data is altered, no holes are filled.

−Fill *value address-range*
This filter may be used to fill any gaps in the data with bytes equal to *value*. The fill will only occur in the address range given.

−Fletcher_16_Big_Endian *address* [*sum1 sum2* [*answer*]]
This filter may be used to insert an Fletcher 16-bit checksum of the data into the data. Two bytes, big-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

Note: If you have holes in your data, you will get a different Fletcher checksum than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the Fletcher checksum filters. You will receive a warning if the data presented for Fletcher checksum has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

http://en.wikipedia.org/wiki/Fletcher%27s_checksum

It is possible to select seed values for *sum1* and *sum2* in the algorithm, by adding seed values on the command line. They each default to 0xFF if not explicitly stated. The default values (0) means that an empty EPROM (all 0x00 or all 0xFF) will sum to zero; by changing the seeds, an empty EPROM will always fail.

The third optional argument is the desired sum, when the checksum itself is summed. A common value is 0x0000, placed in the last two bytes of an EPROM, so that the Fletcher 16 checksum of the EPROM is exactly 0x0000. No manipulation of the final value is performed if this value if

not specified.

−Fletcher_16_Little_Endian *address*

This filter may be used to insert an Fletcher 16-bit checksum of the data into the data. Two bytes, in little-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

Note: If you have holes in your data, you will get a different Fletcher checksum than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the Fletcher filters. You will receive a warning if the data presented for Fletcher checksum has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

http://en.wikipedia.org/wiki/Fletcher%27s_checksum

−Fletcher_32_Big_Endian *address*

This filter may be used to insert a Fletcher 32-bit checksum of the data into the data. Four bytes, big-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

Note: If you have holes in your data, you will get a different Fletcher checksum than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the Fletcher checksum filters. You will receive a warning if the data presented for Fletcher checksum has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

http://en.wikipedia.org/wiki/Fletcher%27s_checksum

−Fletcher_32_Little_Endian *address*

This filter may be used to insert a Fletcher 32-bit checksum of the data into the data. Four bytes, in little-endian order, are inserted at the address given. Holes in the input data are ignored. Bytes are processed in ascending address order (*not* in the order they appear in the input).

Note: If you have holes in your data, you will get a different Fletcher checksum than if there were no holes. This is important because the in-memory EPROM image will not have holes. You almost always want to use the **−fill** filter before any of the Fletcher checksum filters. You will receive a warning if the data presented for Fletcher checksum has holes.

You should also be aware that the lower and upper bounds of your data may not be the same as the lower and upper bounds of your EPROM. This is another reason to use the **−fill** filter, because it will establish the data across the full EPROM address range.

http://en.wikipedia.org/wiki/Fletcher%27s_checksum

−Length_Big_Endian *address* [*nbytes* [*width*]]

This filter may be used to insert the length of the data (high water minus low water) into the data. This includes the length itself. If the data already contains bytes at the length location, you need to use an exclude filter, or this will generate errors. The value will be written with the most significant byte first. The number of bytes defaults to 4. The width defaults to 1, and is divided into the actual length, thus you can insert the width in units of words (2) or longs (4).

−Length_Little_Endian *address* [*nbytes* [*width*]]

The same as the **−Length_Big_Endian** filter, except the value will be written with the least significant byte first.

−MAXimum_Big_Endian *address* [*nbytes*]
This filter may be used to insert the maximum address of the data (high water + 1) into the data. This includes the maximum itself. If the data already contains bytes at the given address, you need to use an exclude filter, or this will generate errors. The value will be written with the most significant byte first. The number of bytes defaults to 4.

−MAXimum_Little_Endian *address* [*nbytes*]
The same as the **−MAXimum_Big_Endian** filter, except the value will be written with the least significant byte first.

−Message_Digest_5 *address*
This filter may be used to insert a 16 byte MD5 hash into the data, at the address given.

−MINimum_Big_Endian *address* [*nbytes*]
This filter may be used to insert the minimum address of the data (low water) into the data. This includes the minimum itself. If the data already contains bytes at the given address, you need to use an exclude filter, or this will generate errors. The value will be written with the most significant byte first. The number of bytes defaults to 4.

−MINimum_Little_Endian *address* [*nbytes*]
The same as the **−MINimum_Big_Endian** filter, except the value will be written with the least significant byte first.

−NOT This filter may be used to bit-wise NOT the value of every data byte. This is useful if you need to invert the data. Only existing data is altered, no holes are filled.

−OFfset *nbytes*
This filter may be used to offset the addresses by the given number of bytes. No data is lost, the addresses will wrap around in 32 bits, if necessary. You may use negative numbers for the offset, if you wish to move data lower in memory.

Please note: the execution start address is a different concept than the first address in memory of your data. If you want to change where your monitor will start executing, use the **−execution-start-address** option (*srec_cat*(1) only).

−OR *value*
This filter may be used to bit-wise OR a *value* to every data byte. This is useful if you need to set bits. Only existing data is altered, no holes are filled.

−Random_Fill *address-range*
This filter may be used to fill any gaps in the data with random bytes. The fill will only occur in the address range given.

−Ripe_Message_Digest_160 *address*
This filter may be used to insert an RMD160 hash into the data.

−Secure_Hash_Algorithm_1 *address*
This filter may be used to insert a 20 byte SHA1 hash into the data, at the address given.

−Secure_Hash_Algorithm_224 *address*
This filter may be used to insert a 28 byte SHA224 hash into the data, at the address given. See Change Notice 1 for FIPS 180-2 for the specification.

−Secure_Hash_Algorithm_256 *address*
This filter may be used to insert a 32 byte SHA256 hash into the data, at the address given. See FIPS 180-2 for the specification.

−Secure_Hash_Algorithm_384 *address*
This filter may be used to insert a 48 byte SHA384 hash into the data, at the address given. See FIPS 180-2 for the specification.

−Secure_Hash_Algorithm_512 *address*
This filter may be used to insert a 64 byte SHA512 hash into the data, at the address given. See FIPS 180-2 for the specification.

−SPlit *multiple* [*offset* [*width*]]
This filter may be used to split the input into a subset of the data, and compress the address range so as to leave no gaps. This useful for wide data buses and memory striping. The *multiple* is the bytes multiple to split over, the *offset* is the byte offset into this range (defaults to 0), the *width* is the number of bytes to extract (defaults to 1) within the multiple. In order to leave no gaps, the output addresses are (*width* / *multiple*) times the input addresses.

−STM32 *address*
This is a synonym for the **−STM32_Little_Endian** filter.

−STM32_Little_Endian *address*

−STM32_Big_Endian *address*
These filters many be use to generate the CRC used by the hardware CRC unit on the STM32 series of ARM MPUs. The algorithm used by the STM32 hardware unit is just a CRC32 with a different polynomial and word-fed instead of byte-fed.

The *address* is where to place the 4-byte STM32 CRC.

The CRC used is documented in “RM0041, STM32F100xx reference manual”, page 46, chapter “CRC Calculation Unit”, which can be found at http://www.st.com/internet/mcu/product/216844.jsp

−TIGer *address*
This filter may be used to insert a 24 byte TIGER/192 hash into the data at the address given.

−UnFill *value* [*min-run-length*]
This filter may be used to create gaps in the data with bytes equal to *value*. You can think of it as reversing the effects of the **−Fill** filter. The gaps will only be created if the are at least *min-run-length* bytes in a row (defaults to 1).

−Un_SPlit *multiple* [*offset* [*width*]]
This filter may be used to reverse the effects of the split filter. The arguments are identical. Note that the address range is expanded (*multiple* / *width*) times, leaving holes between the stripes.

−WHIrlpool *address*
This filter may be used to insert a 64 byte WHIRLPOOL hash into the data, at the address given.

Address Ranges
There are eight ways to specify an address range:

minimum maximum
If you specify two number on the command line (decimal, octal and hexadecimal are understood, using the C conventions) this is an explicit address range. The minimum is inclusive, the maximum is exclusive (one more than the last address). If the maximum is given as zero then the range extends to the end of the address space.

−Within *input-specification*
This says to use the specified input file as a mask. The range includes all the places the specified input has data, and holes where it has holes. The input specification need not be just a file name, it may be anything any other input specification can be.

See also the **−over** option for a discussion on operator precedence.

−OVER *input-specification*
This says to use the specified input file as a mask. The range extends from the minimum to the maximum address used by the input, without any holes, even if the input has holes. The input specification need not be just a file name, it may be anything any other input specification can be.

You may need to enclose *input-specification* in parentheses to make sure it can’t misinterpret

which arguments go with which input specification. This is particularly important when a filter is to follow. For example

filename −fill 0 −over *filename2* −swap-bytes

groups as

filename −fill 0 −over '(' *filename2* −swap-bytes ')'

when what you actually wanted was

'(' *filename* −fill 0 −over *filename2* ')' −swap-bytes

The command line expression parsing tends to be "greedy" (or right associative) rather than conservative (or left associative).

address-range **−RAnge-PADding** *number*

It is also possible to pad ranges to be whole aligned multiples of the given number. For example

input-file −fill 0xFF −within *input-file* −range-pad 512

will fill the *input-file* so that it consists of whole 512-byte blocks, aligned on 512 byte boundaries. Any large holes in the data will also be multiples of 512 bytes, though they may have been shrunk as blocks before and after are padded.

This operator has the same precedence as the explicit union operator.

address-range **−INTERsect** *address-range*

You can intersect two address ranges to produce a smaller address range. The intersection operator has higher precedence than the implicit union operator (evaluated left to right).

address-range **−UNIon** *address-range*

You can union two address ranges to produce a larger address range. The union operator has lower precedence than the intersection operator (evaluated left to right).

address-range **−DIFference** *address-range*

You can difference two address ranges to produce a smaller address range. The result is the left hand range with all of the right hand range removed. The difference operator has the same precedence as the implicit union operator (evaluated left to right).

address-range address-range

In addition, all of these methods may be used, and used more than once, and the results will be combined (implicit union operator, same precedence as explicit union operator).

Calculated Values

Most of the places above where a number is expected, you may supply one of the following:

− *value*

The value of this expression is the negative of the expression argument. Note the **space** between the minus sign and its argument: this space is mandatory.

```
srec_cat in.srec −offset – −minimum-addr in.srec −o
out.srec
```

This example shows how to move data to the base of memory.

(*value* **)**

You may use parentheses for grouping. When using parentheses, they must each be a separate command line argument, they can't be within the text of the preceding or following option, and you will need to quote them to get them past the shell, such as `'('` and `')'`.

−MINimum-Address *input-specification*

This inserts the minimum address of the specified input file. The input specification need not be just a file name, it may be anything any other input specification can be.

See also the **−over** option for a discussion on operator precedence.

−MAXimum-Address *input-specification*

This inserts the maximum address of the specified input file, plus one. The input specification need not be just a file name, it may be anything any other input specification can be.

See also the **−over** option for a discussion on operator precedence.

−Length *input-specification*
This inserts the length of the address range in the specified input file, ignoring any holes. The input specification need not be just a file name, it may be anything any other input specification can be.

See also the **−over** option for a discussion on operator precedence.

For example, the **−OVER** *input-specification* option can be thought of as short-hand for **’(’ −min** *file* **−max** *file* **’)’**, except that it is much easier to type, and also more efficient.

In addition, calculated values may optionally be rounded in one of three ways:

value **−Round_Down** *number*
The *value* is rounded down to the the largest integer smaller than or equal to a whole multiple of the *number*.

value **−Round_Nearest** *number*
The *value* is rounded to the the nearest whole multiple of the *number*.

value **−Round_Up** *number*
The *value* is rounded up to the the smallest integer larger than or equal to a whole multiple of the *number*.

When using parentheses, they must each be a separate command line argument, they can’t be within the text of the preceding or following option, and you will need to quote them to get them past the shell, as `’(’` and `’)’`.

COPYRIGHT

srec_input version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_input* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_input −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_input −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program − to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product,

and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing

the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to

infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF

SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

> *one line to give the program’s name and a brief idea of what it does.*
> Copyright (C) *year name of author*
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author>
> This program comes with ABSOLUTELY NO WARRANTY; for details type “show w”. This is free software, and you are welcome to redistribute it under certain conditions; type “show c” for details.

The hypothetical commands “show w” and “show c” should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

NAME

srecord − library to manipulate EPROM load files

SYNOPSIS

#include <srecord/*name*.h>

cc ... −lsrecord

DESCRIPTION

The srecord sjared library may be used to add all of the EPROM file formats and filters to your own projects.

The full documentation for the shared library is generated by Doxygen from the source files, and is available on the Internet at
http://srecord.sourceforge.net/srecord/index.html

COPYRIGHT

srecord version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srecord* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srecord −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srecord −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

LGPG − GNU Lesser General Public License

DESCRIPTION

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d)

Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user’s computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU

Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

NAME

srec_aomf − Intel Absolute Object Module Format

DESCRIPTION

The Absolute Object Module Format (AOMF) is a subset of the 8051 OMF. The structure of an absolute object file (the order of the records in it) is similar to that of a relocatable object file. There are three main differences: the first is that an absolute object file contains one module only, the second is that not all the records can appear in the absolute file and the third is that the records can contain only absolute information.

Generic Record Format

Each record starts with a record type which indicates the type of the record, and record length which contain the number of bytes in the record exclusive of the first two fields. The record ends with a checksum byte which contains the 2s complement of the sum (modulo 256) of all other bytes in the record. Therefore the sum (modulo 256) of all bytes in the record is zero.

The record length includes the payload and checksum fields, but excludes the type and length fields.

All 16-bit fields are little-endian.

REC TYP 8 bits	Record Length 16 bits	Payload	CHK SUM 8 bits

Here are some of the relevant record types:

0x01	Scope Definition Record
0x02	Module Start Record
0x04	Module End Record
0x06	Content Record
0x0E	Segment Definition Record
0x12	Debug Items Record
0x16	Public Definition Record
0x18	External Definition Record

Names are not stored as C strings. Names are stored as a length byte followed by the contents.

Structure

An AOMF file consists of a module header record (0x02), followed by one or more content (0x06), scope (0x01) or debug (0x12) records, and ends in a module end record (0x04).

The records with the following types are extraneous (they may appear in the file but are ignored): 0x0E, 0x16 and 0x18 (definition records). All records which are not part of the AOMF and are not extraneous are considered erroneous.

Module Header Record

REC TYP 0x02	Record Length 16 bits	Module Name	TRN ID 8 bits	zero 8 bits	CHK SUM 8 bits

Each module must starts with a module header record. It is used to identify the module for the RL51 and other future processors of 8051 object files. In addition to the Module Name the record contains:

TRN ID The byte identifies the program which has generated this module:

0xFD	ASM51
0xFE	PL/M−51
0xFF	RL51.

Module End Record

REC TYP 0x04	Record Length 16 bits	Module Name	zero 16 bits	REG MSK 8 bits	zero 8 bits	CHK SUM 8 bits

The record ends the module sequence and contains the following information: characteristics

MODULE NAME
: The name of the module is given here for a consistency check. It must match the name given in the Module Header Record.

REGISTER MASK (REG MSK)
: The field contains a bit for each of the four register banks. Each bit, when set specifies that the corresponding bank is used by the module:

Bit 0 (the least significant bit)
: bank #0.

Bit 1 bank #1.

Bit 2 bank #2.

Bit 3 bank #3.

Content Record

REC TYP 0x06	Record Length 16 bits	SEG ID 8 bits	Offset 16 bits	DATA	CHK SUM 8 bits

This record provides one or more bytes of contiguous data, from which a portion of a memory image may be constructed.

SEG ID This field must be zero.

OFFSET
: Gives the absolute address of the first byte of data in the record, within the CODE address space.

DATA A sequence of data bytes to be loaded from OFFSET to OFFSET+RECORDLENGTH−5.

Size Multiplier

In general, raw binary data will expand in sized by approximately 1.02 times when represented with this format.

SOURCE

http://www.intel.com/design/mcs96/swsup/omf96_pi.pdf
ftp://download.intel.com/design/mcs51/SWSUP/omf51.exe (zip archive)
http://www.elsist.net/WebSite/ftp/various/OMF51EPS.pdf

COPYRIGHT

srec_cat version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_ascii_hex − Ascii-Hex file format

DESCRIPTION

This format is also known as the *Ascii-Space-Hex* or *Ascii-Hex-Space* format. If you know who invented this format, please let me know. If you have a better or more complete description, I'd like to know that, too.

The file starts with a start-of-text (STX or Control-B) character (0x02). Everything before the STX is ignored.

Each data byte is represented as 2 hexadecimal characters, followed by an "execution character". The default execution character is a space, although many programs which write this format omit the space character immediately preceding end-of-line.

The address for data bytes is set by using a sequence of `$A`*`nnnn`*`,` characters, where *nnnn* is the 4-character ascii representation of the address. The comma is required. There is no need for an address record unless there are gaps. Implicitly, the file starts a address 0 if no address is set before the first data byte.

The file ends with an end-of-text (ETX or Control-C) character (0x03). Everything following the ETX is ignored.

It is also possible to specify a running 16-bit checksum using a sequence of `$S`*`nnnn`*`,` characters, although this usually appears *after* the ETX character and is thus often ignored.

Variant Forms

In addition to a space character, the execution character can also be percent (%) called "ascii-hex-percent" format, apostrophe (') or comma (,) called "ascii-hex-comma" format. The file must use the same execution character throughout.

If the execution character is a comma, the address and checksum commands are terminated by a dot (.) rather than a comma (,).

Size Multiplier

In general, binary data will expand in sized by approximately 3.0 times when represented with this format.

EXAMPLE

Here is an example ascii-hex file. It contains the data "Hello, World[rq] to be loaded at address 0x1000.

```
^B $A1000,
48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 0A ^C
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_atmel_generic − Atmel Generic file format

DESCRIPTION

This format is the output of the Atmel AVR assembler. The file contains two columns of hexadecimal coded values. The first column is the 24-bit word address, the second column is the 16-bit word data. The columns are separated by a colon (‘:’) character.

By default, SRecord treats this is little-endian data (the least significant byte first). If you want big endian order, use the −atmel-generic-be argument instead.

Size Multiplier

In general, binary data will expand in sized by approximately 6.0 times when represented with this format (6.5 times in Windows).

EXAMPLE

Here is an example Atmel Generic file. It contains the data “Hello, World[rq] to be loaded at bytes address 0x0100 (but remember, the file contents are word addressed).

```
000080:4865
000081:6C6C
000082:6F2C
000083:2057
000084:6F72
000085:6C64
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_binary − binary file format

DESCRIPTION

It is possible to read and write binary files using *srec_cat*(1).

File Holes

A file hole is a portion of a regular file that contains NUL characters and is not stored in any data block on disk. Holes are a long-standing feature of Unix files. For instance, the following Unix command creates a file in which the first bytes are a hole:

```
$ echo −n "X" | dd of=/tmp/hole bs=1024 seek=6
$
```

Now `/tmp/hole` has 6,145 characters (6,144 NUL characters plus an X character), yet the file occupies just one data block on disk.

File holes were introduced to avoid wasting disk space. They are used extensively by database applications and, more generally, by all applications that perform hashing on files.

See http://www.oreilly.com/catalog/linuxkernel2/chapter/ch17.pdf for more information.

Reading

The size of binary files is taken from the size of the file on the file system. If the file has holes these will read as blocks of NUL (zero) data, as there is no elegant way to detect Unix file holes. In general, you probably want to use the **−unfill** filter to find and remove large swathes of zero bytes.

Writing

In producing a binary file, *srec_cat*(1) honours the address information and places the data into the binary file at the addresses specified in the hex file. This usually results on holes in the file. Sometimes alarmingly large file sizes are reported as a result.

If you are on a brain-dead operating system without file holes then there are going to be real data blocks containing real zero bytes, and consuming real amounts of disk space. Upgrade − I suggest Linux.

To make a file of the size you expect, use

```
srec_info foo.s19
```

to find the lowest address, then use

```
srec_cat foo.s19 −intel −offset −n −o foo.bin −binary
```

where *n* is the lowest address present in the `foo.s19` file, as reported by *srec_info*(1). The **negative** offset serves to move the data down to have an origin of zero.

SEE ALSO

srec_input(1)
for a description of the **−unfill** filter

srec_examples(1)
has a section about binary files, and ways of automagically offseting the data back to zero in a single command.

COPYRIGHT

SRrecord version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *SRrecord* program comes with ABSOLUTELY NO WARRANTY; for details use the '*SRrecord −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*SRrecord −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_brecord − Freescale MC68EZ328 Dragonball bootstrap record format

DESCRIPTION

This data format is understood by Freescale MC68EZ328 Dragonball series processors on their internal UART.

Lines

Each line contains hexadecimal data, each byte represented by two hexadecimal nybbles in upper case. Characters not in this set, but larger than 0x30 (e.g. lower case) will be ignored, less than 0x30 (e.g. CR or LF) are considered record terminators. Comments are problematic; don't try this at home.

Fields

Each line contains a 4-byte address (big endian), a 1-byte length-and-mode, and then data bytes as dictated by the length. There is **no** checksum. A zero length record is an execution start address record, non-zero length records are data.

1	2	3	4	5	6	7	8	9	10	...	*n*
Address								Length		Data	

The length-and-mode byte is formatted as follows:

7	6	5	4	3	2	1	0
Mode		R	Length				

Mode These bits are ignored by SRecord in input (00 = bytes, 01 = half words, 10 is reserved, 11 = long words). These bits are always zero on output by SRecord.

R This bit indicates a data read rather than a data write; SRecord does not accept input files with this bit set, and will not set it on output.

Length The length of the records data bytes. It does not include the address or length bytes. The maximum payload of a record is 31 bytes of data.

Size Multiplier

In general, binary data will expand in sized by at least 2.35 times when represented with this format.

EXAMPLE

Here is an example b-record format file. It contains the data "Hello, World" to be loaded at address 0.

```
000000000D48656C6C6F2C20576F726C640A
```

SEE ALSO

http://www.freescale.com/files/32bit/doc/ref_manual/MC68VZ328UM.pdf

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_coe − Xilinx Coefficient File Format

DESCRIPTION

The Xilinx Coefficient File Format has a general syntax of the form:

keyword = *value* ; *optional comment*
radix-keyword = *value* ; *optional comment*
data-keyword = *value, ..., value*;

There are numerous keywords, only the “memory_initialization_radix” and “memory_initialization_vector” keywords are used. The semicolons are mandatory.

Size Multiplier

Binary data stored in this format expand approximately 4 times (5 time on Windows).

See Also

http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/cgn_r_coe_file_syntax.htm

COPYRIGHT

srec_coe.5 version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_coe.5* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_coe.5 −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_coe.5 −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_cosmac − RCA Cosmac Elf file format

DESCRIPTION

This file takes the form of one or more RCA Cosmac Elf monitor commands, also known as the IDIOT/4 monitor. Only the change memory command (`!M`) is allowed.

The general form of the `!M` command takes the form

!M*aaaa dd ... dd*

The `!M` command writes data byte bytes (represented by character pairs *dd*) into successive memory locations, started at address *aaaa*. Spaces between data bytes are ignored.

Using the comma (`,`) line continuation character resumes from the next address in sequence.

!M*aaaa dd ... dd, dd ... dd*

Using the semicolon (`;`) line continuation character takes an address on the next line

!M*aaaa dd ... dd*; *aaaa dd ... dd*

It is also possible to have the semicolon immediately after the command.

!M; *aaaa dd ... dd*

All of these forms may be used in combination.

Size Multiplier

In general, binary data will expand in size by approximately 2.0 times when represented with this format.

EXAMPLE

Here is an example Cosmac file. It contains the data “Hello, World[rq] to be loaded at address 0x1000.

```
!M1000 48656C6C6F2C20576F726C640A
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_dec_binary − DEC Binary (XXDP) file format

DESCRIPTION

The DEC Binary (XXDP) format was used on the PDP 11 series machines. This is a binary format, and is not readable or editable with a text editor. The file consists of records of the form

type	length	address	...data...	checksum

The field are defined as follows:

type Two byte little-endian value. Must always be 1.

length Two byte little-endian value. This is the number of bytes in the data, plus six.

address Two byte little-endian value. This is the load address of the data.

data The data is simple raw bytes. There are (length-6) of them.

checksum
: The checcksum is a single byte. It is the negative of the simple summ of all the header and data bytes.

If the record length is exactly 6 (*i.e.* no data), this is the execution start address record, indicating the transfer address.

In addition there may be NUL padding characters between records. It is common for records to be padded so that they start on even byte boundaries. In the days of paper tape, it was common for the file to have many leading NULs, to generate blank leader on the tape.

Size Multiplier

In general, raw binary data will expand in sized by approximately 1.03 times when represented with this format.

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_emon52 − Elektor Monitor (EMON52) file format

DESCRIPTION

This format is used by the monitor EMON52, developed by the European electronics magazine Elektor (Elektuur in Holland). Elektor wouldn't be Elektor if they didn't try to reinvent the wheel. It's a mystery why they didn't use an existing format for the project. Only the Elektor Assembler will produce this file format, reducing the choice of development tools dramatically.

Records

All data lines are called records, and each record contains the following four fields:

cc	aaaa	:	dd ... dd	ssss

The field are defined as follows:

cc The byte count. A two digit hex value (1 byte), counting the actual data bytes in the record. The byte count is separated from the next field by a space.

aaaa The address field. A four hex digit (2 byte) number representing the first address to be used by this record.

: The address field and the data field are separated by a colon.

dd The actual data of this record. There can be 1 to 255 data bytes per record (see cc) All bytes in the record are separated from each other (and the checksum) by a space.

ssss Data Checksum, adding all bytes of the data line together, forming a 16 bit checksum. Covers only all the data bytes of this record.

Please note that there is no End Of File record defined.

Byte Count

The byte count cc counts the actual data bytes in the current record. Usually records have 16 data bytes. I don't know what the maximum number of data bytes is. It depends on the size of the data buffer in the EMON52.

Address Field

This is the address where the first data byte of the record should be stored. After storing that data byte, the address is incremented by 1 to point to the address for the next data byte of the record. And so on, until all data bytes are stored.

The address is represented by a 4 digit hex number (2 bytes), with the MSD first.

Data Field

The payload of the record is formed by the Data field. The number of data bytes expected is given by the Byte Count field.

Checksum

The checksum is a 16 bit result from adding all data bytes of the record together.

Size Multiplier

In general, binary data will expand in sized by approximately 3.8 times when represented with this format.

EXAMPLE

Here is an example of an EMON52 file:

```
10 0000:57 6F 77 21 20 44 69 64 20 79 6F 75 20 72 65 61 0564
10 0010:6C 6C 79 20 67 6F 20 74 68 72 6F 75 67 68 20 61 05E9
10 0020:6C 6C 20 74 68 69 73 20 74 72 6F 75 62 6C 65 20 05ED
10 0030:74 6F 20 72 65 61 64 20 74 68 69 73 20 73 74 72 05F0
04 0040:69 6E 67 21 015F
```

SEE ALSO

http://sbprojects.fol.nl/knowledge/fileformats/emon52.htm

AUTHOR

This man page was taken from the above Web page. It was written by San Bergmans <sanmail@bigfoot.com>

NAME

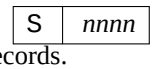
srec_fairchild − Fairchild Fairbug file format

DESCRIPTION

The Fairchild Fairbug format has 8-byte records. A file begins with an address record and ends with an end-of-file record.

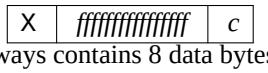
There are three record types in this file format.

Address records are of the form



indicating the address for the following data records.

Data records are of the form



Each data record begins with an X and always contains 8 data bytes. The *ff* characters are hexadecimal byte values (8 bytes). Each data byte is represented by 2 hexadecimal characters. The *c* character is a hex digit being the the nibble-sum of the data bytes. A 1-digit hexadecimal checksum follows the data in each data record. The checksum represents, in hexadecimal notation, the sum of the binary equivalents of the 16 digits in the record; the half carry from the fourth bit is ignored. The programmer ignores any character (except for address characters and the asterisk character, which terminates the data transfer) between a checksum and the start character of the next data record. This space can be used for comments.

The end-of-file record has the form



The last record consists of an asterisk only, which indicates the end of file.

Size Multiplier

In general, binary data will expand in sized by approximately 2.4 times when represented with this format.

EXAMPLE

Here is an example Fairchild Fairbug file. It contains the data “Hello, World[rq] to be loaded at address 0x1000. Notice how the last record is padded with 0xFF bytes.

```
S1000
X48656C6C6F2C2057C
X6F726C64210AFFFF3
*
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_fastload − LSI Logic Fast Load file format

DESCRIPTION

The FastLoad Format uses a compressed ASCII format that permits files to be downloaded in less than half the time taken for Motorola S-records.

The base-64 encoding used is "A-Za-z0−9,.". The data is encoded in groups of 4 characters (3 bytes, 24 bits).

The character '/' is used to introduce a special function. Special functions are:

A*nnnnnn*
: Defines an address.

B*nn* Define a single byte.

C*nnnn* Compare the checksums. The checksum is a simple positive 16-bit sum, of the data bytes only.

EAA Define the program's entry point. The address will be the current address as defined by the **A** command. (The *AA* number in this command is ignored.) This must be the last entry in the file.

KAA Clear the checksum. (The *AA* number in this command is ignored.)

S*name*,X
: Define a symbol. The address of the symbol will be the current address as defined by the **A** command.

Z*nn* Clear a number of bytes.

Size Multiplier

In general, binary data will expand in sized by approximately 1.4 times when represented with this format.

EXAMPLE

Here is an example LSI Logic Fast Load format file. It contains the data “Hello, World[rq] to be loaded at address 0.

```
/AAAA
SGVsbG8sIFdvcmxk/BAK/CARS/AAAA/EAA
```

COPYRIGHT

srec_cat version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_formatted_binary − Formatted Binary file format

DESCRIPTION

This is the PDP-11 paper tape format, described in the DEC-11-GGPC-D PDP-11 "Paper Tape Software Programming Handbook" 1972.

The file starts with a character sequence which appears as an arrow when punched on 8-hole paper tape.

0x08, 0x1C, 0x2A, 0x49, 0x08, 0x00

Then follows a byte count, encoded big-endian in the low 4 bits of the next 4 bytes. The high bits should be zero.

Then follows a 0xFF byte.

The data follows, as many bytes as specified in the header.

The trailer consists of the following bytes:

0x00, 0x00,

and then a 2-byte checksum (big-endian).

The alternate header sequence

0x08, 0x1C, 0x3E, 0x6B, 0x08, 0x00

is followed by an 8-nibble big-endian byte count.

Size Multiplier

In general, binary data will expand in sized very little when represented with this format.

EXAMPLE

Here is a hex dump of a formatted binary file containing the data "Hello, World!".

```
0000: 08 1C 2A 49 08 00 00 00  ..*I....
0008: 00 0E FF 48 65 6C 6C 6F  ...Hello
0010: 2C 20 57 6F 72 6C 64 21  , World!
0018: 0A 00 00 04 73           ....s
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_forth − FORTH file format

DESCRIPTION

This format can be read by FORTH interpreters

The file starts with HEX to set the number base.

Each line contains the address, the byte and a store command, either C! for RAM or EEC! for EEPROM

EXAMPLE

Here is an example srec[hy]forth file. It contains the data “Hello, World” to be loaded at address 0x1000.

```
HEX
48 1000 C!
65 1001 C!
6C 1002 C!
6C 1003 C!
6F 1004 C!
2C 1005 C!
20 1006 C!
57 1007 C!
6F 1008 C!
72 1009 C!
6C 100A C!
64 100B C!
0A 100C C!
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_fpc − four packed code file format

SYNOPSIS

All ASCII based file formats have one disadvantage in common: they all need more than double the amount of characters as opposed to the number of bytes to be sent. Address fields and checksums will add even more characters. So the shorter the records, the more characters have to be sent to get the file across.

The FPC format may be used to reduce the number of characters needed to send a file in ASCII format, although it still needs more characters than the actual bytes it sends. FPC stands for "Four Packed Code". The reduction is accomplished by squeezing 4 real bytes into 5 ASCII characters. In fact every ASCII character will be a digit in the base 85 number system. There aren't enough letters, digits and punctuation marks available to get 85 different characters, but if we use both upper case and lower case letters we will manage. This implies that the FPC *is case sensitive*, as opposed to all other ASCII based file formats.

Base 85

The numbering system is in base 85, and is somewhat hard to understand for us humans who are usually only familiar with base 10 numbers. Some of us understand base 2 and base 16 as well, but base 85 is for most people something new. Luckily we don't have to do any math with this number system. We just convert a 32 bit number into a 5 digit number in base 85. A 32 bit number has a range of 4,294,967,296, while a 5 digit number in base 85 has a range of 4,437,053,125, which is enough to do the trick. One drawback is that we always have to send multiples of 4 bytes, even if we actually want to send 1, 2 or 3 bytes. Unused bytes are padded with zeroes, and are discarded at the receiving end.

The digits of the base 85 numbering system start at %, which represents the value of 0. The highest value of a digit in base 85 is 84, and is represented by the character 'z'. If you want to check this with a normal ASCII table you will notice that we have used one character too many! Why? I don't know, but for some reason we have to skip the '*' character in the row. This means that after the ')' character follows the '+' character.

We can use normal number conversion algorithms to generate the FPC digits, with this tiny difference. We have to check whether the digit is going to be equal or larger than the ASCII value for '*'. If this is the case we have to increment the digit once to stay clear of the '*'. In base 85 MSD digits go first, like in all number systems!

The benefit of this all is hopefully clear. For every 4 bytes we only have to send 5 ASCII characters, as opposed to 8 characters for all other formats.

Records

Now we take a look at the the formatting of the FPC records. We look at the record at byte level, not at the actual base 85 encoded level. Only after formatting the FPC record at byte level we convert 4 bytes at a time to a 5 digit base 85 number. If we don't have enough bytes in the record to fill the last group of 5 digits we will add bytes with the value of 0 behind the record.

$	ss	cc	ffff	aaaaaaaa	dddddddd

The field are defined as:

$ Every line starts with the character $, all other characters are digits of base 85.

ss The checksum. A one byte 2's-complement checksum of all bytes of the record.

cc The byte-count. A one byte value, counting all the bytes in the record minus 4.

ffff Format code, a two byte value, defining the record type.

aaaaaaaa
The address field. A 4 byte number representing the first address of this record.

dddddddd
The actual data of this record.

Record Begin

Every record begins with the ASCII character "**$**". No spaces or tabs are allowed in a record. All other characters in the record are formed by groups of 5 digits of base 85.

Checksum field

This field is a one byte 2's-complement checksum of the entire record. To create the checksum make a one byte sum from all of the bytes from all of the fields of the record:

Then take the 2's-complement of this sum to create the final checksum. The 2's-complement is simply inverting all bits and then increment by 1 (or using the negative operator). Checking the checksum at the receivers end is done by adding all bytes together including the checksum itself, discarding all carries, and the result must be $00. The padding bytes at the end of the line, should they exist, should not be included in checksum. But it doesn't really matter if they are, for their influence will be 0 anyway.

Byte Count

The byte count **cc** counts the number of bytes in the current record minus 4. So only the number of address bytes and the data bytes are counted and not the first 4 bytes of the record (checksum, byte count and format flags). The byte count can have any value from 0 to 255.

Usually records have 32 data bytes. It is not recommended to send too many data bytes in a record for that may increase the transmission time in case of errors. Also avoid sending only a few data bytes per record, because the address overhead will be too heavy in comparison to the payload.

Format Flags

This is a 2 byte number, indicating what format is represented in this record. Only a few formats are available, so we actually waste 1 byte in each record for the sake of having multiples of 4 bytes.

Format code 0 means that the address field in this record is to be treated as the absolute address where the first data byte of the record should be stored.

Format code 1 means that the address field in this record is missing. Simply the last known address of the previous record +1 is used to store the first data byte. As if the FPC format wasn't fast enough already ;-)

Format code 2 means that the address field in this record is to be treated as a relative address. Relative to what is not really clear. The relative address will remain in effect until an absolute address is received again.

Address Field

The first data byte of the record is stored in the address specified by the Address field **aaaaaaaa**. After storing that data byte, the address is incremented by 1 to point to the address for the next data byte of the record. And so on, until all data bytes are stored.

The length of the address field is always 4 bytes, if present of course. So the address range for the FPC format is always 2**32.

If only the address field is given, without any data bytes, the address will be set as starting address for records that have no address field.

Addresses between records are non sequential. There may be gaps in the addressing or the address pointer may even point to lower addresses as before in the same file. But every time the sequence of addressing must be changed, a format 0 record must be used. Addressing within one single record *is* sequential of course.

Data Field

This field contains 0 or more data bytes. The actual number of data bytes is indicated by the byte count in the beginning of the record less the number of address bytes. The first data byte is stored in the location indicated by the address in the address field. After that the address is incremented by 1 and the next data byte is stored in that new location. This continues until all bytes are stored. If there are not enough data bytes to obtain a multiple of 4 we use 0x00 as padding bytes at the end of the record. These padding bytes are ignored on the receiving side.

End of File

End of file is recognized if the first four bytes of the record all contain 0x00. In base 85 this will be "$%%%%%". This is the only decent way to terminate the file.

Size Multiplier

In general, binary data will expand in sized by approximately 1.7 times when represented with this format.

Example

Now it's time for an example. In the first table you can see the byte representation of the file to be transferred. The 4th row of bytes is not a multiple of 4 bytes. But that does not matter, for we append $00 bytes at the end until we do have a multiple of 4 bytes. These padding bytes are not counted in the byte count however!

```
D81400000000B000576F7721204469642079 6F7520726561
431400000000B0106C6C7920676F207468726F7567682061
361400000000B0206C6C20746861742074726F75626C6520
591100000000B030746F20726561642074686973F000000
00000000
```

Only after converting the bytes to base 85 we get the records of the FPC type file format presented in the next table. Note that there is always a multiple of 5 characters to represent a multiple of 4 bytes in each record.

```
$kL&@h%%,:,B.\?00EPuX0K3rO0JI))
$;UPR'%%,:<Hn&FCG:at<GVF(;G9wIw
$7FD1p%%,:LHmy:>GTV%/KJ7@GE[kYz
$B[6\;%%,:\KIn?GFWY/qKI1G5:;-_e
$%%%%%
```

As you can see the length of the lines is clearly shorter than the original ASCII lines.

SEE ALSO

http://sbprojects.fol.nl/knowledge/fileformats/fpc.htm

AUTHOR

This man page was taken from the above Web page. It was written by San Bergmans <sanmail@bigfoot.com>

For extra points: Who invented this format? Where is it used?

NAME

srec_idt − IDT/sim binary file format

DESCRIPTION

This format comes from Integrated Device Technology (IDT) System Integration Manager (IDT/sim).

It is almost identical to the Motorola S-Record format, except that most of each record is in binary, and there is no line termination character. The 'S' and tag characters are the same (ascii), however all other bytes are emitted as binary, rather than as a 2-byte hexadecimal ascii encoding.

SEE ALSO

srec_motorola(5)
The orginal Motorola S-Record format.

srec_wilson(5)
For a different spin on making S-Record into a more densely packed binary file.

COPYRIGHT

srec_cat version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_intel16 − Intel Hexadecimal 16-bit file format specification

DESCRIPTION

This format is also known as the *INHX16* format.

This document describes the hexadecimal object file format for 16-bit microprocessors.

This format is very similar to the *srec_intel*(5) format, except that the addresses are word addresses. The count field is a word count.

The hexadecimal representation of binary is coded in ASCII alphanumeric characters. For example, the 8-bit binary value 0011-1111 is 3F in hexadecimal. To code this in ASCII, one 8-bit byte containing the ASCII code for the character ’3’ (0011-0011 or 0x33) and one 8-bit byte containing the) ASCII code for the character ’F’ (0100-0110 or 0x46) are required. For each byte value, the high-order hexadecimal digit is always the first digit of the pair of hexadecimal digits. This representation (ASCII hexadecimal) requires twice as many bytes as the binary representation.

A hexadecimal object file is blocked into records, each of which contains the record type, length, memory load address and checksum in addition to the data. There are currently six (6) different types of records that are defined, not all combinations of these records are meaningful, however. The record are:

- Data Record
- End of File Record
- Extended Segment Address Record
- Start Segment Address Record
- Extended Linear Address Record
- Start Linear Address Record

General Record Format

Record Mark	Record Length	Load Offset	Record Type	Data	Check sum

Record Mark.

Each record begins with a Record Mark field containing 0x3A, the ASCII code for the colon (“:”) character.

Record Length

Each record has a Record Length field which specifies the number of 16-bit words of information or data which follows the Record Type field of the record. This field is one byte, represented as two hexadecimal characters. The maximum value of the Record Length field is hexadecimal ’FF’ or 255.

Load Offset

Each record has a Load Offset field which specifies the 16-bit starting load offset of the data words, therefore this field is only used for Data Records (if the words are loaded as bytes, the address needs to be doubled). In other records where this field is not used, it should be coded as four ASCII zero characters (“0000” or 0x30303030). This field one 16-bit word, represented as four hexadecimal characters.

Record Type

Each record has a Record Type field which specifies the record type of this record. The Record Type field is used to interpret the remaining information within the record. This field is one byte, represented as two hexadecimal characters. The encoding for all the current record types are:

0 Data Record

1 End of File Record

5 Execution Start Address Record

Data Each record has a variable length Data field, it consists of zero or more 16-bit words encoded as set of 4 hexadecimal digits, most significant digit first. The interpretation of this field depends on the Record Type field.

Checksum

Each record ends with a Checksum field that contains the ASCII hexadecimal representation of the two's complement of the 8-bit bytes that result from converting each pair of ASCII hexadecimal digits to one byte of binary, from and including the Record Length field to and including the last byte of the Data field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the Record Length field to and including the Checksum field, is zero.

Data Record

(8-, 16- or 32-bit formats)

Record Mark (":")	Record Length	Load Offset	Record Type	Data	Check sum

The Data Record provides a set of hexadecimal digits that represent the ASCII code for data bytes that make up a portion of a memory image.

The contents of the individual fields within the record are:

Record Mark

This field contains 0x3A, the hexadecimal encoding of the ASCII colon (":") character.

Record Length

The field contains two ASCII hexadecimal digits that specify the number of 16-bit data words in the record. The maximum value is 255 decimal.

Load Offset

This field contains four ASCII hexadecimal digits representing the word address at which the first word of the data is to be placed. (For an exquivalent bytes address, double it.)

Record Type

This field contains 0x3030, the hexadecimal encoding of the ASCII character "00", which specifies the record type to be a Data Record.

Data This field contains sets of four ASCII hexadecimal digits, one set for each 16-bit data word, most significant digit first.

Checksum

This field contains the check sum on the Record Length, Load Offset, Record Type, and Data fields.

Execution Start Address Record

Record Mark (":")	Record Length (4)	Load Offset (0)	Record Type (5)	EIP (4 bytes)	Check sum

The Execution Start Address Record is used to specify the execution start address for the object file. This is where the loader is to jump to begin execution once the hex load is complete.

The Execution Start Address Record can appear anywhere in a hexadecimal object file. If such a record is not present in a hexadecimal object file, a loader is free to assign a default execution start address.

The contents of the individual fields within the record are:

Record mark

This field contains 0x3A, the hexadecimal encoding of the ASCII colon (":") character.

Record length
: The field contains 0x3032, the hexadecimal encoding of the ASCII characters “02”, which is the length, in bytes, of the EIP register content within this record.

Load Offset
: This field contains 0x30303030, the hexadecimal encoding of the ASCII characters “0000”, since this field is not used for this record.

Record Type
: This field contains 0x3035, the hexadecimal encoding of the ASCII character “05”, which specifies the record type to be a Start Address Record.

EIP
: This field contains eight ASCII hexadecimal digits that specify the address. The field is encoded big-endian (most significant digit first).

Checksum
: This field contains the check sum on the Record length, Load Offset, Record Type, and EIP fields.

End of File Record

This shall be the last record in the file.

Record Mark (“:”)	Record Length (0)	Load Offset (0)	Record Type (1)	Check sum (0xFF)

The End of File Record specifies the end of the hexadecimal object file.

The contents of the individual fields within the record are:

Record mark
: This field contains 0x3A, the hexadecimal encoding of the ASCII colon (“:”) character.

Record Length
: The field contains 0x3030, the hexadecimal encoding of the ASCII characters “00”. Since this record does not contain any Data bytes, the length is zero.

Load Offset
: This field contains 0x30303030, the hexadecimal encoding of the ASCII characters “0000”, since this field is not used for this record.

Record Type
: This field contains 0x3031, the hexadecimal encoding of the ASCII character “01”, which specifies the record type to be an End of File Record.

Checksum
: This field contains the check sum an the Record Length, Load Offset, and Record Type fields. Since all the fields are static, the check sum can also be calculated statically, and the value is 0x4646, the hexadecimal encoding of the ASCII characters “FF”.

Size Multiplier

In general, binary data will expand in sized by approximately 2.3 times when represented with this format.

EXAMPLE

Here is an example INHX16 file. It contains the data “Hello, World” to be loaded at address 0.

```
:0700000065486C6C2C6F5720726F646CFF0AA8
:00000001FF
```

COPYRIGHT

srec_cat version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_intel − Intel Hexadecimal object file format specification

DESCRIPTION

This format is also known as the *Intel MCS-86 Object* format.

This document describes the hexadecimal object file format for the Intel 8-bit, 16-bit, and 32-bit microprocessors. The hexadecimal format is suitable as input to PROM programmers or hardware emulators.

Hexadecimal object file format is a way of representing an absolute binary object file in ASCII. Because the file is in ASCII instead of binary, it is possible to store the file is non-binary medium such as paper-tape, punch cards, etc.; and the file can also be displayed on CRT terminals, line printers, etc.. The 8-bit hexadecimal object file format allows for the placement of code and data within the 16-bit linear address space of the Intel 8-bit processors. The 16-bit hexadecimal format allows for the 20-bit segmented address space of the Intel 16-bit processors. And the 32-bit format allows for the 32-bit linear address space of the Intel 32-bit processors.

`--address-length=2`	“i8hex”	16-bit	
`--address-length=3`	“i16hex”	20-bit	segmented
`--address-length=4`	“i32hex”	32-bit	linear

The hexadecimal representation of binary is coded in ASCII alphanumeric characters. For example, the 8-bit binary value 0011-1111 is 3F in hexadecimal. To code this in ASCII, one 8-bit byte containing the ASCII code for the character ’3’ (0011-0011 or 0x33) and one 8-bit byte containing the) ASCII code for the character ’F’ (0100-0110 or 0x46) are required. For each byte value, the high-order hexadecimal digit is always the first digit of the pair of hexadecimal digits. This representation (ASCII hexadecimal) requires twice as many bytes as the binary representation.

A hexadecimal object file is blocked into records, each of which contains the record type, length, memory load address and checksum in addition to the data. There are currently six (6) different types of records that are defined, not all combinations of these records are meaningful, however. The record are:

- Data Record (8-, 16-, or 32-bit formats)
- End of File Record (8-, 16-, or 32-bit formats)
- Extended Segment Address Record (16- or 32-bit formats)
- Start Segment Address Record (16- or 32-bit formats)
- Extended Linear Address Record (32-bit format only)
- Start Linear Address Record (32-bit format only)

General Record Format

Record Mark	Record Length	Load Offset	Record Type	Data	Check sum

Record Mark.

Each record begins with a Record Mark field containing 0x3A, the ASCII code for the colon (“:”) character.

Record Length

Each record has a Record Length field which specifies the number of bytes of information or data which follows the Record Type field of the record. This field is one byte, represented as two hexadecimal characters. The maximum value of the Record Length field is hexadecimal ’FF’ or 255.

Load Offset

Each record has a Load Offset field which specifies the 16-bit starting load offset of the data bytes, therefore this field is only used for Data Records. In other records where this field is not used, it should be coded as four ASCII zero characters (“0000” or 0x30303030). This field is two

byte, represented as four hexadecimal characters.

Record Type
: Each record has a Record Type field which specifies the record type of this record. The Record Type field is used to interpret the remaining information within the record. This field is one byte, represented as two hexadecimal characters. The encoding for all the current record types are:

  0 Data Record

  1 End of File Record

  2 Extended Segment Address Record

  3 Start Segment Address Record

  4 Extended Linear Address Record

  5 Start Linear Address Record

Data
: Each record has a variable length Data field, it consists of zero or more bytes encoded as pairs of hexadecimal digits. The interpretation of this field depends on the Record Type field.

Checksum
: Each record ends with a Checksum field that contains the ASCII hexadecimal representation of the two's complement of the 8-bit bytes that result from converting each pair of ASCII hexadecimal digits to one byte of binary, from and including the Record Length field to and including the last byte of the Data field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the Record Length field to and including the Checksum field, is zero.

Extended Linear Address Record

(32-bit format only)

Record Mark (":")	Record Length (2)	Load Offset (0)	Record Type (4)	ULBA (2 bytes)	Check sum

The 32-bit Extended Linear Address Record is used to specify bits 16-31 of the Linear Base Address (LBA), where bits 0-15 of the LBA are zero. Bits 16-31 of the LBA are referred to as the Upper Linear Base Address (ULBA). The absolute memory address of a content byte in a subsequent Data Record is) obtained by adding the LBA to an offset calculated by adding the Load Offset field of the containing Data Record to the index of the byte in the Data Record (0, 1, 2, ... *n*). This offset addition is done) modulo 4G (*i.e.* 32-bits from 0xFFFFFFFF to 0x00000000) results in wrapping around from the end to the beginning of the 4G linear address defined by the LBA. The linear address at which a particular byte is loaded is calculated as:

(LBA + DRLO + DRI) MOD 4G

where:

DRLO is the Load Offset field of a Data Record.

DRI is the data byte index within the Data Record.

When an Extended Linear Address Record defines the value of LBA, it may appear anywhere within a 32-bit hexadecimal object file. This value remains in effect until another Extended Linear Address Record is encountered. The LBA defaults to zero until an Extended Linear Address Record is encountered. The contents of the individual fields within the record are:

Record Mark
: This field contains 0x3A, the hexadecimal encoding of the ASCII colon (":") character.

Record Length
: The field contains 0x3032, the hexadecimal encoding of the ASCII characters "02", which is the length, in bytes, of the ULBA data information within this record.

Load Offset
: This field contains 0x30303030, the hexadecimal encoding of the ASCII characters “0000”, since this field is not used for this record.

Record Type
: This field contains 0x3034, the hexadecimal encoding of the ASCII character “04”, which specifies the record type to be an Extended Linear Address Record.

ULBA
: This field contains four ASCII hexadecimal digits that specify the 16-bit Upper Linear Base Address value. The value is encoded big-endian (most significant digit first).

Checksum
: This field contains the check sum on the Record Length, Load Offset, Record Type, and ULBA fields.

Extended Segment Address Record

(16- or 32-bit formats)

Record Mark (“:”)	Record Length (2)	Load Offset (0)	Record Type (2)	USBA (2 bytes)	Check sum

The 16-bit Extended Segment Address Record is used to specify bits 4-19 of the Segment Base Address (SBA), where bits 0-3 of the SBA are zero. Bits 4-19 of the SBA are referred to as the Upper Segment Base Address (USBA). The absolute memory address of a content byte in a subsequent Data Record is) obtained by adding the SBA to an offset calculated by adding the Load Offset field of the containing Data Record to the index of the byte in the Data Record (0, 1, 2, ... *n*). This offset addition is done modulo 64K (*i.e.* 16-bits from 0xFFFF to 0x0000 results in wrapping around from the end to the beginning of the 64K segment defined by the SBA. The address at which a particular byte is loaded is calculated as:

SBA + ((DRLO + DRI) MOD 64K)

where:

DRLO
: is the LOAD OFFSET field of a Data Record.

DRI
: is the data byte index within the Data Record.

When an Extended Segment Address Record defines the value of SBA, it may appear anywhere within a 16-bit hexadecimal object file. This value remains in effect until another Extended Segment Address Record is encountered. The SBA defaults to zero until an Extended Segment Address Record is encountered.

The contents of the individual fields within the record are:

Record Mark
: This field contains 0x3A, the hexadecimal encoding of the ASCII colon (“:”) character.

Record Length
: The field contains 0x3032, the hexadecimal encoding of the ASCII characters ’02’, which is the length, in bytes, of the USBA data information within this record.

Load Offset
: This field contains 0x30303030, the hexadecimal encoding of the ASCII characters ’0000’, since this field is not used for this record.

Record Type
: This field contains 0x3032, the hexadecimal encoding of the ASCII character “02”, which specifies the record type to be an Extended Segment Address Record.

USBA
: This field contains four ASCII hexadecimal digits that specify the 16-bit Upper Segment Base Address value. The field is encoded big-endian (most significant digit first).

Checksum
: This field contains the check sum on the Record length, Load Offset, Record Type, and USBA fields.

Data Record

(8-, 16- or 32-bit formats)

Record Mark (":")	Record Length	Load Offset	Record Type	Data	Check sum

The Data Record provides a set of hexadecimal digits that represent the ASCII code for data bytes that make up a portion of a memory image. The method for calculating the absolute address (linear in the 8-bit and 32-bit case and segmented in the 16-bit case) for each byte of data is described in the discussions of the Extended Linear Address Record and the Extended Segment Address Record.

The contents of the individual fields within the record are:

Record Mark
: This field contains 0x3A, the hexadecimal encoding of the ASCII colon (":") character.

Record Length
: The field contains two ASCII hexadecimal digits that specify the number of data bytes in the record. The maximum value is 255 decimal.

Load Offset
: This field contains four ASCII hexadecimal digits representing the offset from the LBA (see Extended Linear Address Record see Extended Segment Address Record) defining the address which the first byte of the data is to be placed.

Record Type
: This field contains 0x3030, the hexadecimal encoding of the ASCII character "00", which specifies the record type to be a Data Record.

Data
: This field contains pairs of ASCII hexadecimal digits, one pair for each data byte.

Checksum
: This field contains the check sum on the Record Length, Load Offset, Record Type, and Data fields.

Note: Care must be taken when the addresses with an record span the end of addressing. The behaviour is different for linear and segmented addressing modes.

linear
: If a record starts just short of 2**32, and would finish after 2**32, the later part of the record wraps around to address 0. TP 8n segment If a record starts just for of a 2**16 boundary, and would finish after that 2**16 boundary, the later part of the record wraps around to address 0 within the same segment (**not** the next segment).

The *srec_cat*(1) program will never output records such as these, it will always produce separate records on output.

Start Linear Address Record

(32-bit format only)

Record Mark (":")	Record Length (4)	Load. Offset (0)	Record Type (5)	EIP (4 bytes)	Check sum

The Start Linear Address Record is used to specify the execution start address for the object file. The value given is the 32-bit linear address for the EIP register. Note that this record only specifies the code address within the 32-bit linear address space of the 80386. If the code is to start execution in the real mode of the 80386, then the Start Segment Address Record should be used instead, since that record specifies both the CS and IP register contents necessary for real mode.

The Start Linear Address Record can appear anywhere in a 32-bit hexadecimal object file. If such a record is not present in a hexadecimal object file, a loader is free to assign a default execution start address.

The contents of the individual fields within the record are:

Record mark
: This field contains 0x3A, the hexadecimal encoding of the ASCII colon (“:”) character.

Record length
: The field contains 0x3034, the hexadecimal encoding of the ASCII characters “04”, which is the length, in bytes, of the EIP register content within this record.

Load Offset
: This field contains 0x30303030, the hexadecimal encoding of the ASCII characters “0000”, since this field is not used for this record.

Record Type
: This field contains 0x3035, the hexadecimal encoding of the ASCII character “05”, which specifies the record type to be a Start Linear Address Record.

EIP
: This field contains eight ASCII hexadecimal digits that specify the 32-bit EIP register contents. The field is encoded big-endian (most significant digit first).

Checksum
: This field contains the check sum on the Record length, Load Offset, Record Type, and EIP fields.

Start Segment Address Record

(16- or 32-bit formats)

Record Mark (“:”)	Record Length (4)	Load. Offset (0)	Record Type (3)	CS (2 bytes)	IP (2 bytes)	Check sum

The Start Segment Address Record is used to specify the execution start address for the object file. The value given is the 20-bit segment address for the CS and IP registers. Note that this record only specifies the code address within the 20-bit segmented address space of the 8086/80186. The Start Segment Address Record can appear anywhere in a 16-bit hexadecimal object file. If such a record is not present in a hexadecimal object file, a loader is free to assign a default start address.

The contents of the individual fields within the record are:

Record Mark
: This field contains 0x3A, the hexadecimal encoding of the ASCII colon (“:”) character.

Record Length
: The field contains 0x3034, the hexadecimal encoding of the ASCII characters “04”, which is the length, in bytes, of the CS and IP register contents within this record.

Load Offset
: This field contains 0x30303030, the hexadecimal encoding of the ASCII characters “0000”, since this field is not used for this record.

Record Type
: This field contains 0x3033, the hexadecimal encoding of the ASCII character ’03’, which specifies the record type to be a Start Segment Address Record.

CS
: This field contains four ASCII hexadecimal digits that specify the 16-bit CS register contents. The field is encoded big-endian (most significant digit first).

IP
: This field contains four ASCII hexadecimal digits that specify the 16-bit IP register contents. The field is encoded big-endian (most significant digit first).

Checksum
: This field contains the check sum on the Record length, Load Offset, Record Type, CS, and IP fields.

End of File Record

(8-, 16-, or 32-bit formats)

Record Mark (":")	Record Length (0)	Load Offset (0)	Record Type (1)	Check sum (0xFF)

The End of File Record specifies the end of the hexadecimal object file.

The contents of the individual fields within the record are:

Record mark
: This field contains 0x3A, the hexadecimal encoding of the ASCII colon (":") character.

Record Length
: The field contains 0x3030, the hexadecimal encoding of the ASCII characters "00". Since this record does not contain any Data bytes, the length is zero.

Load Offset
: This field contains 0x30303030, the hexadecimal encoding of the ASCII characters "0000", since this field is not used for this record. In ancient times, i8hex used this for the start address record.

Record Type
: This field contains 0x3031, the hexadecimal encoding of the ASCII character "01", which specifies the record type to be an End of File Record.

Checksum
: This field contains the check sum an the Record Length, Load Offset, and Record Type fields. Since all the fields are static, the check sum can also be calculated statically, and the value is 0x4646, the hexadecimal encoding of the ASCII characters "FF".

Size Multiplier

In general, binary data will expand in sized by approximately 2.3 times when represented with this format.

EXAMPLE

Here is an example Intel hex file. It contains the data "Hello, World" to be loaded at address 0.

```
:0D00000048656C6C6F2C20576F726C640AA1
:00000001FF
```

REFERENCE

This information comes (very indirectly) from *Microprocessors and Programmed Logic*, Second Edition, Kenneth L. Short, 1987, Prentice-Hall, ISBN 0-13-580606-2.

http://en.wikipedia.org/wiki/Intel_HEX

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

Derivation

This manual page is derived from a file marked as follows:

Intel Hexadecimal Object File Format Specification; Revision A, 1/6/88

Disclaimer: Intel makes no representation or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Intel reserves the right to revise this publication from time to time in the content hereof without obligation of Intel to notify any person of such revision or changes. The publication of this specification should not be construed as a commitment on Intel's part to implement any product.

NAME

srec_logisim − format Logisim EPROM load files

DESCRIPTION

The file format used for image files is intentionally simple; this permits you to write a program, such as an assembler, that generates memory images that can then be loaded into memory. As an example of this file format, if we had a 256-byte memory whose first five bytes were 2, 3, 0, 20, and −1, and all subsequent values were 0, then the image would be the following text file.

```
v2.0 raw

02 03 00 14 ff
```

The first line identifies the file format used (currently, there is only one file format recognized). Subsequent values list the values in hexadecimal, starting from address 0; you can place several such values on the same line. Logisim will assume that any values unlisted in the file are zero.

The image file can use run-length encoding; for example, rather than list the value 00 sixteen times in a row, the file can include 16*00 rather than repeat 00 sixteen times. Notice than the number of repetitions is written in base 10. Files produced by Logisim will use run-length encoding for runs of at least four values

Size Multiplier

In general, binary data will expand in sized by approximately 2.95 times when represented with this format.

Commentary

This format is particularly deficient.

- The is no way for changing address in-flight. And therefore the format can’t have holes in the data.
- There are no checksums, or any other error checking.
- There is no provision for a comment header, as distinct to the “v2.0 raw” magic number.
- There is no provision for an execution start adddress.

SEE ALSO

http://ozark.hendrix.edu/~burch/logisim/docs/2.3.0/guide/mem/menu.html

COPYRIGHT

srec_logisim.5 version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_logisim.5* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_logisim.5 −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_logisim.5 −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_mem − Lattice Memory Initialization format

DESCRIPTION

A Lattice Memory Initialization format (.mem), by Lattice Semiconductor, file is an ASCII text file that consists of a header followed by lines of memory data.

Syntax

The data must be in one of the following formats: Bin (binary), Hex (hexadecimal), or Address-Hex (described below).

For hexadecimal values, both upper and lower case can be used. If the data has fewer bits than the specified data width, the most significant bits are filled with 0. Any address not specified will be filled with 0.

Comments can be added at any point after the header (defined below) by starting the comment with a pound sign (#) or two slashes (//). The comment then includes everything to the end of the line. Comments may be added to any of the data, but never add comments to the header.

Header

A `.mem` file starts with a header, which declares the file format, memory size, and address and data display radix for Memory Generator. The syntax of the header is:

```
#Format=Bin | Hex | AddrHex
#Depth=1 to 65536
#Width=1 to 256
#AddrRadix=index-number
#DataRadix=index-number
#Data
```

The *index-number* can be one of the following numbers. AddrRadix and DataRadix can have different values.

Binary: 0
Octal: 1
Decimal: 2
Hexadecimal: 3

For example, the following header says the .mem file is using the binary format for a 32x8 memory. When displayed in Memory Generator, the address will be shown in hexadecimal and the data will be shown in binary.

```
#Format=Bin
#Depth=32
#Width=8
#AddrRadix=3
#DataRadix=0
#Data
```

Bin and Hex Formats

The data is represented in binary or hexadecimal format. Each line of data specifies the contents for one memory location, starting with address 0. That is, the first line is for address 0, the second line is for address 1, and so on. For each line, the data is interpreted as least significant bit on the right.

For example, in the Bin format, the following lines will initialize address 0 to “00011011”, address 1 to “11111010” (assuming it is a 32x8 memory).

```
# for a 32x8 memory
11011
11111010
```

In the Hex format, the following lines will initialize address 0 to “003B”, address 1 to “FB0A” (assuming it is a 32x16 memory).

```
# for a 32x16 memory
3B
```

FB0A

AddrHex

The data is represented in hexadecimal format. Each line consists of an address followed by a colon and then any number of data words, separated by spaces:

*address***:** *data data> data...*

The data will be applied starting at <address> and filling in sequentially from there.

For example:

A0:03 F3 3E 4F
B2:3B 9F

will initialize A0 with 03, A1 with F3, A2 with 3E, A3 with 4F, B2 with 3B, and B3 with 9F. The other addresses will be initialized to 0. So A4 through B1 will be set to 0.

See Also

http://help.latticesemi.com/docs/webhelp/eng/wwhelp/wwhimpl/common/html/wwhelp.htm#href=Design%20Entry/mem

Size Multiplier

The size multiplier depends on the width selected. As files grow larger, their size multipliers will approach those in the table, from above.

Width	Linux	Windows
8	2.96	3.0
16	2.47	2.5
32	2.25	2.28
64	2.13	2.15

Byte Order

This format is implicitly big-endian. Use a −byte-swap filter if you need something different.

COPYRIGHT

srec_mem version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_mem* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_mem −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_mem −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_mif − Memory Initialization File (MIF) format

DESCRIPTION

This format was invented by Altera.

An ASCII text file (with the extension .mif) that specifies the initial content of a memory block (CAM, RAM, or ROM), that is, the initial values for each address. This file is used during project compilation and/or simulation. You can create a Memory Initialization File in the Memory Editor, the In-System Memory Content Editor, or the Quartus II Text Editor.

A Memory Initialization File serves as an input file for memory initialization in the Compiler and Simulator. You can also use a Hexadecimal (Intel-Format) File (.hex) to provide memory initialization data.

A Memory Initialization File contains the initial values for each address in the memory. A separate file is required for each memory block. In a Memory Initialization File, you must specify the memory depth and width values. In addition, you can specify data radixes as binary (BIN), hexadecimal (HEX), octal (OCT), signed decimal (DEC), or unsigned decimal (UNS) to display and interpret addresses and data values. Data values must match the specified data radix.

When creating a Memory Initialization File in the Quartus II Text Editor, you must start with the DEPTH, WIDTH, ADDRESS_RADIX and DATA_RADIX keywords. You can use Tab "" and Space " " characters as separators, and insert multiple lines of comments with the percent "%" character, or a single comment with double dash "−−" characters. Address:data pairs represent data contained inside certain memory addresses and you must place them between the CONTENT BEGIN and END keywords, as shown in the following examples.

```
%  multiple-line comment
multiple-line comment  %
-- single-line comment
DEPTH = 8;                    -- The size of data in bits
WIDTH = 32;                   -- The size of memory in words
ADDRESS_RADIX = HEX;          -- The radix for address values
DATA_RADIX = BIN;             -- The radix for data values
CONTENT                       -- start of (address : data pairs)
BEGIN
00 : 00000000;                -- memory address : data
01 : 00000001;
02 : 00000010;
03 : 00000011;
04 : 00000100;
05 : 00000101;
06 : 00000110;
07 : 00000111;
08 : 00001000;
09 : 00001001;
0A : 00001010;
0B : 00001011;
0C : 00001100;
END;
```

There are several ways to specify the address and data, as seen in the following table:

Notation	Interpretation	Example
A : D;	Addr[A] = D	2 : 4 Address: 01234567 Data: 00400000

[A0..A1] : D;
(See note below.)
	Addr[A0] to [A1] contain data D	`[0..7] : 6`
		`Address: 01234567`
		`Data: 66666666`

[A0..A1] : D0 D1;
(See note below.)
	Addr[A0] = D0,
	Addr[A0+1] = D1,
	Addr[A0+2] = D0,
	Addr[A0+3] = D1,
	until A0+n = A1
		`[0..7] : 5 6`
		`Address: 01234567`
		`Data: 56565656`

A : D0 D1 D2;	Addr[A] = D0,
	Addr[A+1] = D1,
	Addr[A+2] = D2
		`2 : 4 5 6`
		`Address: 01234567`
		`Data: 00456000`

Note: The address range forms are limited in SRecord, the range must be less than 255 bytes. SRecord will never write an address range.

Note: When reading MIF file, SRecord will round up the number of bits in the WIDTH to be a multiple of 8. Multi-byte values will be laid down in memory as big-endian.

An ASCII text file (with the extension .mif) that specifies the initial content of a memory block (CAM, RAM, or ROM), that is, the initial values for each address. This file is used during project compilation and/or simulation. A MIF contains the initial values for each address in the memory. In a MIF, you are also required to specify the memory depth and width values. In addition, you can specify the radixes used to display and interpret addresses and data values.

SIZE MULTIPLIER

In general, binary data will expand in sized by approximately 3.29 times when 8-bit data is represented with this format (16 bit = 2.75, 32 bit = 2.47, 64 bit = 2.34).

EXAMPLE

Following is a sample MIF:

```
DEPTH = 32; % Memory depth and width are required %
% DEPTH is the number of addresses %
WIDTH = 14; % WIDTH is the number of bits of data per word %
% DEPTH and WIDTH should be entered as decimal numbers %
ADDRESS_RADIX = HEX; % Address and value radixes are required %
DATA_RADIX = HEX; % Enter BIN, DEC, HEX, OCT, or UNS; unless %
                  % otherwise specified, radixes = HEX %
 --Specify values for addresses, which can be single address or range
CONTENT
BEGIN
[0..F]: 3FFF;     % Range: Every address from 0 to F = 3FFF %
6     :    F;     % Single address: Address 6 = F %
8     :    F E 5; % Range starting from specific address %
--                % Addr[8] = F, Addr[9] = E, Addr[A] = 5 %
END;
```

REFERENCE

The above information was gleaned from the following sources:
http://www.altera.com/support/software/nativelink/quartus2/glossary/def_mif.html
http://www.mil.ufl.edu/4712/docs/mif_help.pdf

COPYRIGHT

srec_mif version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_mif* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_mif −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_mif −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_mips_flash − MIPS-Flash file format

DESCRIPTION

The MIPS SDE tool chain has a *convert* program that is able to output this format. I have no idea what reads it, some kind of flash programmer I suppose.

Format

The file must start with '!R' to reset the state machine. White space appears to be ignored, except as it serves to separate tokens.

Data is presented as 32-bit hexadecimal numbers, in the normal big-endian text number format. To write them to memory, you have to know if the target is big-endian or little-endian. for little endian targets, reverse the order of the bytes in this number.

The eight bytes following a '>' are a sort of comment. The SDE code reads like they are displayed in the flash programmer as a kind of progress indicator.

The number following '@' is a new address for the following data.

Each segment must be erased before it can be written, this is done with the '!E' command. Each segment is assumed to be 128kB in size.

Special segments must unlocked (with the '!C' command) and locked again (with the '!S' command).

This file format contains no checksum mechanism.

Command Line Options

This format is specified using one of the **−Mips-Flash-Big_Endian** or **−Mips-Flash-Little_Endian** options. The endian-ness must be specified on the command line, because there is nothing in the file contents to indicate the endian-ness of the data it contains.

Size Multiplier

In general, binary data will expand in sized by approximately 2.3 times when represented with this format (worse if you use shorter lines).

EXAMPLE

Here is an example MIPS-Flash file. It contains the data “Hello, World” to be loaded at bytes address 0x0000 (but remember, the file contents are always multiples of four bytes).

```
!R
>00000xxx @00000000 !E
@00000000
>00000000
48656C6C 6F2C2057 6F726C64 210A0000
>#DL_DONE
>FINISHED
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

COPYRIGHT

srec_cat version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_mos_tech − MOS Technology file format

DESCRIPTION

The MOS Technology format allows binary files to be uploaded and downloaded between between a computer system (such as a PC, Macintosh, or workstation) and an emulator or evaluation board for microcontrollers and microprocessors.

The Lines

Each line consists of 5 fields. These are the length field, address field, data field, and the checksum. The lines always start with a semicolon (;) character.

The Fields

;	Length	Address	Data	Checksum	CRLF

Length The record length field is a 2 character (1 byte) field that specifies the number of data bytes in the record. Typically this is 24 or less.

Address This is a 2-byte address that specifies where the data in the record is to be loaded into memory, big-endian.

Data The data field contains the executable code, memory-loadable data or descriptive information to be transferred.

Checksum

The checksum is an 2-byte field that represents the least significant two bytes of the the sum of the values represented by the pairs of characters making up the record's length, address, and data fields, big-endian.

End of File

The final line should have a data length of zero, and the data line count in the address field. The checksum is not the usual checksum, it is instead a repeat of the data line count.

Size Multiplier

In general, binary data will expand in sized by approximately 2.54 times when represented with this format.

EXAMPLE

Here is an example MOS Technology format file. It contains the data "Hello, World" to be loaded at address 0.

```
;0C000048656C6C6F2C20576F726C640454
;0000010001
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

KIM-1 User Manual − Appendix F − Paper Tape Format

(The following information is reproduced from http://users.telenet.be/kim1-6502/6502/usrman.html#F just in case it vanishes from the Web.)

The paper tape LOAD and DUMP routines store and retrieve data in a specific format designed to insure error free recovery. Each byte of data to be stored is converted to two half bytes. The half bytes (whose possible values are 0 to F HEX) are translated into their ASCII equivalents and written out onto paper tape

in this form.

Each record outputted begins with a “;” character (ASCII 3B) to mark the start of a valid record. The next byte transmitted (18HEX) or (24 decimal) is the number of data bytes contained in the record. The record’s starting address High (1 byte, 2 characters), starting address Lo (1 byte, 2 characters), and data (24 bytes, 48 characters) follow. Each record is terminated by the record’s check-sum (2 bytes, 4 characters), a carriage return (ASCII 0D), line feed (ASCII 0A), and six “NULL” characters (ASCII 00). (NULL characters cause a blank area on the paper tape.)

The last record transmitted has zero data bytes (indicated by ;00) The starting address field is replaced by a four digit Hex number representing the total number of data records contained in the transmission, followed by the records usual check-sum digits. An “XOFF” character ends the transmission.

```
;180000FFEEDDCCBBAA00998877665544332211223344556677889 90AFC
;0000010001
```

During a “LOAD” all incoming data is ignored until a “;” character is received. The receipt of non ASCII data or a mismatch between a records calculated check-sum and the check-sum read from tape will cause an error condition to be recognized by KIM. The check-sum is calculated by adding all data in the record except the “;” character.

The paper tape format described is compatible with all other MOS Technology, Inc. software support programs.

NAME

srec_motorola − Motorola S-Record hexadecimal file format

DESCRIPTION

This format is also known as the *Exorciser*, *Exormacs* or *Exormax* format.

Motorola's S-record format allows binary files to be uploaded and downloaded between two computer systems. This type of format is widely used when transferring programs and data between a computer system (such as a PC, Macintosh, or workstation) and an emulator or evaluation board for Motorola microcontrollers and microprocessors.

The Lines

Most S-Record file contain only S-Record lines (see the next section), which always start with a capital S character. Some systems generate various "extensions[rq] which usually manifest as lines which start with something else. These "extension[rq] lines may or may not break other systems made by other vendors. Caveat emptor.

The Fields

The S-Record format consists of 5 fields. These are the type field, length field, address field, data field, and the checksum. The lines always start with a capital S character.

S	Type	Record Length	Address	Data	Checksum

Type The type field is a 1 character field that specifies whether the record is an S0, S1, S2, S3, S5, S6, S7, S8 or S9 field.

Record Length
: The record length field is a 2 character (1 byte) field that specifies the number of character pairs (bytes) in the record, excluding the type and record length fields.

Address This is a 2-, 3- or 4-byte address that specifies where the data in the S-Record is to be loaded into memory.

Data The data field contains the executable code, memory-loadable data or descriptive information to be transferred.

Checksum
: The checksum is an 8-bit field that represents the least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record's length, address, and data fields.

Record Types

S0 This type of record is the header record for each block of S-Records. The data field may contain any descriptive information identifying the following block of S-Records. (It is commonly "HDR[rq] on many systems.) The address field is normally zero.

S1 A record containing data and the 2-byte address at which the data is to reside.

S2 A record containing data and the 3-byte address at which the data is to reside.

S3 A record containing data and the 4-byte address at which the data is to reside.

S5
: An optional record containing the number of S1, S2 and S3 records transmitted in a particular block. The count appears in the two-byte address field. There is no data field.

  This record is optional, you do not have to use it. Nobody knows if you can have more than one in a file; and if you do, nobody knows whether or not the line count resets after each one.

  The *srec_cat* command will only ever use one, provided the number of lines fits in 16 bits, otherwise it will use S6.

S6
: An optional record containing the number of S1, S2 and S3 records

transmitted in a particular block. The count appears in the three-byte address field. There is no data field.

This record is optional, you do not have to use it. Nobody knows if you can have more than one in a file; and if you do, nobody knows whether or not the line count resets after each one.

Nobody knows what happens if you mix S5 and S6 records in a file.

The *srec_cat* command will only ever use one, provided the number of lines fits in 24 bits.

S7 A termination record for a block of S3 records. The address field may contain the 4-byte address of the instruction to which control is passed. There is no data field.

S8 A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is passed. There is no data field.

S9 A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is passed. If not specified, the first entry point specification encountered in the object module input will be used. There is no data field.

Size Multiplier

In general, binary data will expand in sized by approximately 2.4 times when represented with this format.

EXAMPLE

Here is an example S-Record file. It contains the data “Hello, World[rq] to be loaded at address 0.

```
S00600004844521B
S110000048656C6C6F2C20576F726C640A9D
S5030001FB
S9030000FC
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_msbin − Windows CE Binary Image Data Format

DESCRIPTION

This format is the output of the Microsoft WinCE Platform Builder. This is a binary (non-text) file format. File names in this format typically (ambiguously) use the `.bin` suffix.

File Format

Files in this format start with a header record.

Then comes the data itself, organized into records.

The file finishes with an execution start address record. This is mandatory.

File Header Record

Data in this format start with an optional header containing the magic “B000FF\n”, followed by the image start (four bytes, little endian) address and the span of the image (highest address − lowest address + 1) (four bytes, little endian). The file header does not have a checksum; it is therefore possible that a corrupt file header will go undetected.

Magic “B000FF\n” (7 bytes)	Least Address (4 bytes)	Greatest Address (4 bytes)

There is no provision for a file comment of any kind.

Data Record

Each record consists of a record start address (four bytes, little endian), a record length (four bytes, little endian), a record checksum (four bytes, little endian), followed by the record data. The data part of each record is raw byte values, no encoding.

Start address (4 bytes)	Length (4 bytes)	Checksum (4 bytes)	Data

The checksum is calculated by a simple sum of unsigned bytes into a 32-bit accumulator.

The 12 record header bytes are not included in the record checksums; it is therefore possible that a corrupt record header will go undetected.

It is not possible to place data at address zero with this format. Address zero is reserved for use by the execution start address record.

There is effectively no limit on the length of a record (2^32−1). It is not uncommon for a MsBin file to contain records with sizes in the tens of megabytes.

Execution Start Address Record

Last comes a special record with the record address set to zero and record length set to the image execution start address. According to specification the record describing the execution start address must be always present, and must always be the last record in the file.

Zero (4 bytes)	Start Address (4 bytes)	Checksum = 0 (4 bytes)

Commentary

The MsBin files produced by SRecord are valid and can be successfully parsed by the command line utilities `viewbin` and `cvrtbin` (part of Windows CE platform).

For a MsBin file to be usable in Microsoft WinCE Platform Builder it has to contain a TOC meta-structure. This is data embedded in the file by Microsoft WinCE Platform Builder itself.

The opposite conversion − from MsBin − comes in handy when analyzing a MsBin file (*i.e.* a WinCE image).

Size Multiplier

In general, binary data will expand in sized by approximately 1.0 times (approaching asymptotically from above) when represented with this format, as the 15-byte file header is averaged over the data content. Holes in the data will also increase the size.

SOURCE

http://msdn.microsoft.com/en−us/library/ms924510.aspx

COPYRIGHT

srec_cat version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_needham − Needham EMP-series programmer ASCII file format

DESCRIPTION

This format is understood by Needham Electronics' EMP-series programmers. See `www.needhams.com/winman.pdf` for more information. (This format is very similar to the ASCII-Hex format, but without the ^B and ^C guard characters.)

Each data byte is represented as 2 hexadecimal characters, and is separated by white space from all other data bytes.

The address for data bytes is set by using a sequence of `$A`*`nnnn`*`,` characters, where *nnnn* is the 8-character ascii representation of the address. The comma is required. There is no need for an address record unless there are gaps. Implicitly, the file starts a address 0 if no address is set before the first data byte.

Size Multiplier

In general, binary data will expand in sized by approximately 3.0 times when represented with this format.

EXAMPLE

Here is an example ascii-hex file. It contains the data "Hello, World[rq] to be loaded at address 0x1000.

```
$A1000,
48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 0A
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_os65v − OS65V Loader file format

DESCRIPTION

This format is used by Ohio Scientific OS65V-compatible loaders. This family of machines includes the OSI C1P, Superboard II, C2, C4, C8, and Challenger III, as well as the UK101, and Elektor Junior.

The file startes with a period '.' (0x2E), to ensure address entry mode. then a 4-digit hex address, followed by a slash '/' (0X2F) to enter the data entry mode. The initial address is always present. There is no need for an additional address record unless there are gaps.

Each data byte is represented as 2 hexadecimal characters, and is separated by a carriage return character (0x0D) (advance address). The final return character may be omitted.

The data is concluded with a period '.' (0x2E) to re-enter address mode. If an address to start execution is specified, then the last 5 bytes are *nnnn*G where *nnnn* is the 4-digit execution address, and G is the 'Go' command.

Size Multiplier

In general, binary data will expand in sized by approximately 3.0 times when represented with this format.

EXAMPLE

Here is an example ascii-hex file. It contains the data “Hello, World[rq] to be loaded at address 0x1000, with execution at 0x1003. (On a 6502, this is the opcode for indirect jump to 0x2C6F.)

1000/48^M65^M6C^M6C^M6F^M2C^M20^M57^M6F^M72^M6C^M64^M0A^M.1010G

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_ppb − Stag Prom Programmer binary format

DESCRIPTION

This is the native binary format of the Stag Prom Programmer.

Format

The format is packet based. The packet is somposed of an 0x01 byte, the packet payload size (4 bytes, big-endian), the packet address size (4 bytes, big-endian), the packet data, and a one-byte simple sum of the payload data.

If the packet payload is more than 1024 bytes, there is an intermediate checksum after each 1024th payload pyte.

The end of file is indicated by a packet with a zero-length payload.

Command Line Option

This format is specified using the **−PPB** command line option.

Size Multiplier

In general, binary data will expand in sized by approximately 1.002 times when represented with this format (worse if there are many short data regions).

SEE ALSO

http://www.stag.co.uk/

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_ppx − Stag Prom Programmer hexadecimal format

DESCRIPTION

This is the native hexadecimal format of the Stag Prom Programmer.

Format

The file must start with an asterisk '*' on a line by itself.

Each line has a 16-bit address, followed by 8-bit bytes.

The end is indicated by '$S' folloowed by a 16-bit checksum of the data bytes as 4 hex digits.

Command Line Option

This format is specified using the **−PPX** command line option.

Size Multiplier

In general, binary data will expand in sized by approximately 3 times when represented with this format (worse if you use shorter lines).

EXAMPLE

Here is an example PPX file. It contains the data “Hello, World” to be loaded at bytes address 0x0000 (but remember, the file contents are always multiples of four bytes).

```
*
0000 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21 0A $S0473
```

SEE ALSO

http://matthieu.benoit.free.fr/pdf/pp39.pdf http://www.stag.co.uk/

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_signetics − Signetics file format

DESCRIPTION

The Signetics file format is not often used. The major disadvantage in modern applications is that the addressing range is limited to only 64kb.

Records

All data lines are called records, and each record contains the following 5 fields:

:	aaaa	cc	as	dd	ss

The field are defined as follows:

: Every record starts with this identifier.

aaaa The address field. A four digit (2 byte) number representing the first address to be used by this record.

cc The byte-count. A two digit value (1 byte), counting the actual data bytes in the record.

as Address checksum. Covers 2 address bytes and the byte count.

dd The actual data of this record. There can be 1 to 255 data bytes per record (see cc)

ss Data Checksum. Covers only all the data bytes of this record.

Record Begin

Every record begins with a colon ":[rq] character. Records contain only ASCII characters. No spaces or tabs are allowed in a record. In fact, apart from the 1st colon, no other characters than 0..9 and A..F are allowed in a record. Interpretation of a record should be case less, it does not matter if you use a..f or A..F.

Unfortunately the colon was chosen for the Signetics file format, similar to the Intel format (see *srec_intel*(5) for more information). However, SRecord is able to automatically detect the dofference between the two format, when you use the **−Guess** format specifier.

Address Field

This is the address where the first data byte of the record should be stored. After storing that data byte, the address is incremented by 1 to point to the address for the next data byte of the record. And so on, until all data bytes are stored. The address is represented by a 4 digit hex number (2 bytes), with the MSD first. The order of addresses in the records of a file is not important. The file may also contain address gaps, to skip a portion of unused memory.

Byte Count

The byte count cc counts the actual data bytes in the current record. Usually records have 32 data bytes, but any number between 1 and 255 is possible.

A value of 0x00 for cc indicates the end of the file. In this case not even the address checksum will follow! The record (and file) are terminated immediately.

It is not recommended to send too many data bytes in a record for that may increase the transmission time in case of errors. Also avoid sending only a few data bytes per record, because the address overhead will be too heavy in comparison to the payload.

Address Checksum

This is not really a checksum anymore, it looks more like a CRC. The checksum can not only detect errors in the values of the bytes, but also bytes out of order can be detected.

The checksum is calculated by this algorithm:

```
checksum = 0
for i = 1 to 3
  checksum = checkum XOR byte
  ROL checksum
next i
```

For the Address Checksum we only need 2 Address bytes and 1 Byte Count byte to be added. That's why

we count to 3 in the loop. Every byte is XORed with the previous result. Then the intermediate result is rolled left (carry rolls back into b0).

This results in a very reliable checksum, and that for only 3 bytes!

The last record of the file does not contain any checksums! So the file ends right after the Byte Count of 0.

Data Field

The payload of the record is formed by the Data field. The number of data bytes expected is given by the Byte Count field. The last record of the file may not contain a Data field.

Data Checksum

This checksum uses the same algorithm as used for the Address Checksum. This time we calculate the checksum with only the data bytes of this record.

```
checksum = 0
for i = 1 to cc
  checksum = checksum XOR byte
  ROL checksum
next i
```

Note that we count to the Byte Count cc this time.

Size Multiplier

In general, binary data will expand in sized by approximately 2.4 times when represented with this format.

EXAMPLE

Here is an example Signetics file

```
:B00010A5576F77212044696420796F75207265617B
:B01010E56C6C7920676F207468726F756768206136
:B02010256C6C20746861742074726F75626C652068
:B0300D5F746F20726561642074686973FD1
:B03D00
```

In the example above you can see a piece of code in Signetics format. The first 3 lines have 16 bytes of data each, which can be seen by the byte count. The 4th line has only 13 bytes, because the program is at it’s end there.

Notice that the last record of the file contains no data bytes, and not even an Address Checksum.

SEE ALSO

http://sbprojects.fol.nl/knowledge/fileformats/signetics.htm

AUTHOR

This man page was taken from the above Web page. It was written by San Bergmans <sanmail@bigfoot.com>

NAME

srec_spasm − SPASM file format

DESCRIPTION

This format is the output of the Paralax SPASM assembler (now defunct, I'm told). The file contains two columns of 16-bit hexadecimal coded values. The first column is the word address, the second column is the word data.

By default, SRecord treats this is big-endian data (the most significant byte first). If you want little endian order, use the −spasm-le argument instead.

Size Multiplier

In general, binary data will expand in sized by approximately 5.0 times when represented with this format (5.5 times in Windows).

EXAMPLE

Here is an example SPASM file. It contains the data "Hello, World[rq] to be loaded at bytes address 0x0100 (but remember, the file contents are word addressed).

```
0080 6548
0081 6C6C
0082 2C6F
0083 5720
0084 726F
0085 646C
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_spectrum − Spectrum file format

DESCRIPTION

In this format, bytes are recorded as ASCII code with binary digits represented by 1s and 0s. Each byte is preceded by a decimal address.

The file ends with a Control-C character (0x03).

Size Multiplier

In general, binary data will expand in sized by approximately 14 times when represented with this format (or 15 times on DOS or Windows).

EXAMPLE

Here is an example Spectrum file. It contains the data “Hello, World[rq] to be loaded at address 0x0.

```
^B
0000 01001000
0001 01100101
0002 01101100
0003 01101100
0004 01101111
0005 00101100
0006 00100000
0007 01010111
0008 01101111
0009 01110010
0010 01101100
0011 01100100
0012 00100001
0013 00001010
^C
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_stewie − Stewie's binary file format

DESCRIPTION

If you have a URL for documentation of this format, please let me know.

Any resemblance to the Motorola S-Record is superficial, and extends only to the data records. The header records and termination records are completely different. None of the other Motorola S-Records record type are available.

The Records

All records start with an ASCII capital S character, value 0x53, followed by a type specifier byte. All records consist of binary bytes.

The Header Record

Each file starts with a fixed four byte header record.

0x53	0x30	0x30	0x33

The Data Records

Each data record consists of 5 fields. These are the type field, length field, address field, data field, and the checksum. The lines always start with a capital S character.

`0x53`	Type	Record Length	Address	Data	Checksum

Type The type field is a one byte field that specifies whether the record has a two-byte address field (0x31), a three-byte address field (0x32) or a four-byte address field (0x33). The address is big-endian.

Record Length
: The record length field is a one byte field that specifies the number of bytes in the record following this byte.

Address This is a 2-, 3- or 4-byte address that specifies where the data in the record is to be loaded into memory.

Data The data field contains the executable code, memory-loadable data or descriptive information to be transferred.

Checksum
: The checksum is a one byte field that represents the least significant byte of the one's complement of the sum of the values represented by the bytes making up the record's length, address, and data fields.

The Termination Record

Each file ends with a fixed two byte termination record.

0x53	0x38

Size Multiplier

In general, binary data will expand in sized by approximately 1.2 times when represented with this format.

EXAMPLE

Here is an hex-dump example file. It contains the data “Hello, World[rq] to be loaded at address 0.

```
0000: 53 30 30 33 53 31 10 00 00 48 65 6C 6C 6F 2C 20  S003S1...Hello,
0010: 57 6F 72 6C 64 0A 9D 53 38                       World..S8
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_tektronix − Tektronix hexadecimal file format

DESCRIPTION

The Tektronix hexadecimal file format is no longer very common. It serves a similar purpose to the Motorola and Intel formats, usually used to transfer data into EPROM programmers.

The Lines

Most Tektronix hex files contain only Tektronix hex lines (see the next section), which always start with a slash (“/[rq]) character. There are only two types of lines − data lines and a termination line.

Data Lines

Data lines have five fields: address, length, checksum 1, data and checksum 2. The lines always start with a slash (“/[rq]) character.

/	Address	Length	Checksum1	Data	Checksum2

Address This is a 4 character (2 byte) address that specifies where the data in the record is to be loaded into memory.

Data Length
: The data length field is a 2 character (1 byte) field that specifies the number of character pairs (bytes) in the data field. This field never has a value of zero.

Checksum 1
: The checksum 1 field is a 2 character (1 byte) field. Its value is the 8-bit sum of the six 4-bit values which make up the address and length fields.

Data The data field contains character pairs (bytes); the number of character pairs (bytes) is indicated by the length field.

Checksum 2
: The checksum 2 field is a 2 character (1 byte) field. Its value is the least significant byte of the sum of the all the 4-bit values of the data field.

Termination Line

Termination lines have three fields: address, zero and checksum. The lines always start with a slash (“/[rq]) character.

/	Address	Zero	Checksum

Address This is a 4 character (2 byte) address that specifies where to begin execution.

Zero The data length field is a 2 character (1 byte) field of value zero.

Checksum
: The checksum 1 field is a 2 character (1 byte) field. Its value is the 8-bit sum of the six 4-bit values which make up the address and zero fields.

Size Multiplier

In general, binary data will expand in sized by approximately 2.4 times when represented with this format.

EXAMPLE

Here is an example Tektronix hex file. It contains the data “Hello, World[rq] to be loaded at address 0.

```
/00000D0D48656C6C6F2C20576F726C640A52
/00000000
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_tektronix_extended − Tektronix Extended hexadecimal file format

DESCRIPTION

This format allows binary files to be uploaded and downloaded between two computer systems, typically between a computer system (such as a PC, Macintosh, or workstation) and an emulator or evaluation board for microcontrollers and microprocessors.

The Lines

Lines always start with a percent (%) character. Each line consists of 5 fields. These are the length field, the type field, the checksum, the address field (including address length), and the data field.

The Fields

%	Length	Type	Checksum	Address	Data

Record Length

The record length field is a 2 character (1 byte) field that specifies the number of characters (not bytes) in the record, excluding the percent.

Type The type field is a 1 character field that specifies whether the record is data (6) or termination (8).

Checksum

The checksum is an 2 character (1 byte) field that represents the sum of all the nibbles on the line, excluding the checksum.

Address This is a 9 character field. The first character is the address size; it is always 8. The remaining 8 chgaracters are the 4-byte address that specifies where the data is to be loaded into memory.

Data The data field contains the executable code, memory-loadable data or descriptive information to be transferred.

Record Types

6 A record containing data. The data is placed at the address specified.

8 A termination record. The address field may optionally contain the address of the instruction to which control is passed. There is no data field.

Size Multiplier

In general, binary data will expand in sized by approximately 2.5 times when represented with this format.

EXAMPLE

Here is an example Tektronix extended file. It contains the data “Hello, World” to be loaded at address 0x006B.

```
%256D980000006B48656C6C6F2C20576F726C64210A
%09819800000000
```

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_ti_tagged_16 − Texas Instruments Tagged (SDSMAC 320) file format

DESCRIPTION

This format is also known as the *TI-Tagged* or *Texas Instruments SDSMAC (320)* format.

This format allows binary files to be uploaded and downloaded between two computer systems, typically between a computer system (such as a PC, Macintosh, or workstation) and an emulator or evaluation board for 16-bit microcontrollers and microprocessors.

The Lines

Unlike many other object formats, the lines themselves are not especially significant. The format consits of a number of *tagged* fields, and lines are composed of a series of these fields.

Tag	Description
*	Data byte.
:	End of file.
0	File header (optional).
7	Checksum.
8	Dummy checksum (ignored).
9	Word Address.
B	Data word.
F	End of data record.
K	Program identifier (optional).

Data Byte

B	*n*	*n*

One byte of data. The *nn* is 8-bit big-endian hexadecimal.

End of File

:	CRLF

The end of data is indicated by this tag. The end of line sequence (LF on Unix systems, CRLF on PCs) follows this tag.

File Header

0	*length*	*filename*

The optional start-of-file record begins with a tag character (’0’) and a 12-character file header. The first four characters are the count (in hex) of the 16-bit data word values (B) which follow, not including data byte values (*). The remaining file header characters are the name of the file and may be any ASCII characters, blank padded.

Checksum

7	*n*	*n*	*n*	*n*

The checksum is the 2s complement sum of the 8-bit ASCII values of characters, beginning with the first tag character and ending with the checksum tag character (7). The *nnnn* is 16-bit big-endian hexadecimal.

Dummy Checksum

8	*n*	*n*	*n*	*n*

The checksum is the 2s complement sum of the 8-bit ASCII values of characters, beginning with the first tag character and ending with the checksum tag character (8). The *nnnn* is 16-bit big-endian hexadecimal.

Address

9	*n*	*n*	*n*	*n*

Addresses may be given for any data byte, but none is mandatory. The file begins at 0000 if no address is given before the first data field. The *nnnn* is 16-bit big-endian hexadecimal.

Data Word

B	*a*	*a*	*b*	*b*

Two bytes of data. The *aa* and *bb* are each 8-bit big-endian hexadecimal.

End of Record

F	CRLF

The end of line sequence (LF on Unix systems, CRLF on PCs) is escaped using this tag. The checksum is reset to zero at this point.

Program Identifier

K	*n*	*n*	*n*	*n*	*text*

The program identifier can contain a brief description of the program, or can be empty (*i.e.* the text portion is optional). The *nnnn* length (hex) of the field includes the ‘K’, the length and the text; it is at least 5.

Size Multiplier

In general, binary data will expand in sized by approximately 2.9 times when represented with this format.

EXAMPLE

Here is an example TI-Tagged file. It contains the data “Hello, World[rq] to be loaded at address 0x0100.

```
K000590080B4865B6C6CB6F2CB2057B6F72B6C64*0A7F641F
:
```

Here is another example from the reference below

```
00028        7FDCFF
90000BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F400F
90008BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F3F8F
90010BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F3FFF
90018BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F3F7F
90020BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F3FEF
:
```

SEE ALSO

http://www.dataio.com/pdf/Manuals/Unifamily/981-0014-016.pdf (page 6-7)

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_ti_tagged − Texas Instruments Tagged (SDSMAC) file format

DESCRIPTION

This format is also known as the *TI-Tagged* or *TI-SDSMAC* format.

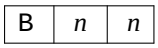
This format allows binary files to be uploaded and downloaded between two computer systems, typically between a computer system (such as a PC, Macintosh, or workstation) and an emulator or evaluation board for microcontrollers and microprocessors.

The Lines

Unlike many other object formats, the lines themselves are not especially significant. The format consits of a number of *tagged* fields, and lines are composed of a series of these fields.

Tag	Description
*	Data byte.
:	End of file.
0	File header (optional).
7	Checksum.
8	Dummy checksum (ignored).
9	Address.
B	Data word.
F	End of data record.
K	Program identifier (optional).

Data Byte



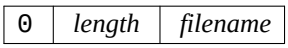
One byte of data. The *nn* is 8-bit big-endian hexadecimal.

End of File



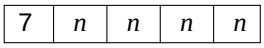
The end of data is indicated by this tag. The end of line sequence (LF on Unix systems, CRLF on PCs) follows this tag.

File Header



The optional start-of-file record begins with a tag character (’0’) and a 12-character file header. The first four characters are the byte count of the file data. The remaining 8 characters are the name of the file and may be any ASCII characters, blank padded.

Checksum



The checksum is the 2s complement sum of the 8-bit ASCII values of characters, beginning with the first tag character and ending with the checksum tag character (7). The *nnnn* is 16-bit big-endian hexadecimal.

Dummy Checksum

8	*n*	*n*	*n*	*n*

The checksum is the 2s complement sum of the 8-bit ASCII values of characters, beginning with the first tag character and ending with the checksum tag character (8). The *nnnn* is 16-bit big-endian hexadecimal.

Address

9	*n*	*n*	*n*	*n*

Addresses may be given for any data byte, but none is mandatory. The file begins at 0000 if no address is given before the first data field. The *nnnn* is 16-bit big-endian hexadecimal.

Data Word

B	*a*	*a*	*b*	*b*

Two bytes of data. The *aa* and *bb* are each 8-bit big-endian hexadecimal.

End of Record

F	CRLF

The end of line sequence (LF on Unix systems, CRLF on PCs) is escaped using this tag. The checksum is reset to zero at this point.

Program Identifier

K	*n*	*n*	*n*	*n*	*text*

The program identifier can contain a brief description of the program, or can be empty (*i.e.* the text portion is optional). The *nnnn* length (hex) of the field includes the ‘K’, the length and the text; it is at least 5.

Size Multiplier

In general, binary data will expand in sized by approximately 2.9 times when represented with this format.

EXAMPLE

Here is an example TI-Tagged file. It contains the data “Hello, World[rq] to be loaded at address 0x0100.

```
K000590080B4865B6C6CB6F2CB2057B6F72B6C64*0A7F648F
:
```

and here is another example from the reference below

```
00050        7FDD4F
90000BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F400F
90010BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F3FFF
90020BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F3FEF
90030BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F3FDF
90040BFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFFBFFFF7F3FCF
:
```

SEE ALSO

http://www.dataio.com/pdf/Manuals/Unifamily/981-0014-016.pdf (page 6-33)

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the ’*srec_cat −VERSion License*’ command. This is free software and you are welcome to redistribute it under certain conditions; for details use the ’*srec_cat −VERSion License*’ command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_ti_txt − Texas Instruments ti-txt (MSP430) file format

DESCRIPTION

The ti-TXT format is used by the Texas Instruments MSP430 familty programming adapter.

The TI-TXT hex format supports 16-bit hexadecimal data. It consists of one or more sections, followed by the end-of-file indicator.

Each section consistes of an at (@) sign followed a execution start address (in hexadecimal), and newline, and then data bytes (in hexadecimal). The section address is followed by a newline. There are to be 16 data bytes per line, except for the last line in a section.

The end-of-file indicator is the letter `q` followed by a newline. The end-of-file indicator mandatory.

Size Multiplier

In general, binary data will expand in sized by approximately 3.0 times when represented with this format.

EXAMPLE

Here is an example ti-txt file taken from the reference below:

```
@F000
31 40 00 03 B2 40 80 5A 20 01 D2 D3 22 00 D2 E3
21 00 3F 40 E8 FD 1F 83 FE 23 F9 3F
@FFFE
00 F0
q
```

SEE ALSO

http://www.ti.com/lit/pdf/slau101, section A.2. **Note:** the portion which says addresses must be even, and the number of data bytes in a section must be even, is wrong.

COPYRIGHT

srec_ti_txt version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_ti_txt* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_ti_txt −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_ti_txt −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_trs80 − Radio Shack TRS-80 object file format specification

DESCRIPTION

This document describes the binary object file format for the Z80-based Radio Shack TRS-80 microcomputers, such as the Model I, II, III, 4, 4D, and 4P. The binary format is generated by the disk-based Assembler/Editor, and used for TRS-DOS program files.

The object file is blocked into records, each of which contains the record type, length, and payload data. For Data and End of File records, the payload starts with two bytes of address in little-endian format. There are four main types of records that are defined. The record types are:

- Data Record
- End of File Record with Execution Transfer
- End of File Record without Execution Transfer
- Comment
- Start Linear Address Record (32-bit format only)

General Record Format

Record Type	Record Length	Load Address	Data

Record Type.

Each record begins with a single byte Record Type field which specifies the record type of this record. The Record Type field is used to interpret the remaining information within the record. This field is one byte, represented as two hexadecimal characters. The encoding for all the current record types are:

1 Data Record

2 End of File Record with Execution Transfer

3 End of File Record without Execution Transfer

5 Comment Record

Record Length

Each record has a single byte Record Length field which specifies the number of bytes of information or data which follows the Record Length field of the record. The maximum value of the Record Length field is hexadecimal “FF” or 255. In the case of Data Records only, Record Length byte values of zero to two are considered to be lengths of 256 to 258, respectively.

Address Data and End records have a two-byte Address field in little-endian byte order. For Data records, this is the starting address at which to load the remaining payload of the record. In End records, this is the address for the start of execution of the file, or zero if not applicable.

Data Each record has a variable length Data field, it consists of zero or more bytes. The interpretation of this field depends on the Record Type field.

REFERENCE

This information comes from the "Program Files" section of *TRSDOS-II Reference Manual*, Tandy Corporation, 1982.

COPYRIGHT

srec_cat version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_vmem − vmem file format

DESCRIPTION

This format is the Verilog VMEM format. This is a hex format suitable for loading into Verilog simulations using the `$readmemh` call.

The text file to be read shall contain only the following:
White space (spaces, new lines, tabs, and form-feeds)
Comments (both types of C++ comment are allowed)
Hexadecimal numbers

White space and/or comments shall be used to separate the numbers.

In the following discussion, the term "address" refers to an index into the array that models the memory.

As the file is read, each number encountered is assigned to a successive word element of the memory. Addressing is controlled both by specifying start and/or finish addresses in the system task invocation and by specifying addresses in the data file.

When addresses appear in the data file, the format is an "at" character (@) followed by a hexadecimal number as follows:

```
@hh...h
```

Both uppercase and lowercase digits are allowed in the number. No white space is allowed between the @ and the number. As many address specifications as needed within the data file can be used. When the system task encounters an address specification, it loads subsequent data starting at that memory address.

Commentary

There is no checksum in this format, which can generate false positives when guessing file formats on input.

There is no indication of the word size in the file, since it is dependent on the word type of the Verilog memory it is being read into. SRecord will guess the word size based on the number of digits it sees in the numbers, but this is only a guess.

SRecord will also assume that the numbers are to be loaded big-endian; that is, most significant byte (first byte seen) into the lowest address covered by the word.

You can use the **−byte-swap** filter to change the byte order; it takes an optional width of bytes to swap within.

Size Multiplier

In general, binary data will expand in sized by approximately 2.9 times (32-bit), 3.1 times (16-bit) or 3.6 times (8-bit) when represented with this format.

EXAMPLE

Here is an example Verilog VMEM file. It contains the data “Hello, World[rq] to be loaded at address 0x1000.

```
@00000400 48656C6C 6F2C2057 6F726C64 0AFFFFFF
```

REFERENCE

IEEE P1364-2005/D2, Standard for Verilog Hardware Description Language (Draft), section 17.2.8 "Loading memory data from a file", p. 295.
Copyright © 2003 IEEE
http://www.boyd.com/1364/
http://www.boyd.com/1364/1364-2005-d2.pdf.gz

COPYRIGHT

srec_cat version 1.64
Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

NAME

srec_wilson − wilson file format

DESCRIPTION

This is a mystery format, added to support a mysery EPROM loader used by Alan Wilson <dvdsales@dvdlibrary.co.uk>

If you know the true name of this format, please let me know! It bears a remarkable similarity to the Motorola S-Record format, however I can find no reference to a "compressed" Motorola format.

The Lines

Each line contains normal ASCII characters, and “high bit on[rq] characters, but the ASCII control characters are avoided (the high-bit-on con characters are not avoided). Normal line termination characters (CRLF or LF, depending on your system) are used.

The presence of high-bit-on characters makes this format unattractive to send via email, as it must be wrapped as a binary attachment, increasing its size.

In general, a single byte per byte is used to encode values, however some values use two bytes, according to the following table:

Byte Value	Encoding (1 or 2 chars)
0x00 .. 0x9F	0x40 .. 0xDF
0xA0 .. 0xAF	0x3A 0x30 .. 0x3A 0x3F
0xB0 .. 0xBF	0x3B 0x30 .. 0x3B 0x3F
0xC0 .. 0xCF	0x3C 0x30 .. 0x3C 0x3F
0xD0 .. 0xDF	0x3D 0x30 .. 0x3D 0x3F
oxE0 .. 0xFF	0xE0 .. 0xFF

The rest of this description, when refering to “bytes[rq] means byte values encoded using the above table.

The Fields

Each line consists of 5 fields. These are the type field, length field, address field, data field, and the checksum.

Type	Record Length	Address	Data	Checksum

Type The type field is a 1 character field that specifies whether the record is data (0x43), or termination (0x47).

Record Length
The record length field is a 1 byte field that specifies the number of bytes in the record, excluding the type and record length fields.

Address This is a 4-byte address that specifies where the data is to be loaded into memory.

Data The data field contains the executable code, memory-loadable data or descriptive information to be transferred.

Checksum
The checksum is an 1-byte field that represents the least significant byte of the one’s complement of the sum of the values represented by the bytes making up the length, address, and data fields.

Record Types

0x43 (#) A record containing data and the 4-byte address at which the data is to reside.

0x47 (') A termination record. The address field may contain the 4-byte address of the instruction to which control is passed. There is no data field.

Size Multiplier

In general, binary data will expand in sized by approximately 1.5 times when represented with this format.

COPYRIGHT

srec_cat version 1.64

Copyright © 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Peter Miller

The *srec_cat* program comes with ABSOLUTELY NO WARRANTY; for details use the '*srec_cat −VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*srec_cat −VERSion License*' command.

MAINTAINER

Scott Finneran E-Mail: scottfinneran@yahoo.com.au
Peter Miller E-Mail: pmiller@opensource.org.au

	The README file	1
	Release Notes	5
	How to build SRecord	16
	How to add a new file format	20
	How to add a new filter	24
srec_cat(1)	Manipulate EPROM load files	27
srec_cmp(1)	Compare two EPROM load files for equality	35
srec_examples(1)	Examples of how to use SRecord	38
srec_info(1)	Information about EPROM load files	51
srec_input(1)	Input file specifications	54
srec_license(1)	GNU General Public License	68
srecord_license(3)	GNU Lesser General Public License	78
srec_aomf(5)	Intel Absolute Object Module Format	81
srec_ascii_hex(5)	Ascii-Hex file format	83
srec_atmel_generic(5)	Atmel Generic file format	84
srec_binary(5)	Binary file format	85
srec_brecord(5)	Freescale MC68EZ328 Dragonball bootstrap record format	87
srec_coe(5)	Xilinx Coefficient File Format	88
srec_cosmac(5)	RCA Cosmac Elf file format	89
srec_dec_binary(5)	DEC Binary (XXDP) file format	90
srec_emon52(5)	Elektor Monitor (EMON52) file format	91
srec_fairchild(5)	Fairchild Fairbug file format	93
srec_fastload(5)	LSI Logic Fast Load file format	94
srec_formatted_binary(5)	Formatted Binary file format	95
srec_forth(5)	FORTH file format	96
srec_fpc(5)	Four Packed Code (FPC) file format	97
srec_idt(5)	IDT/sim binary file format	100
srec_intel16(5)	Intel Hexadecimal 16-bit file format specification	101
srec_intel(5)	Intel Hexadecimal object file format specification	105
srec_logisim(5)	format Logisim EPROM load files	111
srec_mem(5)	Lattice Memory Initialization format	112
srec_mif(5)	Memory Initialization File (MIF) format	114
srec_mips_flash(5)	MIPS-Flash file format	117
srec_mos_tech(5)	MOS Technology file format	119
srec_motorola(5)	Motorola S-Record hexadecimal file format	121
srec_msbin(5)	Windows CE Binary Image Data Format	123
srec_needham(5)	Needham EMP-series programmer ASCII file format	125
srec_os65v(5)	OS65V Loader file format	126
srec_ppb(5)	Stag Prom Programmer binary format	127
srec_ppx(5)	Stag Prom Programmer hexadecimal format	128
srec_signetics(5)	Signetics file format	129
srec_spasm(5)	SPASM file format	131
srec_spectrum(5)	Spectrum file format	132
srec_stewie(5)	Stewie's binary file format	133
srec_tektronix(5)	Tektronix hexadecimal file format	135
srec_tektronix_extended(5)	Tektronix Extended hexadecimal file format	137
srec_ti_tagged_16(5)	Texas Instruments Tagged (SDSMAC 320) file format	138
srec_ti_tagged(5)	Texas Instruments Tagged (SDSMAC) file format	141
srec_ti_txt(5)	Texas Instruments ti-txt (MSP430) file format	144
srec_trs80(5)	Radio Shack TRS-80 object file format specification	145
srec_vmem(5)	VMEM file format	147
srec_wilson(5)	Wilson file format	149

srec_info(1)	51	srec info - information	about EPROM load files
srec_aomf(5)	81	srec aomf - Intel	Absolute Object Module Format
srec_aomf(5)	81	srec	aomf - Intel Absolute Object Module Forma
srec_needham(5)	125	srec needham - Needham EMP-series programmer	ASCII file format
srec_ascii_hex(5)	83	srec	ascii hex - Ascii-Hex file format
srec_ascii_hex(5)	83	srec ascii hex -	Ascii-Hex file format
srec_atmel_generic(5)	84	srec	atmel generic - Atmel Generic file format
srec_atmel_generic(5)	84	srec atmel generic -	Atmel Generic file format
srec_binary(5)	85	srec	binary - binary file format
srec_dec_binary(5)	90	srec dec	binary - DEC Binary (XXDP) file format
srec_binary(5)	85	srec binary -	binary file format
srec_formatted_binary(5)	95	srec formatted binary - Formatted	Binary file format
srec_idt(5)	100	srec idt - IDT/sim	binary file format
srec_stewie(5)	133	srec stewie - Stewie's	binary file format
srec_ppb(5)	127	srec ppb - Stag Prom Programmer	binary format
srec_formatted_binary(5)	95	srec formatted	binary - Formatted Binary file format
srec_msbin(5)	123	srec msbin - Windows CE	Binary Image Data Format
srec_dec_binary(5)	90	srec dec binary - DEC	Binary (XXDP) file format
srec_intel16(5)	101	srec intel16 - Intel Hexadecimal 16-	bit file format specification
srec_brecord(5)	87	srec brecord - Freescale MC68EZ328 Dragonball	bootstrap record format
srec_brecord(5)	87	srec	brecord - Freescale MC68EZ328 Dragonba bootstrap record format
srec_cat(1)	27	srec	cat - manipulate EPROM load files
srec_msbin(5)	123	srec msbin - Windows	CE Binary Image Data Format
srec_cmp(1)	35	srec	cmp - compare two EPROM load files for equality
srec_fpc(5)	97	srec fpc - four packed	code file format
srec_coe(5)	88	srec coe - Xilinx	Coefficient File Format
srec_coe(5)	88	srec	coe - Xilinx Coefficient File Format
srec_cmp(1)	35	srec cmp -	compare two EPROM load files for equality
srec_cosmac(5)	89	srec cosmac - RCA	Cosmac Elf file format
srec_cosmac(5)	89	srec	cosmac - RCA Cosmac Elf file format
srec_msbin(5)	123	srec msbin - Windows CE Binary Image	Data Format
srec_dec_binary(5)	90	srec	dec binary - DEC Binary (XXDP) file format
srec_dec_binary(5)	90	srec dec binary -	DEC Binary (XXDP) file format
srec_brecord(5)	87	srec brecord - Freescale MC68EZ328	Dragonball bootstrap record format
srec_emon52(5)	91	srec emon52 -	Elektor Monitor (EMON52) file format
srec_cosmac(5)	89	srec cosmac - RCA Cosmac	Elf file format
srec_emon52(5)	91	srec	emon52 - Elektor Monitor (EMON52) file format
srec_emon52(5)	91	srec emon52 - Elektor Monitor (	EMON52) file format
srec_needham(5)	125	srec needham - Needham	EMP-series programmer ASCII file format
srec_cat(1)	27	srec cat - manipulate	EPROM load files
srec_info(1)	51	srec info - information about	EPROM load files
srec_logisim(5)	111	srec logisim - format Logisim	EPROM load files
srecord(3)		srecord - library to manipulate	EPROM load files
srec_cmp(1)	35	srec cmp - compare two	EPROM load files for equality
srec_cmp(1)	35	srec cmp - compare two EPROM load files for	equality
srec_examples(1)	38	srec	examples - examples of how to use SRecor

srec_examples(1)	38	srec examples -	examples of how to use SRecord
srec_tektronix_extended(5)	137	srec tektronix extended - Tektronix	Extended hexadecimal file format
srec_tektronix_extended(5)	137	srec tektronix	extended - Tektronix Extended hexadecima file format
srec_brecord(5)	87	srec brecord - Freescale MC68	EZ328 Dragonball bootstrap record format
srec_fairchild(5)	93	srec fairchild - Fairchild	Fairbug file format
srec_fairchild(5)	93	srec fairchild -	Fairchild Fairbug file format
srec_fairchild(5)	93	srec	fairchild - Fairchild Fairbug file format
srec_fastload(5)	94	srec fastload - LSI Logic	Fast Load file format
srec_fastload(5)	94	srec	fastload - LSI Logic Fast Load file format
srec_ascii_hex(5)	83	srec ascii hex - Ascii-Hex	file format
srec_atmel_generic(5)	84	srec atmel generic - Atmel Generic	file format
srec_binary(5)	85	srec binary - binary	file format
srec_coe(5)	88	srec coe - Xilinx Coefficient	File Format
srec_cosmac(5)	89	srec cosmac - RCA Cosmac Elf	file format
srec_dec_binary(5)	90	srec dec binary - DEC Binary (XXDP)	file format
srec_emon52(5)	91	srec emon52 - Elektor Monitor (EMON52)	file format
srec_fairchild(5)	93	srec fairchild - Fairchild Fairbug	file format
srec_fastload(5)	94	srec fastload - LSI Logic Fast Load	file format
srec_formatted_binary(5)	95	srec formatted binary - Formatted Binary	file format
srec_forth(5)	96	srec forth - FORTH	file format
srec_fpc(5)	97	srec fpc - four packed code	file format
srec_idt(5)	100	srec idt - IDT/sim binary	file format
srec_mips_flash(5)	117	srec mips flash - MIPS-Flash	file format
srec_mos_tech(5)	119	srec mos tech - MOS Technology	file format
srec_motorola(5)	121	srec motorola - Motorola S-Record hexadecimal	file format
srec_needham(5)	125	srec needham - Needham EMP-series programmer ASCII	file format
srec_os65v(5)	126	srec os65v - OS65V Loader	file format
srec_signetics(5)	129	srec signetics - Signetics	file format
srec_spasm(5)	131	srec spasm - SPASM	file format
srec_spectrum(5)	132	srec spectrum - Spectrum	file format
srec_stewie(5)	133	srec stewie - Stewie’s binary	file format
srec_tektronix_extended(5)	137	srec tektronix extended - Tektronix Extended hexadecimal	file format
srec_tektronix(5)	135	srec tektronix - Tektronix hexadecimal	file format
srec_ti_tagged_16(5)	138	srec ti tagged 16 - Texas Instruments Tagged (SDSMAC 320)	file format
srec_ti_tagged(5)	141	srec ti tagged - Texas Instruments Tagged (SDSMAC)	file format
srec_ti_txt(5)	144	srec ti txt - Texas Instruments ti-txt (MSP430)	file format
srec_vmem(5)	147	srec vmem - vmem	file format
srec_wilson(5)	149	srec wilson - wilson	file format
srec_intel16(5)	101	srec intel16 - Intel Hexadecimal 16-bit	file format specification
srec_intel(5)	105	srec intel - Intel Hexadecimal object	file format specification
srec_trs80(5)	145	srec trs80 - Radio Shack TRS-80 object	file format specification
srec_mif(5)	114	srec mif - Memory Initialization	File (MIF) format
srec_cat(1)	27	srec cat - manipulate EPROM load	files
srec_info(1)	51	srec info - information about EPROM load	files
srec_logisim(5)	111	srec logisim - format Logisim EPROM load	files
srecord(3)		srecord - library to manipulate EPROM load	files

srec_cmp(1)	35	srec cmp - compare two EPROM load	files for equality
srec_input(1)	54	SRecord - input	file specifications
srec_mips_flash(5)	117	srec mips flash - MIPS-	Flash file format
srec_mips_flash(5)	117	srec mips	flash - MIPS-Flash file format
srec_cmp(1)	35	srec cmp - compare two EPROM load files	for equality
srec_aomf(5)	81	srec aomf - Intel Absolute Object Module	Format
srec_ascii_hex(5)	83	srec ascii hex - Ascii-Hex file	format
srec_atmel_generic(5)	84	srec atmel generic - Atmel Generic file	format
srec_binary(5)	85	srec binary - binary file	format
srec_brecord(5)	87	srec brecord - Freescale MC68EZ328 Dragonball bootstrap record	format
srec_coe(5)	88	srec coe - Xilinx Coefficient File	Format
srec_cosmac(5)	89	srec cosmac - RCA Cosmac Elf file	format
srec_dec_binary(5)	90	srec dec binary - DEC Binary (XXDP) file	format
srec_emon52(5)	91	srec emon52 - Elektor Monitor (EMON52) file	format
srec_fairchild(5)	93	srec fairchild - Fairchild Fairbug file	format
srec_fastload(5)	94	srec fastload - LSI Logic Fast Load file	format
srec_formatted_binary(5)	95	srec formatted binary - Formatted Binary file	format
srec_forth(5)	96	srec forth - FORTH file	format
srec_fpc(5)	97	srec fpc - four packed code file	format
srec_idt(5)	100	srec idt - IDT/sim binary file	format
srec_mem(5)	112	srec mem - Lattice Memory Initialization	format
srec_mif(5)	114	srec mif - Memory Initialization File (MIF)	format
srec_mips_flash(5)	117	srec mips flash - MIPS-Flash file	format
srec_mos_tech(5)	119	srec mos tech - MOS Technology file	format
srec_motorola(5)	121	srec motorola - Motorola S-Record hexadecimal file	format
srec_msbin(5)	123	srec msbin - Windows CE Binary Image Data	Format
srec_needham(5)	125	srec needham - Needham EMP-series programmer ASCII file	format
srec_os65v(5)	126	srec os65v - OS65V Loader file	format
srec_ppb(5)	127	srec ppb - Stag Prom Programmer binary	format
srec_ppx(5)	128	srec ppx - Stag Prom Programmer hexadecimal	format
srec_signetics(5)	129	srec signetics - Signetics file	format
srec_spasm(5)	131	srec spasm - SPASM file	format
srec_spectrum(5)	132	srec spectrum - Spectrum file	format
srec_stewie(5)	133	srec stewie - Stewie’s binary file	format
srec_tektronix_extended(5)	137	srec tektronix extended - Tektronix Extended hexadecimal file	format
srec_tektronix(5)	135	srec tektronix - Tektronix hexadecimal file	format
srec_ti_tagged_16(5)	138	srec ti tagged 16 - Texas Instruments Tagged (SDSMAC 320) file	format
srec_ti_tagged(5)	141	srec ti tagged - Texas Instruments Tagged (SDSMAC) file	format
srec_ti_txt(5)	144	srec ti txt - Texas Instruments ti-txt (MSP430) file	format
srec_vmem(5)	147	srec vmem - vmem file	format
srec_wilson(5)	149	srec wilson - wilson file	format
srec_logisim(5)	111	srec logisim -	format Logisim EPROM load files

srec_intel16(5)	101	srec intel16 - Intel Hexadecimal 16-bit file	format specification
srec_intel(5)	105	srec intel - Intel Hexadecimal object file	format specification
srec_trs80(5)	145	srec trs80 - Radio Shack TRS-80 object file	format specification
srec_formatted_binary(5)	95	srec formatted binary -	Formatted Binary file format
srec_formatted_binary(5)	95	srec	formatted binary - Formatted Binary file format
srec_forth(5)	96	srec forth -	FORTH file format
srec_forth(5)	96	srec	forth - FORTH file format
srec_fpc(5)	97	srec fpc -	four packed code file format
srec_fpc(5)	97	srec	fpc - four packed code file format
srec_brecord(5)	87	srec brecord -	Freescale MC68EZ328 Dragonball bootstrap record format
srecord_license(3)	78	LGPG - GNU Lesser	General Public License
srec_atmel_generic(5)	84	srec atmel	generic - Atmel Generic file format
srec_atmel_generic(5)	84	srec atmel generic - Atmel	Generic file format
srecord_license(3)	78	LGPG -	GNU Lesser General Public License
srec_intel16(5)	101	srec intel16 - Intel	Hexadecimal 16-bit file format specificatio
srec_motorola(5)	121	srec motorola - Motorola S-Record	hexadecimal file format
srec_tektronix_extended(5)	137	srec tektronix extended - Tektronix Extended	hexadecimal file format
srec_tektronix(5)	135	srec tektronix - Tektronix	hexadecimal file format
srec_ppx(5)	128	srec ppx - Stag Prom Programmer	hexadecimal format
srec_intel(5)	105	srec intel - Intel	Hexadecimal object file format specificatio
srec_ascii_hex(5)	83	srec ascii	hex - Ascii-Hex file format
srec_ascii_hex(5)	83	srec ascii hex - Ascii-	Hex file format
srec_examples(1)	38	srec examples - examples of	how to use SRecord
srec_idt(5)	100	srec	idt - IDT/sim binary file format
srec_idt(5)	100	srec idt -	IDT/sim binary file format
srec_msbin(5)	123	srec msbin - Windows CE Binary	Image Data Format
srec_info(1)	51	srec	info - information about EPROM load files
srec_info(1)	51	srec info -	information about EPROM load files
srec_mif(5)	114	srec mif - Memory	Initialization File (MIF) format
srec_mem(5)	112	srec mem - Lattice Memory	Initialization format
srec_input(1)	54	SRecord -	input file specifications
srec_ti_tagged_16(5)	138	srec ti tagged 16 - Texas	Instruments Tagged (SDSMAC 320) file format
srec_ti_tagged(5)	141	srec ti tagged - Texas	Instruments Tagged (SDSMAC) file format
srec_ti_txt(5)	144	srec ti txt - Texas	Instruments ti-txt (MSP430) file format
srec_intel16(5)	101	srec	intel16 - Intel Hexadecimal 16-bit file format specification
srec_aomf(5)	81	srec aomf -	Intel Absolute Object Module Format
srec_intel16(5)	101	srec intel16 -	Intel Hexadecimal 16-bit file format specification
srec_intel(5)	105	srec intel -	Intel Hexadecimal object file format specification
srec_intel(5)	105	srec	intel - Intel Hexadecimal object file format specification
srec_mem(5)	112	srec mem -	Lattice Memory Initialization format
srecord_license(3)	78	LGPG - GNU	Lesser General Public License
srecord_license(3)	78		LGPG - GNU Lesser General Public License
srecord(3)		srecord -	library to manipulate EPROM load files
srecord_license(3)	78	LGPG - GNU Lesser General Public	License

srec_os65v(5)	126	srec os65v - OS65V	Loader file format
srec_fastload(5)	94	srec fastload - LSI Logic Fast	Load file format
srec_cat(1)	27	srec cat - manipulate EPROM	load files
srec_info(1)	51	srec info - information about EPROM	load files
srec_logisim(5)	111	srec logisim - format Logisim EPROM	load files
srecord(3)		srecord - library to manipulate EPROM	load files
srec_cmp(1)	35	srec cmp - compare two EPROM	load files for equality
srec_fastload(5)	94	srec fastload - LSI	Logic Fast Load file format
srec_logisim(5)	111	srec logisim - format	Logisim EPROM load files
srec_logisim(5)	111	srec	logisim - format Logisim EPROM load file
srec_fastload(5)	94	srec fastload -	LSI Logic Fast Load file format
srec_cat(1)	27	srec cat -	manipulate EPROM load files
srecord(3)		srecord - library to	manipulate EPROM load files
srec_brecord(5)	87	srec brecord - Freescale	MC68EZ328 Dragonball bootstrap record format
srec_mem(5)	112	srec	mem - Lattice Memory Initialization forma
srec_mif(5)	114	srec mif -	Memory Initialization File (MIF) format
srec_mem(5)	112	srec mem - Lattice	Memory Initialization format
srec_mif(5)	114	srec mif - Memory Initialization File (	MIF) format
srec_mif(5)	114	srec	mif - Memory Initialization File (MIF) format
srec_mips_flash(5)	117	srec mips flash -	MIPS-Flash file format
srec_mips_flash(5)	117	srec	mips flash - MIPS-Flash file format
srec_aomf(5)	81	srec aomf - Intel Absolute Object	Module Format
srec_emon52(5)	91	srec emon52 - Elektor	Monitor (EMON52) file format
srec_mos_tech(5)	119	srec	mos tech - MOS Technology file format
srec_mos_tech(5)	119	srec mos tech -	MOS Technology file format
srec_motorola(5)	121	srec	motorola - Motorola S-Record hexadecimal file format
srec_motorola(5)	121	srec motorola -	Motorola S-Record hexadecimal file forma
srec_msbin(5)	123	srec	msbin - Windows CE Binary Image Data Format
srec_ti_txt(5)	144	srec ti txt - Texas Instruments ti-txt (	MSP430) file format
srec_needham(5)	125	srec needham -	Needham EMP-series programmer ASCII file format
srec_needham(5)	125	srec	needham - Needham EMP-series programmer ASCII file format
srec_intel(5)	105	srec intel - Intel Hexadecimal	object file format specification
srec_trs80(5)	145	srec trs80 - Radio Shack TRS-80	object file format specification
srec_aomf(5)	81	srec aomf - Intel Absolute	Object Module Format
srec_os65v(5)	126	srec os65v -	OS65V Loader file format
srec_os65v(5)	126	srec	os65v - OS65V Loader file format
srec_fpc(5)	97	srec fpc - four	packed code file format
srec_ppb(5)	127	srec	ppb - Stag Prom Programmer binary format
srec_ppx(5)	128	srec	ppx - Stag Prom Programmer hexadecimal format
srec_needham(5)	125	srec needham - Needham EMP-series	programmer ASCII file format
srec_ppb(5)	127	srec ppb - Stag Prom	Programmer binary format
srec_ppx(5)	128	srec ppx - Stag Prom	Programmer hexadecimal format
srec_ppb(5)	127	srec ppb - Stag	Prom Programmer binary format
srec_ppx(5)	128	srec ppx - Stag	Prom Programmer hexadecimal format
srecord_license(3)	78	LGPG - GNU Lesser General	Public License

srec_trs80(5)	145	srec trs80 -	Radio Shack TRS-80 object file format specification
srec_cosmac(5)	89	srec cosmac -	RCA Cosmac Elf file format
srec_brecord(5)	87	srec brecord - Freescale MC68EZ328 Dragonball bootstrap	record format
srec_motorola(5)	121	srec motorola - Motorola S-	Record hexadecimal file format
srec_stewie(5)	133	srec stewie - Stewie’	s binary file format
srec_ti_tagged_16(5)	138	srec ti tagged 16 - Texas Instruments Tagged (	SDSMAC 320) file format
srec_ti_tagged(5)	141	srec ti tagged - Texas Instruments Tagged (	SDSMAC) file format
srec_needham(5)	125	srec needham - Needham EMP-	series programmer ASCII file format
srec_trs80(5)	145	srec trs80 - Radio	Shack TRS-80 object file format specification
srec_signetics(5)	129	srec signetics -	Signetics file format
srec_signetics(5)	129	srec	signetics - Signetics file format
srec_idt(5)	100	srec idt - IDT/	sim binary file format
srec_spasm(5)	131	srec spasm -	SPASM file format
srec_spasm(5)	131	srec	spasm - SPASM file format
srec_intel16(5)	101	srec intel16 - Intel Hexadecimal 16-bit file format	specification
srec_intel(5)	105	srec intel - Intel Hexadecimal object file format	specification
srec_trs80(5)	145	srec trs80 - Radio Shack TRS-80 object file format	specification
srec_input(1)	54	SRecord - input file	specifications
srec_spectrum(5)	132	srec spectrum -	Spectrum file format
srec_spectrum(5)	132	srec	spectrum - Spectrum file format
srec_aomf(5)	81		srec aomf - Intel Absolute Object Module Format
srec_ascii_hex(5)	83		srec ascii hex - Ascii-Hex file format
srec_atmel_generic(5)	84		srec atmel generic - Atmel Generic file format
srec_binary(5)	85		srec binary - binary file format
srec_brecord(5)	87		srec brecord - Freescale MC68EZ328 Dragonball bootstrap record format
srec_cat(1)	27		srec cat - manipulate EPROM load files
srec_cmp(1)	35		srec cmp - compare two EPROM load files for equality
srec_coe(5)	88		srec coe - Xilinx Coefficient File Format
srec_cosmac(5)	89		srec cosmac - RCA Cosmac Elf file format
srec_dec_binary(5)	90		srec dec binary - DEC Binary (XXDP) file format
srec_emon52(5)	91		srec emon52 - Elektor Monitor (EMON52) file format
srec_examples(1)	38		srec examples - examples of how to use SRecord
srec_fairchild(5)	93		srec fairchild - Fairchild Fairbug file format
srec_fastload(5)	94		srec fastload - LSI Logic Fast Load file format
srec_formatted_binary(5)	95		srec formatted binary - Formatted Binary file format
srec_forth(5)	96		srec forth - FORTH file format
srec_fpc(5)	97		srec fpc - four packed code file format

srec_idt(5)	100		srec idt - IDT/sim binary file format
srec_info(1)	51		srec info - information about EPROM load files
srec_intel16(5)	101		srec intel16 - Intel Hexadecimal 16-bit file format specification
srec_intel(5)	105		srec intel - Intel Hexadecimal object file format specification
srec_logisim(5)	111		srec logisim - format Logisim EPROM load files
srec_mem(5)	112		srec mem - Lattice Memory Initialization format
srec_mif(5)	114		srec mif - Memory Initialization File (MIF) format
srec_mips_flash(5)	117		srec mips flash - MIPS-Flash file format
srec_mos_tech(5)	119		srec mos tech - MOS Technology file forma
srec_motorola(5)	121		srec motorola - Motorola S-Record hexadecimal file format
srec_msbin(5)	123		srec msbin - Windows CE Binary Image Data Format
srec_needham(5)	125		srec needham - Needham EMP-series programmer ASCII file format
srec_examples(1)	38	srec examples - examples of how to use	SRecord
srec_motorola(5)	121	srec motorola - Motorola	S-Record hexadecimal file format
srec_input(1)	54		SRecord - input file specifications
srecord(3)			srecord - library to manipulate EPROM loa files
srec_os65v(5)	126		srec os65v - OS65V Loader file format
srec_ppb(5)	127		srec ppb - Stag Prom Programmer binary format
srec_ppx(5)	128		srec ppx - Stag Prom Programmer hexadecimal format
srec_signetics(5)	129		srec signetics - Signetics file format
srec_spasm(5)	131		srec spasm - SPASM file format
srec_spectrum(5)	132		srec spectrum - Spectrum file format
srec_stewie(5)	133		srec stewie - Stewie’s binary file format
srec_tektronix_extended(5)	137		srec tektronix extended - Tektronix Extended hexadecimal file format
srec_tektronix(5)	135		srec tektronix - Tektronix hexadecimal file format
srec_ti_tagged_16(5)	138		srec ti tagged 16 - Texas Instruments Tagge (SDSMAC 320) file format
srec_ti_tagged(5)	141		srec ti tagged - Texas Instruments Tagged (SDSMAC) file format
srec_ti_txt(5)	144		srec ti txt - Texas Instruments ti-txt (MSP430) file format
srec_trs80(5)	145		srec trs80 - Radio Shack TRS-80 object file format specification
srec_vmem(5)	147		srec vmem - vmem file format
srec_wilson(5)	149		srec wilson - wilson file format
srec_ppb(5)	127	srec ppb -	Stag Prom Programmer binary format
srec_ppx(5)	128	srec ppx -	Stag Prom Programmer hexadecimal forma
srec_stewie(5)	133	srec stewie -	Stewie’s binary file format
srec_stewie(5)	133	srec	stewie - Stewie’s binary file format

srec_ti_tagged_16(5)	138	srec ti	tagged 16 - Texas Instruments Tagged (SDSMAC 320) file format
srec_ti_tagged_16(5)	138	srec ti tagged 16 - Texas Instruments	Tagged (SDSMAC 320) file format
srec_ti_tagged(5)	141	srec ti tagged - Texas Instruments	Tagged (SDSMAC) file format
srec_ti_tagged(5)	141	srec ti	tagged - Texas Instruments Tagged (SDSMAC) file format
srec_mos_tech(5)	119	srec mos	tech - MOS Technology file format
srec_mos_tech(5)	119	srec mos tech - MOS	Technology file format
srec_tektronix_extended(5)	137	srec tektronix extended -	Tektronix Extended hexadecimal file forma
srec_tektronix_extended(5)	137	srec	tektronix extended - Tektronix Extended hexadecimal file format
srec_tektronix(5)	135	srec tektronix -	Tektronix hexadecimal file format
srec_tektronix(5)	135	srec	tektronix - Tektronix hexadecimal file format
srec_ti_tagged_16(5)	138	srec ti tagged 16 -	Texas Instruments Tagged (SDSMAC 320) file format
srec_ti_tagged(5)	141	srec ti tagged -	Texas Instruments Tagged (SDSMAC) file format
srec_ti_txt(5)	144	srec ti txt -	Texas Instruments ti-txt (MSP430) file format
srec_ti_tagged_16(5)	138	srec	ti tagged 16 - Texas Instruments Tagged (SDSMAC 320) file format
srec_ti_tagged(5)	141	srec	ti tagged - Texas Instruments Tagged (SDSMAC) file format
srec_ti_txt(5)	144	srec ti txt - Texas Instruments	ti-txt (MSP430) file format
srec_ti_txt(5)	144	srec	ti txt - Texas Instruments ti-txt (MSP430) file format
srec_trs80(5)	145	srec trs80 - Radio Shack	TRS-80 object file format specification
srec_trs80(5)	145	srec	trs80 - Radio Shack TRS-80 object file format specification
srec_cmp(1)	35	srec cmp - compare	two EPROM load files for equality
srec_ti_txt(5)	144	srec ti txt - Texas Instruments ti-	txt (MSP430) file format
srec_ti_txt(5)	144	srec ti	txt - Texas Instruments ti-txt (MSP430) file format
srec_examples(1)	38	srec examples - examples of how to	use SRecord
srec_os65v(5)	126	srec os65v - OS65	V Loader file format
srec_vmem(5)	147	srec vmem -	vmem file format
srec_vmem(5)	147	srec	vmem - vmem file format
srec_os65v(5)	126	srec os65	v - OS65V Loader file format
srec_wilson(5)	149	srec wilson -	wilson file format
srec_wilson(5)	149	srec	wilson - wilson file format
srec_msbin(5)	123	srec msbin -	Windows CE Binary Image Data Format
srec_coe(5)	88	srec coe -	Xilinx Coefficient File Format
srec_dec_binary(5)	90	srec dec binary - DEC Binary (	XXDP) file format