SRecord

Reference Manual

Peter Miller
millerp@canb.auug.org.au

.

This document describes SRecord version 1.8
and was prepared 30 April 2001.

This document describing the SRecord program, and the SRecord program itself, are
Copyright © 1998, 1999, 2000, 2001 Peter Miller; All rights reserved.

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.

NAME

RSecord − 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.)

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 another program. This format cannot be read.

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

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

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

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 *nput 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.

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

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

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

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

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

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

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

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://www.canb.auug.org.au/˜millerp/srecord/	
File:	srecord.html	# the SRecord page
File:	srecord-1.8.README	# Description, from the tar file
File:	srecord-1.8.lsm	# Description, LSM format
File:	srecord-1.8.spec	# RedHat package specification
File:	srecord-1.8.tar.gz	# the complete source
File:	srecord-1.8.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-1.8.README	# Description, from the tar file
File:	srecord-1.8.lsm	# Description, LSM format
File:	srecord-1.8.spec	# RedHat package specification
File:	srecord-1.8.tar.gz	# the complete source
File:	srecord-1.8.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.org) environment. Instructions are in the *BUILDING* file, including how to get native Windows binaries.

COPYRIGHT

srecord version 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller; All rights reserved.

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.

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

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 1.8

• 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

• 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

• 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.

• 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.

• 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.

Version 1.5

• 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.

• The ‘‘Tektronix Extended’’ file format is now understood for reading and writing.

• The ‘‘Texas Instruments Tagged’’ file format is now understood for reading and writing. (Also known as the TI-Tagged or SDSMAC format.)

• The ‘‘ascii-hex’’ file format is now understood for reading and writing. (Also known as the ascii-space-hex format.)

• There is a new *byte swap* input filter, allowing pairs of odd and even input bytes to be swapped.

• The ‘‘wilson’’ file format is now understood for reading and writing. This mystery format was added for a mysterious type of EPROM writer.

• 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.

• There is a new *-Line-Length* option for the *srec_cat* program, allowing you to specify the maximum width of output lines.

Version 1.4

• 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

• 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.

Version 1.2

• 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.

• The Tektronix hex format is now supported (only the 16-bit version, Extended tektronix hex is not yet suppported).

• There is a new *split* filter, useful for wide data buses and memory striping, and a complementary *unsplit* filter to reverse it.

Version 1.1

First public release.

NAME

SRecord − manipulate EPROM load files

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 system (see `www.cygwin.org`, it's free). This provides the ‘‘porting layer’’ necessary to run Unix programs on Windows. The build process is exactly as described above.

If you want to make ‘‘native binaries’’ (*i.e.* ones which work outside Cygwin) there is one extra step you need after running edit the *Makefile* file, and add `-mno-cygwin` to the end of the `CXX=g++` line.

Once built, Windows (Cygwin) binaries should test in the same way as described in the next section.

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 <millerp@canb.auug.org.au>
```

Any and all feedback is welcome.

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

```
% srec_cat -version
srecord version 1.8.D001
...warranty disclaimer...
%
```

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

COPYRIGHT

srecord version 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller; All rights reserved.

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

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

AUTHOR

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:

−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.

−Binary
This option says the file is a raw binary file, and should be read literally. (May also be written −Raw.)

−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.

−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.

−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.

−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).

−Big_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).

−Big_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).

−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 then 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 crop region 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 crop region extends from the minimum to the maximum address used by the input, and fills 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.

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:

−Ascii_Hex

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

−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.

−WILson

A wilson format file will be written. (See *srec_wilson*(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.)

−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.

−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.

−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.

-Line_Length *number*

This option may be used to limit the length of the output lines to at most *numberP characters. (Not meaningful for binary file format.) Defaults to something less than 80 characters, depending on the format.*

−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.

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

srec_cat version 1.8

Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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_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:

−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.

−Binary
This option says the file is a raw binary file, and should be read literally. (May also be written −Raw.)

−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.

−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.

−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.

−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).

−Big_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).

−Big_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).

−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 then 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 crop region 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 crop region extends from the minimum to the maximum address used by the input, and fills 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.

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 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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:

−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.

−Binary
This option says the file is a raw binary file, and should be read literally. (May also be written −Raw.)

−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.

−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.

−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.

−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).

−Big_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).

−Big_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).

−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 then 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 crop region 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 crop region extends from the minimum to the maximum address used by the input, and fills 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.

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

srec_info version 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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_ascii_hex − Ascii-Hex file format

DESCRIPTION

This format is also known as the *Ascii-Space-Hex* 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 startes with a Crontrol-B character (0x02).

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 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 a Control-C character (0x03).

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 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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_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

srec_cat version 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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/

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

srec_cat version 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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_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 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 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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_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 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

srec_cat version 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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_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 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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_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

srec_cat version 1.8
Copyright © 1998, 1999, 2000, 2001 Peter Miller;
All rights reserved.

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/