WIKIPEDIA

Executable and Linkable Format

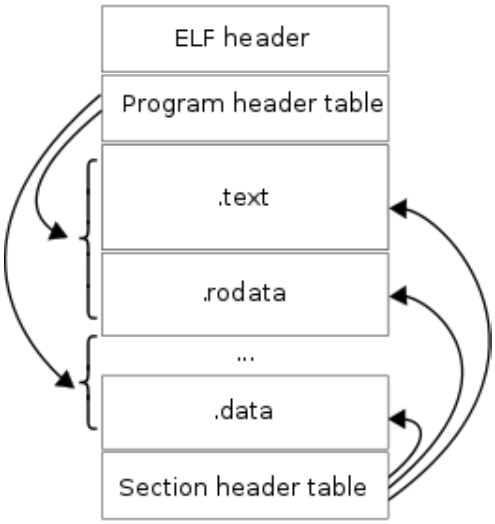
In computing, the **Executable and Linkable Format** (**ELF**, formerly named **Extensible Linking Format**), is a common standard file format for executable files, object code, shared libraries, and core dumps. First published in the specification for the application binary interface (ABI) of the Unix operating system version named System V Release 4 (SVR4),[2] and later in the Tool Interface Standard,[1] it was quickly accepted among different vendors of Unix systems. In 1999, it was chosen as the standard binary file format for Unix and Unix-like systems on x86 processors by the 86open project.

By design, the ELF format is flexible, extensible, and cross-platform. For instance it supports different endiannesses and address sizes so it does not exclude any particular central processing unit (CPU) or instruction set architecture. This has allowed it to be adopted by many different operating systems on many different hardware platforms.

ELF

Filename extension	`none`, `.axf`, `.bin`, `.elf`, `.o`, `.prx`, `.puff`, `.ko`, `.mod` and `.so`
Magic number	`0x7F 'E' 'L' 'F'`
Developed by	Unix System Laboratories[1]:3
Type of format	Binary, executable, object, shared library, core dump
Container for	Many executable binary formats

Contents



An ELF file has two views: the program header shows the *segments* used at run time, whereas the section header lists the set of *sections* of the binary.

File layout

Each ELF file is made up of one ELF header, followed by file data. The data can include:

- Program header table, describing zero or more memory segments
- Section header table, describing zero or more sections
- Data referred to by entries in the program header table or section header table

The segments contain information that is needed for run time execution of the file, while sections contain important data for linking and relocation. Any byte in the entire file can be owned by one section at most, and orphan bytes can occur which are unowned by any section.

File

```
00000000 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00 |.ELF............|
00000010 02 00 3e 00 01 00 00 00 c5 48 40 00 00 00 00 00 |..>......H@.....|
```

Example hexdump of ELF file header[3]

header

The ELF header defines whether to use 32- or 64-bit addresses. The header contains three fields that are affected by this setting and offset other fields that follow them. The ELF header is 52 or 64 bytes long for 32-bit and 64-bit binaries respectively.

ELF header[4]

Offset		Size (bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x00		4		e_ident[EI_MAG0] through e_ident[EI_MAG3]	0x7F followed by ELF(45 4c 46) in ASCII; these four bytes constitute the magic number.
0x04		1		e_ident[EI_CLASS]	This byte is set to either 1 or 2 to signify 32- or 64-bit format, respectively.
0x05		1		e_ident[EI_DATA]	This byte is set to either 1 or 2 to signify little or big endianness, respectively. This affects interpretation of multi-byte fields starting with offset 0x10.
0x06		1		e_ident[EI_VERSION]	Set to 1 for the original and current version of ELF.
0x07		1		e_ident[EI_OSABI]	Identifies the target operating system ABI.

<table>
<tr><td></td><td></td><td></td><td><table>
<tr><th>Value</th><th>ABI</th></tr>
<tr><td>0x00</td><td>System V</td></tr>
<tr><td>0x01</td><td>HP-UX</td></tr>
<tr><td>0x02</td><td>NetBSD</td></tr>
<tr><td>0x03</td><td>Linux</td></tr>
<tr><td>0x04</td><td>GNU Hurd</td></tr>
<tr><td>0x06</td><td>Solaris</td></tr>
<tr><td>0x07</td><td>AIX</td></tr>
<tr><td>0x08</td><td>IRIX</td></tr>
<tr><td>0x09</td><td>FreeBSD</td></tr>
<tr><td>0x0A</td><td>Tru64</td></tr>
<tr><td>0x0B</td><td>Novell Modesto</td></tr>
<tr><td>0x0C</td><td>OpenBSD</td></tr>
<tr><td>0x0D</td><td>OpenVMS</td></tr>
<tr><td>0x0E</td><td>NonStop Kernel</td></tr>
<tr><td>0x0F</td><td>AROS</td></tr>
<tr><td>0x10</td><td>Fenix OS</td></tr>
<tr><td>0x11</td><td>CloudABI</td></tr>
</table>
It is often set to <code>0</code> regardless of the target platform.</td></tr>
<tr><td>0x08</td><td>1</td><td><code>e_ident[EI_ABIVERSION]</code></td><td>Further specifies the ABI version. Its interpretation depends on the target ABI. Linux kernel (after at least 2.6) has no definition of it.[5] In that case, offset and size of EI_PAD are 8.</td></tr>
<tr><td>0x09</td><td>7</td><td><code>e_ident[EI_PAD]</code></td><td>currently unused</td></tr>
<tr><td>0x10</td><td>2</td><td><code>e_type</code></td><td>Identifies object file type.</td></tr>
</table>

				Value / **Type** 0x00 ET_NONE 0x01 ET_REL 0x02 ET_EXEC 0x03 ET_DYN 0x04 ET_CORE 0xfe00 ET_LOOS 0xfeff ET_HIOS 0xff00 ET_LOPROC 0xffff ET_HIPROC
0x12	2		e_machine	Specifies target instruction set architecture. Some examples are: **Value** / **ISA** 0x00 No specific instruction set 0x02 SPARC 0x03 x86 0x08 MIPS 0x14 PowerPC 0x16 S390 0x28 ARM 0x2A SuperH 0x32 IA-64 0x3E x86-64 0xB7 AArch64 0xF3 RISC-V
0x14	4		e_version	Set to 1 for the original version of ELF.
0x18	4	8	e_entry	This is the memory address of the entry point from where the process starts executing. This field is either 32 or 64 bits long depending on the format defined earlier.

0x1C	0x20	4	8	e_phoff	Points to the start of the program header table. It usually follows the file header immediately, making the offset 0x34 or 0x40 for 32- and 64-bit ELF executables, respectively.
0x20	0x28	4	8	e_shoff	Points to the start of the section header table.
0x24	0x30	4		e_flags	Interpretation of this field depends on the target architecture.
0x28	0x34	2		e_ehsize	Contains the size of this header, normally 64 Bytes for 64-bit and 52 Bytes for 32-bit format.
0x2A	0x36	2		e_phentsize	Contains the size of a program header table entry.
0x2C	0x38	2		e_phnum	Contains the number of entries in the program header table.
0x2E	0x3A	2		e_shentsize	Contains the size of a section header table entry.
0x30	0x3C	2		e_shnum	Contains the number of entries in the section header table.
0x32	0x3E	2		e_shstrndx	Contains index of the section header table entry that contains the section names.

Program header

The program header table tells the system how to create a process image. It is found at file offset `e_phoff`, and consists of `e_phnum` entries, each with size `e_phentsize`. The layout is slightly different in 32-bit ELF vs 64-bit ELF, because the `p_flags` are in a different structure location for alignment reasons. Each entry is structured as:

Program header[6]

<table>
<tr><th colspan="2">Offset</th><th colspan="2">Size (bytes)</th><th rowspan="2">Field</th><th rowspan="2">Purpose</th></tr>
<tr><th>32-bit</th><th>64-bit</th><th>32-bit</th><th>64-bit</th></tr>
<tr><td colspan="2">0x00</td><td colspan="2">4</td><td>p_type</td><td>Identifies the type of the segment.
<table><tr><th>Value</th><th>Name</th><th>Meaning</th></tr><tr><td>0x00000000</td><td>PT_NULL</td><td>Program header table entry unused</td></tr><tr><td>0x00000001</td><td>PT_LOAD</td><td>Loadable segment</td></tr><tr><td>0x00000002</td><td>PT_DYNAMIC</td><td>Dynamic linking information</td></tr><tr><td>0x00000003</td><td>PT_INTERP</td><td>Interpreter information</td></tr><tr><td>0x00000004</td><td>PT_NOTE</td><td>Auxiliary information</td></tr><tr><td>0x00000005</td><td>PT_SHLIB</td><td>reserved</td></tr><tr><td>0x00000006</td><td>PT_PHDR</td><td>segment containing program header table itself</td></tr><tr><td>0x60000000</td><td>PT_LOOS</td><td rowspan="4">see below</td></tr><tr><td>0x6FFFFFFF</td><td>PT_HIOS</td></tr><tr><td>0x70000000</td><td>PT_LOPROC</td></tr><tr><td>0x7FFFFFFF</td><td>PT_HIPROC</td></tr></table>PT_LOOS to PT_HIOS (PT_LOPROC to PT_HIPROC) is an inclusive reserved ranges for operating system (processor) specific semantics.</td></tr>
<tr><td></td><td>0x04</td><td></td><td>4</td><td>p_flags</td><td>Segment-dependent flags (position for 64-bit structure).</td></tr>
<tr><td>0x04</td><td>0x08</td><td>4</td><td>8</td><td>p_offset</td><td>Offset of the segment in the file image.</td></tr>
<tr><td>0x08</td><td>0x10</td><td>4</td><td>8</td><td>p_vaddr</td><td>Virtual address of the segment in memory.</td></tr>
<tr><td>0x0C</td><td>0x18</td><td>4</td><td>8</td><td>p_paddr</td><td>On systems where physical address is relevant, reserved for segment's physical address.</td></tr>
<tr><td>0x10</td><td>0x20</td><td>4</td><td>8</td><td>p_filesz</td><td>Size in bytes of the segment in the file image. May be 0.</td></tr>
<tr><td>0x14</td><td>0x28</td><td>4</td><td>8</td><td>p_memsz</td><td>Size in bytes of the segment in memory. May be 0.</td></tr>
<tr><td>0x18</td><td></td><td>4</td><td></td><td>p_flags</td><td>Segment-dependent flags (position for 32-bit structure).</td></tr>
<tr><td>0x1C</td><td>0x30</td><td>4</td><td>8</td><td>p_align</td><td>0 and 1 specify no alignment. Otherwise should be a positive, integral power of 2, with p_vaddr equating</td></tr>
</table>

					`p_offset` modulus `p_align`.
0x20	0x38				End of Program Header (size)

Section header

Offset		Size (bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x00		4		sh_name	An offset to a string in the **.shstrtab** section that represents the name of this section
0x04		4		sh_type	Identifies the type of this header.

Value	Name	Meaning
0x0	SHT_NULL	Section header table entry unused
0x1	SHT_PROGBITS	Program data
0x2	SHT_SYMTAB	Symbol table
0x3	SHT_STRTAB	String table
0x4	SHT_RELA	Relocation entries with addends
0x5	SHT_HASH	Symbol hash table
0x6	SHT_DYNAMIC	Dynamic linking information
0x7	SHT_NOTE	Notes
0x8	SHT_NOBITS	Program space with no data (bss)
0x9	SHT_REL	Relocation entries, no addends
0x0A	SHT_SHLIB	Reserved
0x0B	SHT_DYNSYM	Dynamic linker symbol table
0x0E	SHT_INIT_ARRAY	Array of constructors
0x0F	SHT_FINI_ARRAY	Array of destructors
0x10	SHT_PREINIT_ARRAY	Array of pre-constructors
0x11	SHT_GROUP	Section group
0x12	SHT_SYMTAB_SHNDX	Extended section indices
0x13	SHT_NUM	Number of defined types.
0x60000000	SHT_LOOS	Start OS-specific.
...	...	...

				Identifies the attributes of the section.
0x08	4	8	sh_flags	

Value	Name	Meaning
0x1	SHF_WRITE	Writable
0x2	SHF_ALLOC	Occupies memory during execution
0x4	SHF_EXECINSTR	Executable
0x10	SHF_MERGE	Might be merged
0x20	SHF_STRINGS	Contains nul-terminated strings
0x40	SHF_INFO_LINK	'sh_info' contains SHT index
0x80	SHF_LINK_ORDER	Preserve order after combining
0x100	SHF_OS_NONCONFORMING	Non-standard OS specific handling required
0x200	SHF_GROUP	Section is member of a group
0x400	SHF_TLS	Section hold thread-local data
0x0ff00000	SHF_MASKOS	OS-specific
0xf0000000	SHF_MASKPROC	Processor-specific
0x4000000	SHF_ORDERED	Special ordering requirement (Solaris)
0x8000000	SHF_EXCLUDE	Section is excluded unless referenced or allocated (Solaris)

Tools

- `readelf` is a Unix binary utility that displays information about one or more ELF files. A free software implementation is provided by GNU Binutils.
- `elfutils` provides alternative tools to GNU Binutils purely for Linux.[7]
- `elfdump` is a command for viewing ELF information in an ELF file, available under Solaris and FreeBSD.
- `objdump` provides a wide range of information about ELF files and other object formats. `objdump` uses the Binary File Descriptor library as a back-end to structure the ELF data.
- The Unix `file` utility can display some information about ELF files, including the instruction set architecture for which the code in a relocatable, executable, or shared object file is intended, or on which an ELF core dump was produced.

Applications

Unix-like systems

The ELF format has replaced older executable formats in various environments. It has replaced a.out and COFF formats in Unix-like operating systems:

- Linux
- Solaris / Illumos
- IRIX
- FreeBSD[8]
- NetBSD
- OpenBSD
- Redox
- DragonFly BSD
- Syllable
- HP-UX (except for 32-bit PA-RISC programs which continue to use SOM)
- QNX Neutrino
- MINIX[9]

Non-Unix adoption

ELF has also seen some adoption in non-Unix operating systems, such as:

- OpenVMS, in its Itanium and x86-64 versions[10]
- BeOS Revision 4 and later for x86 based computers (where it replaced the Portable Executable format; the PowerPC version stayed with Preferred Executable Format)
- Haiku, an open source reimplementation of BeOS
- RISC OS[11]
- Stratus VOS, in PA-RISC and x86 versions
- Windows 10 Anniversary Update using the Windows Subsystem for Linux.[12][13]
- SkyOS

- Fuchsia OS
- Z/TPF
- HPE NonStop OS[14]

Game consoles

Some game consoles also use ELF:

- PlayStation Portable,[15] PlayStation Vita, PlayStation 2, PlayStation 3, PlayStation 4
- GP2X
- Dreamcast
- Wii

PowerPC

Other (operating) systems running on PowerPC that use ELF:

- AmigaOS 4, the ELF executable has replaced the prior Extended Hunk Format (EHF) which was used on Amigas equipped with PPC processor expansion cards.
- MorphOS
- AROS

Mobile phones

Some operating systems for mobile phones and mobile devices use ELF:

- Symbian OS v9 uses E32Image[16] format that is based on the ELF file format;
- Sony Ericsson, for example, the W800i, W610, W300, etc.
- Siemens, the SGOLD and SGOLD2 platforms: from Siemens C65 to S75 and BenQ-Siemens E71/EL71;
- Motorola, for example, the E398, SLVR L7, v360, v3i (and all phone LTE2 which has the patch applied).
- Bada, for example, the Samsung Wave S8500.
- Nokia phones or tablets running the Maemo or the Meego OS, for example, the Nokia N900.
- Android uses ELF `.so` (shared object[17]) libraries for the Java Native Interface. With Android Runtime (ART), the default since Android 5.0 "Lollipop", all applications are compiled into native ELF binaries on installation.

Some phones can run ELF files through the use of a patch that adds assembly code to the main firmware, which is a feature known as *ELFPack* in the underground modding culture. The ELF file format is also used with the Atmel AVR (8-bit), AVR32[18] and with Texas Instruments MSP430 microcontroller architectures. Some implementations of Open Firmware can also load ELF files, most notably Apple's implementation used in almost all PowerPC machines the company produced.

Specifications

- Generic:

- *System V Application Binary Interface (http://www.sco.com/developers/devspecs/gabi41.pdf)* Edition 4.1 (1997-03-18)
- *System V ABI Update (http://www.sco.com/developers/gabi/latest/contents.html)* (October 2009)

- AMD64:
  - *System V ABI, AMD64 Supplement (http://refspecs.linuxbase.org/elf/x86_64-abi-0.99.pdf)*
- ARM:
  - *ELF for the ARM Architecture (http://infocenter.arm.com/help/topic/com.arm.doc.ihi0044b/IHI0044B_aaelf.pdf)*
- IA-32:
  - *System V ABI, Intel386 Architecture Processor Supplement (http://www.sco.com/developers/devspecs/abi386-4.pdf)*
- IA-64:
  - *Itanium Software Conventions and Runtime Guide (http://refspecs.linux-foundation.org/IA64conventions.pdf)* (September 2000)
- M32R:
  - *M32R ELF ABI Supplement (http://www.linux-m32r.org/cmn/m32r/M32R-elf-abi.pdf)* Version 1.2 (2004-08-26)
- MIPS:
  - *System V ABI, MIPS RISC Processor Supplement (http://www.sco.com/developers/devspecs/mipsabi.pdf)*
  - *MIPS EABI documentation (http://sources.redhat.com/ml/binutils/2003-06/msg00436.html)* (2003-06-11)
- Motorola 6800:
  - *Motorola 8- and 16- bit Embedded ABI (http://uclibc.org/docs/psABI-m8-16.pdf)*
- PA-RISC:
  - *ELF Supplement for PA-RISC (https://web.archive.org/web/20110317045038/http://refspecs.freestandards.org/elf/elf-pa.pdf)* Version 1.43 (October 6, 1997)
- PowerPC:
  - *System V ABI, PPC Supplement (https://web.archive.org/web/20070630123210/http://refspecs.freestandards.org/elf/elfspec_ppc.pdf)*
  - *PowerPC Embedded Application Binary Interface (https://web.archive.org/web/20110723003758/http://sources-redhat.mirrors.airband.net/binutils/ppc-docs/ppc-eabi-1995-01.pdf) 32-Bit Implementation* (1995-10-01)
  - *64-bit PowerPC ELF Application Binary Interface Supplement (http://refspecs.linuxfoundation.org/ELF/ppc64/PPC-elf64abi-1.9.html)* Version 1.9 (2004)
- SPARC:
  - *System V ABI, SPARC Supplement (https://web.archive.org/web/20080517110249/http://www.sparc.org/standards/psABI3rd.pdf)*
- S/390:
  - *S/390 32bit ELF ABI Supplement (http://refspecs.linuxbase.org/ELF/zSeries/lzsabi0_s390.html)*
- zSeries:
  - *zSeries 64bit ELF ABI Supplement (http://refspecs.linuxbase.org/ELF/zSeries/lzsabi0_zSeries.html)*
- Symbian OS 9:
  - *E32Image file format on Symbian OS 9 (https://web.archive.org/web/20080518002831/http://wiki.forum.nokia.com/index.php/E32Image_file_format_on_Symbian_OS_9)*

The Linux Standard Base (LSB) supplements some of the above specifications for architectures in which it is specified.[19] For example, that is the case for the System V ABI, AMD64 Supplement.[20][21]

86open

86open was a project to form consensus on a common binary file format for Unix and Unix-like operating systems on the common PC compatible x86 architecture, to encourage software developers to port to the architecture.[22] The initial idea was to standardize on a small subset of Spec 1170, a predecessor of the Single UNIX Specification, and the GNU C Library (glibc) to enable unmodified binaries to run on the x86 Unix-like operating systems. The project was originally designated "Spec 150".

The format eventually chosen was ELF, specifically the Linux implementation of ELF, after it had turned out to be a *de facto* standard supported by all involved vendors and operating systems.

The group began email discussions in 1997 and first met together at the Santa Cruz Operation offices on August 22, 1997.

The steering committee was Marc Ewing, Dion Johnson, Evan Leibovitch, Bruce Perens, Andrew Roach, Bryan Wayne Sparks and Linus Torvalds. Other people on the project were Keith Bostic, Chuck Cranor, Michael Davidson, Chris G. Demetriou, Ulrich Drepper, Don Dugger, Steve Ginzburg, Jon "maddog" Hall, Ron Holt, Jordan Hubbard, Dave Jensen, Kean Johnston, Andrew Josey, Robert Lipe, Bela Lubkin, Tim Marsland, Greg Page, Ronald Joe Record, Tim Ruckle, Joel Silverstein, Chia-pi Tien, and Erik Troan. Operating systems and companies represented were BeOS, BSDI, FreeBSD, Intel, Linux, NetBSD, SCO and SunSoft.

The project progressed and in mid-1998, SCO began developing lxrun, an open-source compatibility layer able to run Linux binaries on OpenServer, UnixWare, and Solaris. SCO announced official support of lxrun at LinuxWorld in March 1999. Sun Microsystems began officially supporting lxrun for Solaris in early 1999,[23] and later moved to integrated support of the Linux binary format via Solaris Containers for Linux Applications.

With the BSDs having long supported Linux binaries (through a compatibility layer) and the main x86 Unix vendors having added support for the format, the project decided that Linux ELF was the format chosen by the industry and "declare[d] itself dissolved" on July 25, 1999.[24]

FatELF: universal binaries for Linux

FatELF is an ELF binary-format extension that adds fat binary capabilities.[25] It is aimed for Linux and other Unix-like operating systems. Additionally to the CPU architecture abstraction (byte order, word size, CPU instruction set etc.), there is the potential advantage of software-platform abstraction e.g., binaries which support multiple kernel ABI versions. As of 2014, support for FatELF is not integrated in the Linux kernel mainline.[26][27][28]

See also

- Application binary interface
- Comparison of executable file formats
- DWARF – a format for debugging data
- Intel Binary Compatibility Standard
- Portable Executable – format used by Windows
- vDSO – virtual DSO
- Position-independent code

References

1. Tool Interface Standard (TIS) *Executable and Linking Format (ELF) Specification (http://refspecs.linuxbase.org/elf/elf.pdf) Version 1.2* (May 1995)
2. *System V Application Binary Interface (http://www.sco.com/developers/devspecs/gabi41.pdf)* Edition 4.1 (1997-03-18)
3. "Available lexers — Pygments" (http://pygments.org/docs/lexers/#lexers-for-hexadecimal-dumps). *pygments.org*.
4. "ELF Header" (http://www.sco.com/developers/gabi/2000-07-17/ch4.eheader.html). Sco.com. July 2000. Retrieved 2014-02-07.
5. "LXR linux/include/linux/elf.h" (http://lxr.linux.no/linux+v2.6.11/include/linux/elf.h#L380). *linux.no*. Retrieved 27 April 2015.
6. "Program Header" (http://www.sco.com/developers/gabi/2000-07-17/ch5.pheader.html). Sco.com. July 2000. Retrieved 2017-04-05.
7. "elfutils" (https://sourceware.org/elfutils/). *sourceware.org*. Retrieved 30 April 2017.
8. https://docs.freebsd.org/doc/4.9-RELEASE/usr/share/doc/handbook/binary-formats.html
9. "MinixReleases – Minix Wiki" (https://www.webcitation.org/6DkdM3jWL?url=http://wiki.minix3.org/en/MinixReleases). Wiki.minix3.org. Archived from the original (http://wiki.minix3.org/en/MinixReleases) on 2013-01-18. Retrieved 2014-01-19.
10. https://vmssoftware.com/pdfs/State_of_Port_20160906.pdf
11. "GCCSDK – RISC OS" (http://www.riscos.info/index.php/GCCSDK). Riscos.info. 2012-04-22. Retrieved 2014-01-19.
12. "Announcing Windows 10 Insider Preview Build 14316" (https://blogs.windows.com/windowsexperience/2016/04/06/announcing-windows-10-insider-preview-build-14316/). *Windows Experience Blog*. Retrieved 2016-04-10.
13. Foley, Mary Jo. "Under the hood of Microsoft's Windows Subsystem for Linux | ZDNet" (http://www.zdnet.com/article/under-the-hood-of-microsofts-windows-subsystem-for-linux/). *ZDNet*. Retrieved 2016-08-19.
14. "Guardian Programmer's Guide" (https://web.archive.org/web/20180530042640/http://h20628.www2.hp.com/km-ext/kmcsdirect/emr_na-c02543407-12.pdf) (PDF). Hewlett Packard Enterprise. Archived from the original (http://h20628.www2.hp.com/km-ext/kmcsdirect/emr_na-c02543407-12.pdf) (PDF) on 2018-05-30. Retrieved 2018-05-30. p. 44 archived from the original (http://h20628.www2.hp.com/km-ext/kmcsdirect/emr_na-c02543407-12.pdf) on 2018-5-30
15. PlayStation Portable use encrypted & relocated ELF : PSP
16. *Symbian OS executable file format (https://web.archive.org/web/20091213034509/http://wiki.forum.nokia.com/index.php/E32Image)*
17. Rosen, Kenneth; Host, Douglas; Klee, Rachel; Rosinski, Richard (2007). *UNIX: The Complete Reference* (https://books.google.com/books?id=2Et--84HIkwC) (2 ed.). McGraw Hill Professional. p. 707. ISBN 9780071706988. Retrieved 2017-06-08. "Dynamically linked libraries are also called shared objects (.so)."
18. "Chapter 4: Object Files", *System V Application Binary Interface* (https://www.sco.com/developers/gabi/2009-10-26/ch4.eheader.html#e_machine), 2009-10-26, *e_machine*
19. "LSB Referenced Specifications" (http://refspecs.linuxfoundation.org/lsb.shtml). *linuxfoundation.org*. Retrieved 27 April 2015.
20. "Executable and Linking Format (ELF)" (http://refspecs.linuxfoundation.org/LSB_4.1.0/LSB-Core-AMD64/LSB-Core-AMD64/elf-amd64.html). *linuxfoundation.org*. Retrieved 27 April 2015.

21. "Introduction" (http://refspecs.linuxfoundation.org/LSB_4.1.0/LSB-Core-AMD64/LSB-Core-AMD64/elfintro.html). *linuxfoundation.org*. Retrieved 27 April 2015.
22. Leibovitch, Evan (1997-12-23). "86Open Frequently-Asked Questions" (https://web.archive.org/web/20070311032337/http://www.telly.org/86open-faq). Archived from the original (http://www.telly.org/86open-faq) on 2007-03-11. Retrieved 2007-06-06.
23. Record, Ronald (1998-05-21). "Bulletin on status of 86open at SCO" (https://web.archive.org/web/20081208013909/http://www.mavetju.org/mail/view_message.php?list=freebsd-emulation&id=361608). Archived from the original (http://www.mavetju.org/mail/view_message.php?list=freebsd-emulation&id=361608) on 2008-12-08. Retrieved 2008-05-06.
24. Leibovitch, Evan (1999-07-25). "The86open Project – Final Update" (https://web.archive.org/web/20070227214032/http://www.telly.org/86open/). Archived from the original (http://www.telly.org/86open/) on 2007-02-27. Retrieved 2007-05-06.
25. Gordon, Ryan. "fatelf-specification v1" (http://hg.icculus.org/icculus/fatelf/raw-file/tip/docs/fatelf-specification.txt). icculus.org. Retrieved 2010-07-25.
26. Gordon, Ryan. "FatELF: Turns out I liked the uncertainty better" (http://icculus.org/cgi-bin/finger/finger.pl?user=icculus&date=2009-11-03&time=19-08-04). icculus.org. Retrieved 2010-07-13.
27. Holwerda, Thom (2009-11-03). "Ryan Gordon Halts FatELF Project" (http://www.osnews.com/comments/22446). osnews.com. Retrieved 2010-07-05.
28. Brockmeier, Joe (June 23, 2010). "SELF: Anatomy of an (alleged) failure" (https://lwn.net/Articles/392862/). Linux Weekly News. Retrieved 2011-02-06.

Further reading

- Levine, John R. (October 1999). *Linkers and Loaders* (http://www.iecc.com/linker/). Morgan Kaufmann. ISBN 1-55860-496-0.
- Drepper, Ulrich (2006-08-20). "How To Write Shared Libraries" (http://people.redhat.com/drepper/dsohowto.pdf) (PDF). 4.0. Retrieved 2007-06-20.
- *An unsung hero: The hardworking ELF (https://web.archive.org/web/20070224140341/http://www-128.ibm.com/developerworks/power/library/pa-spec12/)* by Peter Seebach, December 20, 2005, archived from the original on February 24, 2007
- LibElf and GElf - A Library to Manipulate ELf Files (https://web.archive.org/web/20040225174057/http://developers.sun.com/solaris/articles/elf.html) at the Wayback Machine (archived February 25, 2004)
- *The ELF Object File Format by Dissection (https://www.linuxjournal.com/article/1060)* by Eric Youngdale (1995-05-01)
- *A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux (http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html)* by Brian Raiter
- ELF relocation into non-relocatable objects (http://www.phrack.org/issues.html?issue=61&id=8#article) by Julien Vanegue (2003-08-13)
- Embedded ELF debugging without ptrace (http://www.phrack.org/issues.html?issue=63&id=9#article) by the ELFsh team (2005-08-01)
- *Study of ELF loading and relocs (http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html)* by Pat Beirne (1999-08-03)

External links

- FreeBSD Handbook: Binary formats (https://web.archive.org/web/20130403001804/http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/binary-formats.html) (archived version)
- FreeBSD `elf(5)` manual page (http://www.freebsd.org/cgi/man.cgi?query=elf&sektion=5)
- NetBSD ELF FAQ (http://www.netbsd.org/Documentation/elf.html)
- Linux `elf(5)` manual page (https://manpages.debian.org/stretch/manpages/elf.5.en.html)
- Oracle Solaris Linker and Libraries Guide (http://www.oracle.com/pls/topic/lookup?ctx=solaris11&id=OSLLG)
- The ERESI project : reverse engineering on ELF-based operating systems (http://www.eresi-project.org/)
- Linux Today article on 86open (http://www.linuxtoday.com/developer/1999072600605PS) July 26, 1999
- Announcement of 86open on Debian Announce mailing list (http://lists.debian.org/debian-announce/1997/msg00028.html) October 10, 1997, Bruce Perens
- Declaration of Ulrich Drepper (PDF) (http://www.groklaw.net/pdf/IBM-835-Exhibit_184.pdf) in The SCO Group vs IBM, September 19, 2006
- 86open and ELF discussion (http://www.groklaw.net/articlebasic.php?story=20060813114048520) on Groklaw, August 13, 2006
