SRecord

Reference Manual

Peter Miller
millerp@canb.auug.org.au

.

This document describes SRecord version 1.21
and was prepared 11 April 2005.

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.

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.

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.

LSI Logic Fast Load
: The LSI Logic Fast Load 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.

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.)

MOS Technology
: The MOS Technology hexadecimal format is understood 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.)

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

OS65V
: The Ohio Scientific 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. (Also known as the TI-tagged or TI-SDSMAC format.)

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:

checksum
: The *checksum* filter may be used to insert the checksum of the data (bitnot, negative or positive) into the data.

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

CRC
: The *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.

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

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

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.

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

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:	srecord.html	# the SRecord page
File:	srecord-1.21.README	# Description, from the tar file
File:	srecord-1.21.lsm	# Description, LSM format
File:	srecord-1.21.spec	# RedHat package specification
File:	srecord-1.21.tar.gz	# the complete source
File:	srecord-1.21.pdf	# Reference Manual

This Web page also contains a few other pieces of software written by me. Please have a look if you are interested.

SRecord is also carried by `sunsite.unc.edu` in its Linux archives. You will be able to find SRecord on any of its mirrors.

URL:	ftp://sunsite.unc.edu/pub/Linux/apps/circuits/	
File:	srecord-1.21.README	# Description, from the tar file
File:	srecord-1.21.lsm	# Description, LSM format
File:	srecord-1.21.spec	# RedHat package specification
File:	srecord-1.21.tar.gz	# the complete source
File:	srecord-1.21.pdf	# Reference Manual

This site is extensively mirrored around the world, so look for a copy near you (you will get much better response).

FTP by EMail

For those of you without Web or FTP access, I recommend the use of an ftp-by-email server. Here is a list of a few (there may be more):

ftpmail@cs.uow.edu.au	Australia
ftpmail@ftp.uni-stuttgart.de	Germany
ftpmail@grasp.insa-lyon.fr	France
ftpmail@doc.ic.ac.uk	Great Britain
ftpmail@ieunet.ie	Ireland

ftpmail@sunsite.unc.edu	USA
ftpmail@ftp.uu.net	USA

In general, you can get a help message about how to use each system by sending email with a subject of "help" and a message body containing just the word "help".

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

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
`/\/\*`	WWW:	http://www.canb.auug.org.au/~millerp/

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 1.21 (7-Feb-2005)

• 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 (8-Feb-2004)

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

Version 1.19 (3-Jan-2004)

• It is now possible to set the start address in the output using the *srec_cat −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 (1-Jan-2004)

• 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 (12-Oct-2003)

• There is now support for Intel Extended Segment aasddressing 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 (28-Jul-2003)

• Some updates have been made to cope with GCC 3.2

Version 1.15 (16-Jun-2003)

• 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 (11-Mar-2003)

• 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 (5-Feb-2003)

• 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 (6-Dec-2002)

• 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 (21-Oct-2002)

• 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 (14-Jun-2002)

• 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 (27-Nov-2001)

• 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 (20-Apr-2001)

• 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 (19-Mar-2001)

• 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 --address-length=4. 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 (3-Dec-2000)

- 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 (bitnot) 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 (6-Mar-200)

- 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 supresses 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 (13-Jan-2000)

- 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 (12-May-1999)

• 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 (acually, 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 (4-Nov-1998)

• 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 suppported). 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 (22-Mar-1998)

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.

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 *include/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 *include/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.

Note: if you are using GCC 3.*x* where x < 4, you may need to edit the *Makefile* to change `CXX = g++` to read `CXX = g++-2` to fix some weird undefined symbols. This appears to be a bug in these versions of GCC. The bug has apparently been fixed in GCC 3.4 and above.

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 do not hesitate to contact the author at

```
Peter Miller &lt;millerp@canb.auug.org.au&gt;
```

Any and all feedback is welcome.

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

```
% srec_cat -version
srecord version 1.21.D002
...warranty disclaimer...
%
```

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

COPYRIGHT

srecord version 1.21

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

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 -u -r, usually) and it can be added to the sources if appropriate.

New Files

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

lib/srec/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 *include/srec/output/file.h* and *include/srec/output.h* for various helper methods.

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

lib/srec/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 *include/srec/input/file.h* and *include/srec/input.h* for various helper methods.

include/srec/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.

Modified Files

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

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

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

include/srec/arglex.h
: Add the new format to the command line argument type enum.

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

lib/srec/arglex/input.cc
: Add the new format to the code which parses input formats.

lib/srec/arglex/output.cc
: Add the new format to the code which parses output formats.

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

man/man1/o_input.so
: 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.

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

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.

A warning will be emitted for each address wich is redundantly se to the same value. A fatal error will be issued if any address is set with contradictory values. To supress this behaviour, use an **−exclude −within** filter.

INPUT FILE SPECIFICATIONS

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*][*filter* ...]

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

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 specified 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. (May also be written −Raw.)

−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 uased to ask srec_cat to guess the input format. This is slower than specifying an explicit format, as it may open and close the file a number of times.

−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.

−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
This option says to use the Motorola S-Record format to read the file. (May also be written −S-Record.) See *srec_motorola*(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.

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

−SPAsm_LittleEndian
This option says to use the SPASM assembler output format (commonly used by PIC programmers). 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.

−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.

Input Filters

You may specify zero or more *filters* to be applied. Filters are applied in the order the user specifies.

−Big_Endian_Checksum_BitNot *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 literaly 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.

−Big_Endian_Checksum_Negative *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.

−Big_Endian_Checksum_Positive *address* [*nbytes* [*width*]]
This filter may be used to insert the simple checksum of the data into the data. Otherwise similar to the above.

−Little_Endian_Checksum_BitNot *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.

−Little_Endian_Checksum_Negative *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.

−Little_Endian_Checksum_Negative *address* [*nbytes* [*width*]]
This filter may be used to insert the simple checksum of the data into the data. Otherwise similar to the above.

−Byte_Swap
This filter may be used to swap pairs of odd and even bytes.

−Big_Endian_CRC16 *address*
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).

−Little_Endian_CRC16 *address*
As above, except little-endian order.

−Big_Endian_CRC32 *address*
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).

−Little_Endian_CRC32 *address*
As above, except little-endian 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.

−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.

−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).

−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.

−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.

−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.

−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.

−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.

−Big_Endian_Length *address* [*nbytes*]
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.

−Little_Endian_Length *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

−Big_Endian_MAXimum *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.

−Little_Endian_MAXimum *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

−Big_Endian_MINimum *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.

−Little_Endian_MINimum *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

−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.

−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.

−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.

Address Ranges

There are three 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.

−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, and ignores any holes. The input specification need not be just a file name, it may be anything any other input specification can be.

In addition, all of these methods may be used, and used more than once, and the results will be added together.

Calculated Values

Most of the places above where a number is expected, you may supply one of the following:

−MINimum *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.

−MAXimum *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.

−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.

For example, the **-OVER** *file* option can be thought of a short-hand for **(** **−min** *file* **−max** *file*)i, 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*.

OPTIONS

The following options are understood:

−Output *filename* [*format*]
This option may be used to specify the output file to be used. The special file name "-" 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 A series of assembler DB statements will be written.

−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.

−Binary
A raw binary file will be written.

−C-Array *identifier*
A C array declaration will be written. The *identifier* is the name of the variable to be defined.

−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.)

–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.)

–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 32-bit linear addressing; if you want 16-bit extended segment addressing use the **--address-length=2** option.

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

–Motorola
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

–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.

–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_LittleEndian
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.)

–VHdl [*bytes-per-word* [*name*]]
A VHDL format file will be written. The *bytes-per-word* defaaults 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.

–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.)

–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 brain-dead EPROM programmers which do not fully implement the format specification.

–Data_Only
This option may be used to suppress all output except data fields. This helps when talking to brain-dead EPROM programmers which do not fully implement the format specification.

–CRLF This option may be used to specify CRLF line termination for text output. For use with brain-dead EPROM programmers which assume all the world uses Evil Bill's operating system's line termination. The default is to use the current operating system's default line termination. Use this option with caution, because it will also introduce extra (i.e. wrong) CR bytes into binary formats.

-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.

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

–Start_Address *number*
This option may be used to set the start address, in those formats which support it.

–MULTiple
Use this option to permit a file to contain multiple (contradictory) values for some memory locations. A warning will be printed. The last value in the file will be used. The default is for this condition to be a fatal error.

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.

EXAMPLES

The *srec_cat* command is very powerful, due to the ability to combine the the input filters in almost unlimited ways.

Converting File Formats

The simplest case is converting files from Intel hex format to Motorola S-Record format:

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

Converting the other was is just as simple:

`srec_cat` *srec-file* `-o` *intel-file* `-intel`

In each case, the default format is Motorola S-Record format, so it does not need to be specified.

Cropping the Data

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 is inclusive, the upper bound is exclusive.

Address Offset

Just possibly, you have a moronic EPROM programmer, and it barfs if the eprom doesn't start at zero. Rather than butcher the linker command file, just offset the addresses:

`srec_cat` *infile* `-crop 0x200000 0x300000 -offset -0x200000 -o` *outfile*

This example also demonstrates how the input filters may be chained together.

Joining Files Together

The *srec_cat* command takes its name from the UNIX *cat*(1) command, which is short for 'catenate' or 'to join'. Joining 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*

However, this assumes that the files don't overlap in any way (you will get an error if they do). If both files start from address zero, 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` *infile1* `-o` *outfile*

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

Filling the Blanks

It is possible to fill the blanks where our 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` *outile* `-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.

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.

A second use for the −SPlit filter is memory striping. In this 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
```

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. For example, to reverse our previous 16-bit data bus example, use the following command:

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

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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 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*][*filter* ...]

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

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 specified 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. (May also be written −Raw.)

−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 uased to ask srec_cmp to guess the input format. This is slower than specifying an explicit format, as it may open and close the file a number of times.

−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.

−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
This option says to use the Motorola S-Record format to read the file. (May also be written −S-Record.) See *srec_motorola*(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.

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

−SPAsm_LittleEndian
This option says to use the SPASM assembler output format (commonly used by PIC programmers). 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.

−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.

Input Filters

You may specify zero or more *filters* to be applied. Filters are applied in the order the user specifies.

−Big_Endian_Checksum_BitNot *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 literaly 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.

−Big_Endian_Checksum_Negative *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.

−Big_Endian_Checksum_Positive *address* [*nbytes* [*width*]]
This filter may be used to insert the simple checksum of the data into the data. Otherwise similar to the above.

−Little_Endian_Checksum_BitNot *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.

−Little_Endian_Checksum_Negative *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.

−Little_Endian_Checksum_Negative *address* [*nbytes* [*width*]]
This filter may be used to insert the simple checksum of the data into the data. Otherwise similar to the above.

−Byte_Swap
This filter may be used to swap pairs of odd and even bytes.

−Big_Endian_CRC16 *address*
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).

−Little_Endian_CRC16 *address*
As above, except little-endian order.

−Big_Endian_CRC32 *address*
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).

−Little_Endian_CRC32 *address*
As above, except little-endian 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.

–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.

–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).

–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.

–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.

–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.

–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.

–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.

–Big_Endian_Length *address* [*nbytes*]
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.

–Little_Endian_Length *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

–Big_Endian_MAXimum *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.

–Little_Endian_MAXimum *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

–Big_Endian_MINimum *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.

–Little_Endian_MINimum *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

–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.

–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.

−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.

Address Ranges

There are three 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.

−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, and ignores any holes. The input specification need not be just a file name, it may be anything any other input specification can be.

In addition, all of these methods may be used, and used more than once, and the results will be added together.

Calculated Values

Most of the places above where a number is expected, you may supply one of the following:

−MINimum *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.

−MAXimum *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.

−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.

For example, the **-OVER** *file* option can be thought of a short-hand for **(** **−min** *file* **−max** *file* **)**i, 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*.

OPTIONS

The following options are understood:

-Help
Provide some help with using the *srec_cmp* program.

−MULTiple
Use this option to permit a file to contain multiple (contradictory) values for some memory locations. A warning will be printed. The last value in the file will be used. The default is for this condition to be a fatal error.

-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’’, and the EPROM image is in a file called ‘‘image’’. 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 1.21

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.

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

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 start address if any, and the address ranges covered by the data if any.

INPUT FILE SPECIFICATIONS

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*][*filter* ...]

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

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 specified 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. (May also be written −Raw.)

−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 uased to ask srec_info to guess the input format. This is slower than specifying an explicit format, as it may open and close the file a number of times.

−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.

−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
This option says to use the Motorola S-Record format to read the file. (May also be written **−**S-Record.) See *srec_motorola*(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.

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

−SPAsm_LittleEndian
This option says to use the SPASM assembler output format (commonly used by PIC programmers). 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.

−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.

Input Filters

You may specify zero or more *filters* to be applied. Filters are applied in the order the user specifies.

−Big_Endian_Checksum_BitNot *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 literaly 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.

−Big_Endian_Checksum_Negative *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.

−Big_Endian_Checksum_Positive *address* [*nbytes* [*width*]]
This filter may be used to insert the simple checksum of the data into the data. Otherwise similar to the above.

−Little_Endian_Checksum_BitNot *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.

−Little_Endian_Checksum_Negative *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.

−Little_Endian_Checksum_Negative *address* [*nbytes* [*width*]]
This filter may be used to insert the simple checksum of the data into the data. Otherwise similar to the above.

−Byte_Swap
This filter may be used to swap pairs of odd and even bytes.

−Big_Endian_CRC16 *address*
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).

−Little_Endian_CRC16 *address*
As above, except little-endian order.

−Big_Endian_CRC32 *address*
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).

−Little_Endian_CRC32 *address*
As above, except little-endian 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.

−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.

−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).

−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.

−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.

−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.

−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.

−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.

−Big_Endian_Length *address* [*nbytes*]
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.

−Little_Endian_Length *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

−Big_Endian_MAXimum *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.

−Little_Endian_MAXimum *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

−Big_Endian_MINimum *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.

−Little_Endian_MINimum *address* [*nbytes*]
As above, however the value will be written with the least significant byte first.

−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.

−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.

−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.

Address Ranges

There are three 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.

−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, and ignores any holes. The input specification need not be just a file name, it may be anything any other input specification can be.

In addition, all of these methods may be used, and used more than once, and the results will be added together.

Calculated Values

Most of the places above where a number is expected, you may supply one of the following:

−MINimum *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.

−MAXimum *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.

−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.

For example, the **-OVER** *file* option can be thought of a short-hand for **(** **−min** *file* **−max** *file*)i, 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*.

OPTIONS

The following options are understood:

-Help
Provide some help with using the *srec_info* program.

–MULTiple
Use this option to permit a file to contain multiple (contradictory) values for some memory locations. A warning will be printed. The last value in the file will be used. The default is for this condition to be a fatal error.

-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

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.

AUTHOR

Peter Miller E-Mail: millerp@canb.auug.org.au
/\/* WWW: http://www.canb.auug.org.au/~millerp/

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. 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.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, 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.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE 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.

END OF TERMS AND CONDITIONS

Appendix: 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 convey 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) 19yy *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 2 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, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) 19yy name of author
> Gnomovision 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, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> *signature of Ty Coon,* 1 April 1989
> Ty Coon, President of Vice

This 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 Library General Public License instead of this License.

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 a 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 1.21

AUTHOR

Peter Miller E-Mail: millerp@canb.auug.org.au
/\/* WWW: http://www.canb.auug.org.au/~millerp/

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, followd by an "execution character". The default execution character is a space, although many programs which write this format omit the space character immediately preceeding 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” 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 1.21

AUTHOR

Peter Miller E-Mail: millerp@canb.auug.org.au
/\/* WWW: http://www.canb.auug.org.au/~millerp/

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’’ 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

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

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’’ to be loaded at address 0x1000.

```
!M1000 48656C6C6F2C20576F726C640A
```

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

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 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

AUTHOR

Peter Miller E-Mail: millerp@canb.auug.org.au
/\/* WWW: http://www.canb.auug.org.au/˜millerp/

NAME

srec_emon52 − Elektor Monitor (EMON52) file format

DESCRIPTION

This format is used by the monitor EMON52, devolped 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 seperated 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 seperated 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 seperated from each other (and the checksum) by a space.

ssss Data Checksum, adding all bytes of the dataline 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

S	*nnnn*

indicating the address for the following data records.

Data records are of the form

X	*ffffffffffffffff*	*c*

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’’ to be loaded at address 0x1000. Notice how the last record is padded with 0xFF bytes.

```
S1000
X48656C6C6F2C2057C
X6F726C64210AFFFF3
*
```

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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" to be loaded at address 0.

```
/AAAA
SGVsbG8sIFdvcmxk/BAK/CARS/AAAA/EAA
```

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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 startes with a charcter 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

AUTHOR

Peter Miller E-Mail: millerp@canb.auug.org.au
/\/* WWW: http://www.canb.auug.org.au/~millerp/

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 helps 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!

```
D81400000000B000576F77212044696420796F7520726561
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/pfc.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_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	Checksum

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 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	Checksum

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.

Start Address Record

Record Mark (“:”)	Record Length (4)	Load Offset (0)	Record Type (5)	EIP (4 bytes)	Checksum

The Start Address Record is used to specify the execution start address for the object file.

The 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 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)	Checksum (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 1.21

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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.

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	Checksum

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)	Checksum

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)	Checksum

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	Checksum

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.

Start Linear Address Record

(32-bit format only)

Record Mark (“:”)	Record Length (4)	Load Offset (0)	Record Type (5)	EIP (4 bytes)	Checksum

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 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)	Checksum

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)	Checksum (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 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.

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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_mos_tech – MOS Technologies file format

DESCRIPTION

The Mos Technologies 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	

Length The record length field is a 2 character (1 byte) field that specifies the number of data bytes in the record.

Address This is a 2-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 an 2-byte field that represents the least significant two byte of the the sum of the values represented by the pairs of characters making up the record's length, address, and data 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 MOS Technologies format file. It contains the data "Hello, World" to be loaded at address 0.

```
S110000048656C6C6F2C20576F726C640A9D
;00
```

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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’’ which usually manifest as lines which start with something else. These ‘‘extension’’ 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, 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’’ 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
: A 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.

S6
: A 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.

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” to be loaded at address 0.

```
S00600004844521B
S110000048656C6C6F2C20576F726C640A9D
S5030001FB
S9030000FC
```

COPYRIGHT

srec_cat version 1.21

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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 ˆB and ˆC 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" to be loaded at address 0x1000.

```
$A1000,
48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 0A
```

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

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" 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

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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 ":" 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
:B0300D5F746F2072656164207468697333FD1
: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'' 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

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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 preceeded 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’’ 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

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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’’ 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

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

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 (“/”) 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 (“/”) 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 (“/”) 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" to be loaded at address 0.

```
/00000D0D48656C6C6F2C20576F726C640A52
/00000000
```

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

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, the length field, the type field and the checksum.

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 1.21

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.

AUTHOR

Peter Miller E-Mail: millerp@canb.auug.org.au
/\/* WWW: http://www.canb.auug.org.au/~millerp/

NAME

srec_ti_tagged − Texas Instruments Tagged 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.
7	Address.
8	Dummy checksum (ignored).
9	Address.
B	Data word.
F	End of data record.
K	Program identifier.

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.

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.

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 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" to be loaded at address 0x0100.

```
K000590100B4865B6C6CB6F2CB2057B6F72B6C64*0A7F648F
:
```

COPYRIGHT

srec_cat version 1.21

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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.

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” 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

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

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’’ 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’’ 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

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

	The README file	1
	Release Notes	5
	How to build SRecord	10
	How to add a new file format	14
srec_cat(1)	manipulate eprom load files	16
srec_cmp(1)	compare two eprom load files for equality	26
srec_info(1)	information about eprom load files	32
srec_lic(1)	GNU General Public License	38
srec_aomf(5)	Intel Absolute Object Module Format	43
srec_ascii_hex(5)	Ascii-Hex file format	45
srec_atmel_generic(5)	Atmel Generic file format	46
srec_cosmac(5)	RCA Cosmac Elf file format	47
srec_dec_binary(5)	DEC Binary (XXDP) file format	48
srec_emon52(5)	Elektor Monitor (EMON52) file format	49
srec_fairchild(5)	Fairchild Fairbug file format	51
srec_fastload(5)	LSI Logic Fast Load file format	52
srec_formatted_binary(5)	Formatted Binary file format	53
srec_fpc(5)	Four Packed Code (FPC) file format	54
srec_intel16(5)	Intel Hexadecimal 16-bit file format specification	57
srec_intel(5)	Intel Hexadecimal object file format specification	61
srec_mos_tech(5)	MOS Technologies file format	67
srec_motorola(5)	Motorola S-Record hexadecimal file format	68
srec_needham(5)	Needham EMP-series programmer ASCII file format	70
srec_os65v(5)	OS65V Loader file format	71
srec_signetics(5)	Signetics file format	72
srec_spasm(5)	SPASM file format	74
srec_spectrum(5)	Spectrum file format	75
srec_stewie(5)	Stewie's binary file format	76
srec_tektronix(5)	Tektronix hexadecimal file format	78
srec_tektronix_extended(5)	Tektronix Extended hexadecimal file format	80
srec_ti_tagged(5)	Texas Instruments Tagged file format	81
srec_vmem(5)	VMEM file format	83
srec_wilson(5)	wilson file format	85

srec_info(1)	32	srec info - information	about eprom load files
srec_aomf(5)	43	SRecord - Intel	Absolute Object Module Format
srec_needham(5)	70	srec needham - Needham EMP-series programmer	ASCII file format
srec_ascii_hex(5)	45	srec	ascii hex - Ascii-Hex file format
srec_ascii_hex(5)	45	srec ascii hex -	Ascii-Hex file format
srec_atmel_generic(5)	46	srec	atmel generic - Atmel Generic file format
srec_atmel_generic(5)	46	srec atmel generic -	Atmel Generic file format
srec_formatted_binary(5)	53	srec formatted binary - Formatted	Binary file format
srec_stewie(5)	76	srec stewie - Stewie’s	binary file format
srec_formatted_binary(5)	53	srec formatted	binary - Formatted Binary file format
srec_dec_binary(5)	48	SRecord - DEC	Binary (XXDP) file format
srec_intel16(5)	57	srec intel16 - Intel Hexadecimal 16-	bit file format specification
srec_cat(1)	16	srec	cat - manipulate eprom load files
srec_cmp(1)	26	srec	cmp - compare two eprom load files for equality
srec_fpc(5)	54	SRecord - four packed	code file format
srec_cmp(1)	26	srec cmp -	compare two eprom load files for equality
srec_cosmac(5)	47	srec cosmac - RCA	Cosmac Elf file format
srec_cosmac(5)	47	srec	cosmac - RCA Cosmac Elf file format
srec_dec_binary(5)	48	SRecord -	DEC Binary (XXDP) file format
srec_emon52(5)	49	SRecord -	Elektor Monitor (EMON52) file format
srec_cosmac(5)	47	srec cosmac - RCA Cosmac	Elf file format
srec_emon52(5)	49	SRecord - Elektor Monitor (	EMON52) file format
srec_needham(5)	70	srec needham - Needham	EMP-series programmer ASCII file format
srec_cat(1)	16	srec cat - manipulate	eprom load files
srec_info(1)	32	srec info - information about	eprom load files
srec_cmp(1)	26	srec cmp - compare two	eprom load files for equality
srec_cmp(1)	26	srec cmp - compare two eprom load files for	equality
srec_tektronix_extended(5)	80	srec tektronix extended - Tektronix	Extended hexadecimal file format
srec_tektronix_extended(5)	80	srec tektronix	extended - Tektronix Extended hexadecimal file format
srec_fairchild(5)	51	srec fairchild - Fairchild	Fairbug file format
srec_fairchild(5)	51	srec fairchild -	Fairchild Fairbug file format
srec_fairchild(5)	51	srec	fairchild - Fairchild Fairbug file format
srec_fastload(5)	52	srec fastload - LSI Logic	Fast Load file format
srec_fastload(5)	52	srec	fastload - LSI Logic Fast Load file format
srec_ascii_hex(5)	45	srec ascii hex - Ascii-Hex	file format
srec_atmel_generic(5)	46	srec atmel generic - Atmel Generic	file format
srec_cosmac(5)	47	srec cosmac - RCA Cosmac Elf	file format
srec_fairchild(5)	51	srec fairchild - Fairchild Fairbug	file format
srec_fastload(5)	52	srec fastload - LSI Logic Fast Load	file format
srec_formatted_binary(5)	53	srec formatted binary - Formatted Binary	file format
srec_mos_tech(5)	67	srec mos tech - MOS Technologies	file format
srec_motorola(5)	68	srec motorola - Motorola S-Record hexadecimal	file format
srec_needham(5)	70	srec needham - Needham EMP-series programmer ASCII	file format
srec_dec_binary(5)	48	SRecord - DEC Binary (XXDP)	file format
srec_emon52(5)	49	SRecord - Elektor Monitor (EMON52)	file format
srec_fpc(5)	54	SRecord - four packed code	file format
srec_signetics(5)	72	SRecord - Signetics	file format
srec_os65v(5)	71	srec os65v - OS65V Loader	file format

srec_spasm(5)	74	srec spasm - SPASM	file format
srec_spectrum(5)	75	srec spectrum - Spectrum	file format
srec_stewie(5)	76	srec stewie - Stewie’s binary	file format
srec_tektronix_extended(5)	80	srec tektronix extended - Tektronix Extended hexadecimal	file format
srec_tektronix(5)	78	srec tektronix - Tektronix hexadecimal	file format
srec_ti_tagged(5)	81	srec ti tagged - Texas Instruments Tagged	file format
srec_vmem(5)	83	srec vmem - vmem	file format
srec_wilson(5)	85	srec wilson - wilson	file format
srec_intel16(5)	57	srec intel16 - Intel Hexadecimal 16-bit	file format specification
srec_intel(5)	61	srec intel - Intel Hexadecimal object	file format specification
srec_cat(1)	16	srec cat - manipulate eprom load	files
srec_info(1)	32	srec info - information about eprom load	files
srec_cmp(1)	26	srec cmp - compare two eprom load	files for equality
srec_cmp(1)	26	srec cmp - compare two eprom load files	for equality
srec_ascii_hex(5)	45	srec ascii hex - Ascii-Hex file	format
srec_atmel_generic(5)	46	srec atmel generic - Atmel Generic file	format
srec_cosmac(5)	47	srec cosmac - RCA Cosmac Elf file	format
srec_fairchild(5)	51	srec fairchild - Fairchild Fairbug file	format
srec_fastload(5)	52	srec fastload - LSI Logic Fast Load file	format
srec_formatted_binary(5)	53	srec formatted binary - Formatted Binary file	format
srec_mos_tech(5)	67	srec mos tech - MOS Technologies file	format
srec_motorola(5)	68	srec motorola - Motorola S-Record hexadecimal file	format
srec_needham(5)	70	srec needham - Needham EMP-series programmer ASCII file	format
srec_dec_binary(5)	48	SRecord - DEC Binary (XXDP) file	format
srec_emon52(5)	49	SRecord - Elektor Monitor (EMON52) file	format
srec_fpc(5)	54	SRecord - four packed code file	format
srec_aomf(5)	43	SRecord - Intel Absolute Object Module	Format
srec_signetics(5)	72	SRecord - Signetics file	format
srec_os65v(5)	71	srec os65v - OS65V Loader file	format
srec_spasm(5)	74	srec spasm - SPASM file	format
srec_spectrum(5)	75	srec spectrum - Spectrum file	format
srec_stewie(5)	76	srec stewie - Stewie’s binary file	format
srec_tektronix_extended(5)	80	srec tektronix extended - Tektronix Extended hexadecimal file	format
srec_tektronix(5)	78	srec tektronix - Tektronix hexadecimal file	format
srec_ti_tagged(5)	81	srec ti tagged - Texas Instruments Tagged file	format
srec_vmem(5)	83	srec vmem - vmem file	format
srec_wilson(5)	85	srec wilson - wilson file	format
srec_intel16(5)	57	srec intel16 - Intel Hexadecimal 16-bit file	format specification
srec_intel(5)	61	srec intel - Intel Hexadecimal object file	format specification
srec_formatted_binary(5)	53	srec formatted binary -	Formatted Binary file format
srec_formatted_binary(5)	53	srec	formatted binary - Formatted Binary file format
srec_fpc(5)	54	SRecord -	four packed code file format
srec_atmel_generic(5)	46	srec atmel	generic - Atmel Generic file format
srec_atmel_generic(5)	46	srec atmel generic - Atmel	Generic file format
srec_intel16(5)	57	srec intel16 - Intel	Hexadecimal 16-bit file format specification
srec_motorola(5)	68	srec motorola - Motorola S-Record	hexadecimal file format

srec_tektronix_extended(5)	80	srec tektronix extended - Tektronix Extended	hexadecimal file format
srec_tektronix(5)	78	srec tektronix - Tektronix	hexadecimal file format
srec_intel(5)	61	srec intel - Intel	Hexadecimal object file format specification
srec_ascii_hex(5)	45	srec ascii	hex - Ascii-Hex file format
srec_ascii_hex(5)	45	srec ascii hex - Ascii-	Hex file format
srec_info(1)	32	srec	info - information about eprom load files
srec_info(1)	32	srec info -	information about eprom load files
srec_ti_tagged(5)	81	srec ti tagged - Texas	Instruments Tagged file format
srec_intel16(5)	57	srec	intel16 - Intel Hexadecimal 16-bit file format specification
srec_aomf(5)	43	SRecord -	Intel Absolute Object Module Format
srec_intel16(5)	57	srec intel16 -	Intel Hexadecimal 16-bit file format specification
srec_intel(5)	61	srec intel -	Intel Hexadecimal object file format specification
srec_intel(5)	61	srec	intel - Intel Hexadecimal object file format specification
srec_os65v(5)	71	srec os65v - OS65V	Loader file format
srec_fastload(5)	52	srec fastload - LSI Logic Fast	Load file format
srec_cat(1)	16	srec cat - manipulate eprom	load files
srec_info(1)	32	srec info - information about eprom	load files
srec_cmp(1)	26	srec cmp - compare two eprom	load files for equality
srec_fastload(5)	52	srec fastload - LSI	Logic Fast Load file format
srec_fastload(5)	52	srec fastload -	LSI Logic Fast Load file format
srec_cat(1)	16	srec cat -	manipulate eprom load files
srec_aomf(5)	43	SRecord - Intel Absolute Object	Module Format
srec_emon52(5)	49	SRecord - Elektor	Monitor (EMON52) file format
srec_mos_tech(5)	67	srec	mos tech - MOS Technologies file format
srec_mos_tech(5)	67	srec mos tech -	MOS Technologies file format
srec_motorola(5)	68	srec	motorola - Motorola S-Record hexadecimal file format
srec_motorola(5)	68	srec motorola -	Motorola S-Record hexadecimal file format
srec_needham(5)	70	srec needham -	Needham EMP-series programmer ASCII file format
srec_needham(5)	70	srec	needham - Needham EMP-series programmer ASCII file format
srec_intel(5)	61	srec intel - Intel Hexadecimal	object file format specification
srec_aomf(5)	43	SRecord - Intel Absolute	Object Module Format
srec_os65v(5)	71	srec os65v -	OS65V Loader file format
srec_os65v(5)	71	srec	os65v - OS65V Loader file format
srec_fpc(5)	54	SRecord - four	packed code file format
srec_needham(5)	70	srec needham - Needham EMP-series	programmer ASCII file format
srec_cosmac(5)	47	srec cosmac -	RCA Cosmac Elf file format
srec_motorola(5)	68	srec motorola - Motorola S-	Record hexadecimal file format
srec_stewie(5)	76	srec stewie - Stewie’	s binary file format
srec_needham(5)	70	srec needham - Needham EMP-	series programmer ASCII file format
srec_signetics(5)	72	SRecord -	Signetics file format
srec_spasm(5)	74	srec spasm -	SPASM file format
srec_spasm(5)	74	srec	spasm - SPASM file format
srec_intel16(5)	57	srec intel16 - Intel Hexadecimal 16-bit file format	specification

srec_intel(5)	61	srec intel - Intel Hexadecimal object file format	specification
srec_spectrum(5)	75	srec spectrum -	Spectrum file format
srec_spectrum(5)	75	srec	spectrum - Spectrum file format
srec_ascii_hex(5)	45		srec ascii hex - Ascii-Hex file format
srec_atmel_generic(5)	46		srec atmel generic - Atmel Generic file format
srec_cat(1)	16		srec cat - manipulate eprom load files
srec_cmp(1)	26		srec cmp - compare two eprom load files for equality
srec_cosmac(5)	47		srec cosmac - RCA Cosmac Elf file format
srec_fairchild(5)	51		srec fairchild - Fairchild Fairbug file format
srec_fastload(5)	52		srec fastload - LSI Logic Fast Load file format
srec_formatted_binary(5)	53		srec formatted binary - Formatted Binary file format
srec_info(1)	32		srec info - information about eprom load files
srec_intel16(5)	57		srec intel16 - Intel Hexadecimal 16-bit file format specification
srec_intel(5)	61		srec intel - Intel Hexadecimal object file format specification
srec_mos_tech(5)	67		srec mos tech - MOS Technologies file format
srec_motorola(5)	68		srec motorola - Motorola S-Record hexadecimal file format
srec_needham(5)	70		srec needham - Needham EMP-series programmer ASCII file format
srec_dec_binary(5)	48		SRecord - DEC Binary (XXDP) file format
srec_emon52(5)	49		SRecord - Elektor Monitor (EMON52) file format
srec_fpc(5)	54		SRecord - four packed code file format
srec_motorola(5)	68	srec motorola - Motorola	S-Record hexadecimal file format
srec_aomf(5)	43		SRecord - Intel Absolute Object Module Format
srec_signetics(5)	72		SRecord - Signetics file format
srec_os65v(5)	71		srec os65v - OS65V Loader file format
srec_spasm(5)	74		srec spasm - SPASM file format
srec_spectrum(5)	75		srec spectrum - Spectrum file format
srec_stewie(5)	76		srec stewie - Stewie's binary file format
srec_tektronix_extended(5)	80		srec tektronix extended - Tektronix Extended hexadecimal file format
srec_tektronix(5)	78		srec tektronix - Tektronix hexadecimal file format
srec_ti_tagged(5)	81		srec ti tagged - Texas Instruments Tagged file format
srec_vmem(5)	83		srec vmem - vmem file format
srec_wilson(5)	85		srec wilson - wilson file format
srec_stewie(5)	76	srec stewie -	Stewie's binary file format
srec_stewie(5)	76	srec	stewie - Stewie's binary file format
srec_ti_tagged(5)	81	srec ti tagged - Texas Instruments	Tagged file format
srec_ti_tagged(5)	81	srec ti	tagged - Texas Instruments Tagged file format

srec_mos_tech(5)	67	srec mos	tech - MOS Technologies file format
srec_mos_tech(5)	67	srec mos tech - MOS	Technologies file format
srec_tektronix_extended(5)	80	srec tektronix extended -	Tektronix Extended hexadecimal file format
srec_tektronix_extended(5)	80	srec	tektronix extended - Tektronix Extended hexadecimal file format
srec_tektronix(5)	78	srec tektronix -	Tektronix hexadecimal file format
srec_tektronix(5)	78	srec	tektronix - Tektronix hexadecimal file format
srec_ti_tagged(5)	81	srec ti tagged -	Texas Instruments Tagged file format
srec_ti_tagged(5)	81	srec	ti tagged - Texas Instruments Tagged file format
srec_cmp(1)	26	srec cmp - compare	two eprom load files for equality
srec_os65v(5)	71	srec os65v - OS65	V Loader file format
srec_vmem(5)	83	srec vmem -	vmem file format
srec_vmem(5)	83	srec	vmem - vmem file format
srec_os65v(5)	71	srec os65	v - OS65V Loader file format
srec_wilson(5)	85	srec wilson -	wilson file format
srec_wilson(5)	85	srec	wilson - wilson file format
srec_dec_binary(5)	48	SRecord - DEC Binary (	XXDP) file format