WIKIPEDIA
The Free Encyclopedia

Executable and Linkable Format

In computing, the **Executable and Linkable Format**[2] (**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),[3] 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.

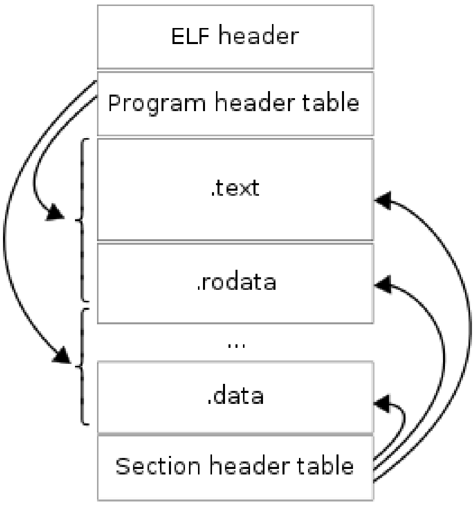
Executable and Linkable Format

Filename extension	`none, .axf, .bin, .elf, .o, .out, .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

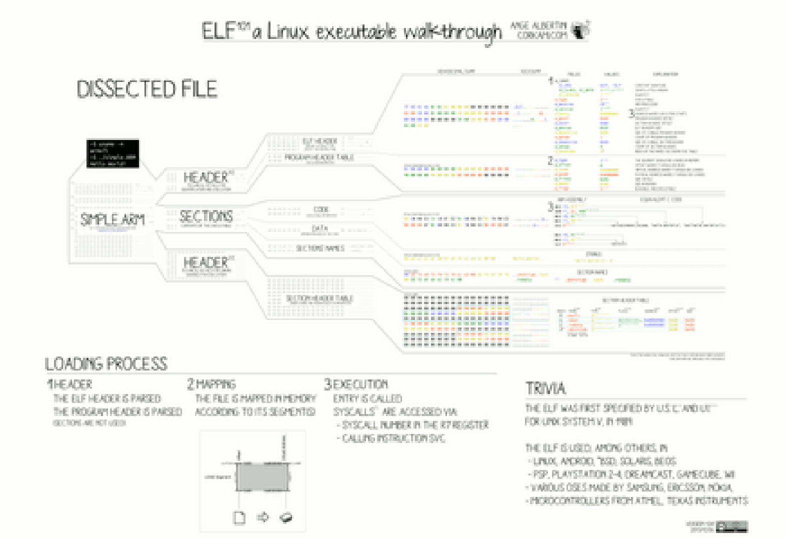
File layout

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



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*.


ELF101 a Linux executable walk-through
ANGE ALBERTINI CORKAMI.COM
DISSECTED FILE
HEADER
SIMPLE ARM
SECTIONS
HEADER
LOADING PROCESS
1 HEADER
THE ELF HEADER IS PARSED
THE PROGRAM HEADER IS PARSED
SECTIONS ARE NOT USED
2 MAPPING
THE FILE IS MAPPED IN MEMORY ACCORDING TO ITS SEGMENT(S)
3 EXECUTION
ENTRY IS CALLED
SYSCALLS ARE ACCESSED VIA:
- SYSCALL NUMBER IN THE R7 REGISTER
- CALLING INSTRUCTION SVC
TRIVIA
THE ELF WAS FIRST SPECIFIED BY U.S.L. AND UI FOR UNIX SYSTEM V, IN 1989
THE ELF IS USED, AMONG OTHERS, IN
- LINUX, ANDROID, *BSD, SOLARIS, BEOS
- PSP, PLAYSTATION 2-4, DREAMCAST, GAMECUBE, WII
- VARIOUS OSES MADE BY SAMSUNG, ERICSSON, NOKIA,
- MICROCONTROLLERS FROM ATMEL, TEXAS INSTRUMENTS

Structure of an ELF file with key entries highlighted

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.

```
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[4]

File header

The ELF header defines whether to use 32-bit 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[5]

<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><code>e_ident[EI_MAG0]</code> through <code>e_ident[EI_MAG3]</code></td><td><code>0x7F</code> followed by <code>ELF</code>(<code>45 4c 46</code>) in ASCII; these four bytes constitute the magic number.</td></tr>
<tr><td colspan="2">0x04</td><td colspan="2">1</td><td><code>e_ident[EI_CLASS]</code></td><td>This byte is set to either <code>1</code> or <code>2</code> to signify 32- or 64-bit format, respectively.</td></tr>
<tr><td colspan="2">0x05</td><td colspan="2">1</td><td><code>e_ident[EI_DATA]</code></td><td>This byte is set to either <code>1</code> or <code>2</code> to signify little or big endianness, respectively. This affects interpretation of multi-byte fields starting with offset <code>0x10</code>.</td></tr>
<tr><td colspan="2">0x06</td><td colspan="2">1</td><td><code>e_ident[EI_VERSION]</code></td><td>Set to <code>1</code> for the original and current version of ELF.</td></tr>
<tr><td colspan="2">0x07</td><td colspan="2">1</td><td><code>e_ident[EI_OSABI]</code></td><td>Identifies the target operating system ABI.
<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 (Monterey)</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>FenixOS</td></tr>
<tr><td>0x11</td><td>Nuxi CloudABI</td></tr>
<tr><td>0x12</td><td>Stratus Technologies OpenVOS</td></tr>
</table></td></tr>
<tr><td colspan="2">0x08</td><td colspan="2">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,[6] so it is ignored for statically-linked executables. In that case, offset and size of EI_PAD are <code>8</code>.
glibc 2.12+ in case <code>e_ident[EI_OSABI] == 3</code> treats this field as ABI version of the dynamic linker:[7] it defines a list of dynamic linker's features,[8] treats <code>e_ident[EI_ABIVERSION]</code> as</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td>a feature level requested by the shared object (executable or dynamic library) and refuses to load it if an unknown feature is requested, i.e. <code>e_ident[EI_ABIVERSION]</code> is greater than the largest known feature.[9]</td></tr>
<tr><td>0x09</td><td>7</td><td><code>e_ident[EI_PAD]</code></td><td>Reserved padding bytes. Currently unused. Should be filled with zeros and ignored when read.</td></tr>
<tr><td>0x10</td><td>2</td><td><code>e_type</code></td><td>Identifies object file type.
<table>
<tr><th>Value</th><th>Type</th><th>Meaning</th></tr>
<tr><td>0x00</td><td>ET_NONE</td><td>Unknown.</td></tr>
<tr><td>0x01</td><td>ET_REL</td><td>Relocatable file.</td></tr>
<tr><td>0x02</td><td>ET_EXEC</td><td>Executable file.</td></tr>
<tr><td>0x03</td><td>ET_DYN</td><td>Shared object.</td></tr>
<tr><td>0x04</td><td>ET_CORE</td><td>Core file.</td></tr>
<tr><td>0xFE00</td><td>ET_LOOS</td><td rowspan="2">Reserved inclusive range. Operating system specific.</td></tr>
<tr><td>0xFEFF</td><td>ET_HIOS</td></tr>
<tr><td>0xFF00</td><td>ET_LOPROC</td><td rowspan="2">Reserved inclusive range. Processor specific.</td></tr>
<tr><td>0xFFFF</td><td>ET_HIPROC</td></tr>
</table>
</td></tr>
<tr><td>0x12</td><td>2</td><td><code>e_machine</code></td><td>Specifies target instruction set architecture. Some examples are:</td></tr>
</table>

Value	ISA
0x00	No specific instruction set
0x01	AT&T WE 32100
0x02	SPARC
0x03	x86
0x04	Motorola 68000 (M68k)
0x05	Motorola 88000 (M88k)
0x06	Intel MCU
0x07	Intel 80860
0x08	MIPS
0x09	IBM System/370
0x0A	MIPS RS3000 Little-endian
0x0B – 0x0E	Reserved for future use
0x0F	Hewlett-Packard PA-RISC
0x13	Intel 80960
0x14	PowerPC
0x15	PowerPC (64-bit)
0x16	S390, including S390x
0x17	IBM SPU/SPC
0x18 – 0x23	Reserved for future use
0x24	NEC V800
0x25	Fujitsu FR20
0x26	TRW RH-32
0x27	Motorola RCE
0x28	Arm (up to Armv7/AArch32)
0x29	Digital Alpha
0x2A	SuperH
0x2B	SPARC Version 9
0x2C	Siemens TriCore embedded processor
0x2D	Argonaut RISC Core
0x2E	Hitachi H8/300
0x2F	Hitachi H8/300H
0x30	Hitachi H8S
0x31	Hitachi H8/500
0x32	IA-64
0x33	Stanford MIPS-X

0x34	Motorola ColdFire
0x35	Motorola M68HC12
0x36	Fujitsu MMA Multimedia Accelerator
0x37	Siemens PCP
0x38	Sony nCPU embedded RISC processor
0x39	Denso NDR1 microprocessor
0x3A	Motorola Star*Core processor
0x3B	Toyota ME16 processor
0x3C	STMicroelectronics ST100 processor
0x3D	Advanced Logic Corp. TinyJ embedded processor family
0x3E	AMD x86-64
0x3F	Sony DSP Processor
0x40	Digital Equipment Corp. PDP-10
0x41	Digital Equipment Corp. PDP-11
0x42	Siemens FX66 microcontroller
0x43	STMicroelectronics ST9+ 8/16 bit microcontroller
0x44	STMicroelectronics ST7 8-bit microcontroller
0x45	Motorola MC68HC16 Microcontroller
0x46	Motorola MC68HC11 Microcontroller
0x47	Motorola MC68HC08 Microcontroller
0x48	Motorola MC68HC05 Microcontroller
0x49	Silicon Graphics SVx
0x4A	STMicroelectronics ST19 8-bit microcontroller
0x4B	Digital VAX
0x4C	Axis Communications 32-bit embedded processor
0x4D	Infineon Technologies 32-bit embedded processor
0x4E	Element 14 64-bit DSP Processor
0x4F	LSI Logic 16-bit DSP Processor
0x8C	TMS320C6000 Family
0xAF	MCST Elbrus e2k
0xB7	Arm 64-bits (Armv8/AArch64)
0xDC	Zilog Z80
0xF3	RISC-V
0xF7	Berkeley Packet Filter

					0x101 WDC 65C816
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 (byte 0x04). If the file doesn't have an associated entry point, then this holds zero.
0x1C	0x20	4	8	e_phoff	Points to the start of the program header table. It usually follows the file header immediately following this one, 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.
0x34	0x40				End of ELF Header (size).

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[10]

<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><code>p_type</code></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><code>PT_NULL</code></td><td>Program header table entry unused.</td></tr>
<tr><td>0x00000001</td><td><code>PT_LOAD</code></td><td>Loadable segment.</td></tr>
<tr><td>0x00000002</td><td><code>PT_DYNAMIC</code></td><td>Dynamic linking information.</td></tr>
<tr><td>0x00000003</td><td><code>PT_INTERP</code></td><td>Interpreter information.</td></tr>
<tr><td>0x00000004</td><td><code>PT_NOTE</code></td><td>Auxiliary information.</td></tr>
<tr><td>0x00000005</td><td><code>PT_SHLIB</code></td><td>Reserved.</td></tr>
<tr><td>0x00000006</td><td><code>PT_PHDR</code></td><td>Segment containing program header table itself.</td></tr>
<tr><td>0x00000007</td><td><code>PT_TLS</code></td><td>Thread-Local Storage template.</td></tr>
<tr><td>0x60000000</td><td><code>PT_LOOS</code></td><td rowspan="2">Reserved inclusive range. Operating system specific.</td></tr>
<tr><td>0x6FFFFFFF</td><td><code>PT_HIOS</code></td></tr>
<tr><td>0x70000000</td><td><code>PT_LOPROC</code></td><td rowspan="2">Reserved inclusive range. Processor specific.</td></tr>
<tr><td>0x7FFFFFFF</td><td><code>PT_HIPROC</code></td></tr>
</table></td></tr>
<tr><td></td><td>0x04</td><td></td><td>4</td><td><code>p_flags</code></td><td>Segment-dependent flags (position for 64-bit structure).
<table>
<tr><th>Value</th><th>Name</th><th>Meaning</th></tr>
<tr><td>0x1</td><td><code>PF_X</code></td><td>Executable segment.</td></tr>
<tr><td>0x2</td><td><code>PF_W</code></td><td>Writeable segment.</td></tr>
<tr><td>0x4</td><td><code>PF_R</code></td><td>Readable segment.</td></tr>
</table></td></tr>
<tr><td>0x04</td><td>0x08</td><td>4</td><td>8</td><td><code>p_offset</code></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><code>p_vaddr</code></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><code>p_paddr</code></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><code>p_filesz</code></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><code>p_memsz</code></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><code>p_flags</code></td><td>Segment-dependent flags (position for 32-bit structure). See above <code>p_flags</code> field for flag definitions.</td></tr>
<tr><td>0x1C</td><td>0x30</td><td>4</td><td>8</td><td><code>p_align</code></td><td><code>0</code> and <code>1</code> specify no alignment. Otherwise should be a positive, integral power of 2, with <code>p_vaddr</code> equating <code>p_offset</code> modulus <code>p_align</code>.</td></tr>
<tr><td>0x20</td><td>0x38</td><td colspan="3"></td><td>End of Program Header (size).</td></tr>
</table>

Section header

<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>sh_name</td><td>An offset to a string in the <b>.shstrtab</b> section that represents the name of this section.</td></tr>
<tr><td colspan="2">0x04</td><td colspan="2">4</td><td>sh_type</td><td>Identifies the type of this header.
<table>
<tr><th>Value</th><th>Name</th><th>Meaning</th></tr>
<tr><td>0x0</td><td>SHT_NULL</td><td>Section header table entry unused</td></tr>
<tr><td>0x1</td><td>SHT_PROGBITS</td><td>Program data</td></tr>
<tr><td>0x2</td><td>SHT_SYMTAB</td><td>Symbol table</td></tr>
<tr><td>0x3</td><td>SHT_STRTAB</td><td>String table</td></tr>
<tr><td>0x4</td><td>SHT_RELA</td><td>Relocation entries with addends</td></tr>
<tr><td>0x5</td><td>SHT_HASH</td><td>Symbol hash table</td></tr>
<tr><td>0x6</td><td>SHT_DYNAMIC</td><td>Dynamic linking information</td></tr>
<tr><td>0x7</td><td>SHT_NOTE</td><td>Notes</td></tr>
<tr><td>0x8</td><td>SHT_NOBITS</td><td>Program space with no data (bss)</td></tr>
<tr><td>0x9</td><td>SHT_REL</td><td>Relocation entries, no addends</td></tr>
<tr><td>0x0A</td><td>SHT_SHLIB</td><td>Reserved</td></tr>
<tr><td>0x0B</td><td>SHT_DYNSYM</td><td>Dynamic linker symbol table</td></tr>
<tr><td>0x0E</td><td>SHT_INIT_ARRAY</td><td>Array of constructors</td></tr>
<tr><td>0x0F</td><td>SHT_FINI_ARRAY</td><td>Array of destructors</td></tr>
<tr><td>0x10</td><td>SHT_PREINIT_ARRAY</td><td>Array of pre-constructors</td></tr>
<tr><td>0x11</td><td>SHT_GROUP</td><td>Section group</td></tr>
<tr><td>0x12</td><td>SHT_SYMTAB_SHNDX</td><td>Extended section indices</td></tr>
<tr><td>0x13</td><td>SHT_NUM</td><td>Number of defined types.</td></tr>
<tr><td>0x60000000</td><td>SHT_LOOS</td><td>Start OS-specific.</td></tr>
<tr><td>...</td><td>...</td><td>...</td></tr>
</table></td></tr>
<tr><td colspan="2">0x08</td><td>4</td><td>8</td><td>sh_flags</td><td>Identifies the attributes of the section.</td></tr>
</table>

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 null-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)

0x0C	0x10	4	8	sh_addr	Virtual address of the section in memory, for sections that are loaded.
0x10	0x18	4	8	sh_offset	Offset of the section in the file image.
0x14	0x20	4	8	sh_size	Size in bytes of the section in the file image. May be 0.
0x18	0x28	4		sh_link	Contains the section index of an associated section. This field is used for several purposes, depending on the type of section.
0x1C	0x2C	4		sh_info	Contains extra information about the section. This field is used for several purposes, depending on the type of section.
0x20	0x30	4	8	sh_addralign	Contains the required alignment of the section. This field must be a power of two.
0x24	0x38	4	8	sh_entsize	Contains the size, in bytes, of each entry, for sections that contain fixed-size entries. Otherwise, this field contains zero.
0x28	0x40				End of Section Header (size).

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.[11]

- `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[12]
- NetBSD
- OpenBSD
- Redox
- DragonFly BSD
- Syllable
- HP-UX (except for 32-bit PA-RISC programs which continue to use SOM)
- QNX Neutrino
- MINIX[13]

Non-Unix adoption

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

- OpenVMS, in its Itanium and amd64 versions[14]
- 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[15]
- Stratus VOS, in PA-RISC and x86 versions
- SkyOS
- Fuchsia OS
- Z/TPF
- HPE NonStop OS[16]
- Deos

Microsoft Windows also uses the ELF format, but only for its Windows Subsystem for Linux compatibility system.[17]

Game consoles

Some game consoles also use ELF:

- PlayStation Portable,[18] PlayStation Vita, PlayStation (console), PlayStation 2, PlayStation 3, PlayStation 4, PlayStation 5
- GP2X
- Dreamcast
- GameCube
- Nintendo 64
- Wii
- Wii U

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
- Café OS (The operating system ran on Wii U)

Mobile phones

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

- Symbian OS v9 uses E32Image[19] 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[20]) 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.[21] It also possible to use native Linux software from package managers like Termux, or compile them from sources via Clang or GCC, that also available in repositories.

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[22] 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 (https://github.com/ARM-software/abi-aa/releases/download/2022Q1/aaelf32.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) Archived (https://web.archive.org/web/20120401235051/http://sources.redhat.com/ml/binutils/2003-06/msg00436.html) 2012-04-01 at the Wayback Machine* (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)
- RISC-V:
  - *RISC-V ELF Specification (https://github.com/riscv-non-isa/riscv-elf-psabi-doc/blob/master/riscv-elf.adoc)*
- 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.[23] For example, that is the case for the System V ABI, AMD64 Supplement.[24][25]

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.[26] 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,[27] 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.[28]

FatELF: universal binaries for Linux

FatELF is an ELF binary-format extension that adds fat binary capabilities.[29] 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 2021, FatELF has not been integrated into the mainline Linux kernel.[30][31][32]

See also

Computer programming portal

- 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. Tool Interface Standard (TIS) *Portable Formats Specification (https://refspecs.linuxfoundation.org/elf/TIS1.1.pdf)* *Version 1.1* (October 1993)
3. *System V Application Binary Interface (http://www.sco.com/developers/devspecs/gabi41.pdf)* Edition 4.1 (1997-03-18)
4. "Available lexers — Pygments" (http://pygments.org/docs/lexers#lexers-for-hexadecimal-dumps). *pygments.org*.
5. "ELF Header" (http://www.sco.com/developers/gabi/2000-07-17/ch4.eheader.html). Sco.com. July 2000. Retrieved 2014-02-07.
6. "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.
7. "glibc 2.12 announce" (https://sourceware.org/ml/libc-alpha/2010-05/msg00000.html).
8. "sourceware.org Git - glibc.git/blob - libc-abis" (https://sourceware.org/git/?p=glibc.git;a=blob;f=libc-abis;h=e702f6ae245c1528f5a608df8cfae4f037809de2;hb=HEAD).
9. "sourceware.org Git - glibc.git/blob - sysdeps/gnu/ldsodefs.h" (https://web.archive.org/web/20210307223850/https://sourceware.org/git/?p=glibc.git%3Ba%3Dblob%3Bf%3Dsysdeps%2Fgnu%2Fldsodefs.h%3Bh%3D253b4d934c5ca457ab5580a2a6398205e4f961ba%3Bhb%3DHEAD#l32). Archived from the original (https://sourceware.org/git/?p=glibc.git%3Ba%3Dblob%3Bf%3Dsysdeps%2Fgnu%2Fldsodefs.h%3Bh%3D253b4d934c5ca457ab5580a2a6398205e4f961ba%3Bhb%3DHEAD#l32) on 2021-03-07. Retrieved 2019-10-28.
10. "Program Header" (http://www.sco.com/developers/gabi/2000-07-17/ch5.pheader.html). Sco.com. July 2000. Retrieved 2017-04-05.
11. "elfutils" (https://sourceware.org/elfutils/). *sourceware.org*. Retrieved 30 April 2017.
12. "Binary Formats" (https://web.archive.org/web/20190331192807/https://docs.freebsd.org/doc/4.9-RELEASE/usr/share/doc/handbook/binary-formats.html). Archived from the original (https://docs.freebsd.org/doc/4.9-RELEASE/usr/share/doc/handbook/binary-formats.html) on 2019-03-31. Retrieved 2019-03-31.
13. "MinixReleases — Minix Wiki" (https://web.archive.org/web/20130330150621/http://wiki.minix3.org/en/MinixReleases). Wiki.minix3.org. Archived from the original (http://wiki.minix3.org/en/MinixReleases) on 2013-03-30. Retrieved 2014-01-19.
14. "Archived copy" (https://web.archive.org/web/20200915215111/https://vmssoftware.com/pdfs/State_of_Port_20160906.pdf) (PDF). Archived from the original (https://vmssoftware.com/pdfs/State_of_Port_20160906.pdf) (PDF) on 2020-09-15. Retrieved 2016-10-19.
15. "GCCSDK — RISC OS" (http://www.riscos.info/index.php/GCCSDK). Riscos.info. 2012-04-22. Retrieved 2014-01-19.
16. "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) Archived (https://web.archive.org/web/20180530042640/http://h20628.www2.hp.com/km-ext/kmcsdirect/emr_na-c02543407-12.pdf) 2018-05-30 at the Wayback Machine on 2018-5-30
17. Foley, Mary Jo. "Under the hood of Microsoft's Windows Subsystem for Linux | ZDNet" (https://www.zdnet.com/article/under-the-hood-of-microsofts-windows-subsystem-for-linux/). *ZDNet*. Retrieved 2016-08-19.
18. PlayStation Portable use encrypted & relocated ELF : PSP

19. *Symbian OS executable file format (https://web.archive.org/web/20091213034509/http://wiki.forum.nokia.com/index.php/E32Image)*
20. Rosen, Kenneth; Host, Douglas; Klee, Rachel; Rosinski, Richard (2007). *UNIX: The Complete Reference* (https://books.google.com/books?id=2Et--84HlkwC) (2 ed.). McGraw Hill Professional. p. 707. ISBN 9780071706988. Retrieved 2017-06-08. "Dynamically linked libraries are also called shared objects (.so)."
21. Thomas, Romain. "Android formats" (https://archive.today/20230216233708/https://lief-project.github.io/doc/latest/tutorials/10_android_formats.html). *Quarks Lab*. Archived from the original (https://lief-project.github.io/doc/latest/tutorials/10_android_formats.html) on 16 Feb 2023. Retrieved 17 Jan 2023.
22. "Chapter 4: Object Files" (http://www.sco.com/developers/gabi/2009-10-26/ch4.eheader.html#e_machine), *System V Application Binary Interface*, 2009-10-26, e_machine
23. "LSB Referenced Specifications" (http://refspecs.linuxfoundation.org/lsb.shtml). *linuxfoundation.org*. Retrieved 27 April 2015.
24. "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.
25. "Introduction" (http://refspecs.linuxfoundation.org/LSB_4.1.0/LSB-Core-AMD64/LSB-Core-AMD64/elfintro.html). *linuxfoundation.org*. Retrieved 27 April 2015.
26. 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.
27. 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.
28. 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.
29. Gordon, Ryan. "fatelf-specification v1" (http://hg.icculus.org/icculus/fatelf/raw-file/tip/docs/fatelf-specification.txt). icculus.org. Retrieved 2010-07-25.
30. 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.
31. Holwerda, Thom (2009-11-03). "Ryan Gordon Halts FatELF Project" (https://www.osnews.com/story/22446/ryan-gordon-halts-fatelf-project). osnews.com. Retrieved 2010-07-05.
32. 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. (2000) [October 1999]. *Linkers and Loaders* (https://www.iecc.com/linker/). The Morgan Kaufmann Series in Software Engineering and Programming (1 ed.). San Francisco, USA: Morgan Kaufmann. ISBN 1-55860-496-0. OCLC 42413382 (https://www.worldcat.org/oclc/42413382). Archived (https://archive.today/20121205032107/http://www.iecc.com/linker/) from the original on 2012-12-05. Retrieved 2020-01-12. Code: [1] (https://archive.today/20200114225034/https://linker.iecc.com/code.html)[2] (ftp://ftp.iecc.com/pub/linker/) Errata: [3] (https://linker.iecc.com/)
- 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: Introduction (https://www.linuxjournal.com/article/1059)*, *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/) Archived (https://web.archive.org/web/20210314190846/http://www.eresi-project.org/) 2021-03-14 at the Wayback Machine
- 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) Archived (https://web.archive.org/web/20190201013659/http://www.groklaw.net/articlebasic.php?story=20060813114048520) 2019-02-01 at the Wayback Machine on Groklaw, August 13, 2006

Retrieved from "https://en.wikipedia.org/w/index.php?title=Executable_and_Linkable_Format&oldid=1174409285"