NAME

elf_update - update an ELF descriptor

LIBRARY

ELF Access Library (libelf, -lelf)

SYNOPSIS

#include <libelf.h>

off_t
elf_update(*Elf *elf, Elf_Cmd cmd*);

DESCRIPTION

Function **elf_update**() causes the library to recalculate the structure of an ELF object and optionally write out the image of the object to file.

Argument *elf* should reference a valid ELF descriptor.

Argument *cmd* can be one of the following values:

ELF_C_NULL — The library will recalculate structural information flagging modified structures with the ELF_F_DIRTY flag, but will not write data to the underlying file image.

ELF_C_WRITE — The library will recalculate structural information and will also write the new image to the underlying file. The ELF descriptor referenced by argument *elf* should permit the underlying ELF object to be written or updated (see elf_begin(3)).

All pointers to *Elf_Scn* and *Elf_Data* descriptors associated with descriptor *elf* should be considered invalid after a call to **elf_update**().

Specifying Object Layout

The ELF Access Library (libelf, -lelf) supports two layout modes.

Library Layout

If the ELF_F_LAYOUT flag is not set on the ELF descriptor, the ELF library will lay out the ELF object according to the following scheme:

EHDR — The ELF executable header will be placed at the start of the object.

PHDR — If the ELF descriptor contains a program header table, it will be placed after the Executable Header.

Section Data ELF section data, if any, will be placed next, keeping each section's alignment requirements in mind.

SHDR The ELF section header table, if any, will be placed last.

Application Controlled Layout

The application can take full control of the layout of the ELF object by setting the ELF_F_LAYOUT flag on the ELF descriptor (see elf_flagelf(3)). In this case the library will lay out the ELF object using application-supplied information as below:

EHDR The ELF executable header will be placed at the start of the object.

PHDR The ELF program header table, if any, it will be placed at the offset specified in the *e_phoff* field of the ELF executable header.

Section Data The data for each ELF section will be placed at the offset specified by the *sh_offset* field of the section's header. The size of the section will be taken from the *sh_size* field of the section header.

SHDR The ELF section header table, if any, will be placed at the offset specified by the *e_shoff* field of the executable header.

Gaps in the coverage of the file's contents will be set to the fill value specified by elf_fill(3).

Application Supplied Information

The application needs to set the following fields in the data structures associated with the ELF descriptor prior to calling **elf_update**().

Executable Header

The fields of the ELF executable header that need to be set by the application are:

e_entry To be set to the desired entry address for executables.

e_flags To be set to the desired processor specific flags.

e_ident[EI_DATA] Must be set to one of ELFDATA2LSB or ELFDATA2MSB.

e_ident[EI_OSABI] To be set to the OS ABI desired. For example, for FreeBSD executables, this field should be set to ELFOSABI_FREEBSD.

e_machine To be set to the desired machine architecture, one of the EM_* values in the header file <*elfdefinitions.h*>.

e_phoff If the application is managing the object's layout, it must set this field to the file offset of the ELF program header table.

e_shoff If the application is managing the object's layout, it must set this field to the file offset of the ELF section header table.

e_shstrndx To be set to the index of the string table containing section names.

e_type To be set to the type of the ELF object, one of the ET_* values in the

header file <*elfdefinitions.h*>.

e_version To be set to the desired version of the ELF object.

Program Header

All fields of the entries in the program header table need to be set by the application.

Section Header

The fields of ELF section headers that need to be set by the application are:

sh_addr To be set to the memory address where the section should reside.

sh_addralign If the application is managing the file layout, it must set this field to the desired alignment for the section's contents. This value must be a power of two and must be at least as large as the largest alignment needed by any *Elf_Data* descriptor associated with the section.

sh_entsize To be set to the size of each entry, for sections containing fixed size elements, or set to zero for sections without fixed size elements. If the application is not managing file layout, it may leave this field as zero for those sections whose types are known to the library.

sh_flags To be set to the desired section flags.

sh_info To be set as described in elf(5).

sh_link To be set as described in elf(5).

sh_name To be set to the index of the section's name in the string table containing section names.

sh_offset If the application is managing the file layout, it must set this field to the file offset of the section's contents.

sh_size If the application is managing the file layout, it must set this field to the file size of the section's contents.

sh_type To be set to the type of the section.

Section Data

The *Elf_Data* descriptors associated with each section specify its contents (see elf_getdata(3)). While all the fields in these descriptors are under application control, the following fields influence object layout:

d_align To be set to the desired alignment, within the containing section, of the descriptor's data.

d_off If the application is managing object layout, it must set this field to the file offset, within the section, at which the descriptor's data should be placed.

d_size To be set to the size in bytes of the memory representation of the descriptor's data.

RETURN VALUES

Function **elf_update**() returns the total size of the file image if successful, or -1 if an error occurred.

ERRORS

This function may fail with the following errors:

[ELF_E_ARGUMENT]
: Argument *elf* was null.

[ELF_E_ARGUMENT]
: Argument *cmd* was not recognized.

[ELF_E_ARGUMENT]
: The argument *elf* was not a descriptor for an ELF object.

[ELF_E_CLASS] The *e_ident[EI_CLASS]* field of the executable header of argument *elf* did not match the class of the file.

[ELF_E_DATA] An *Elf_Data* descriptor contained in argument *elf* specified an unsupported type.

[ELF_E_DATA] An *Elf_Data* descriptor specified an alignment that was zero or was not a power of two.

[ELF_E_HEADER] The ELF header in argument *elf* requested a different byte order from the byte order already associated with the file.

[ELF_E_IO] An I/O error was encountered.

[ELF_E_LAYOUT] An *Elf_Data* descriptor contained in argument *elf* specified an alignment incompatible with its containing section.

[ELF_E_LAYOUT] Argument *elf* contained section descriptors that overlapped in extent.

[ELF_E_LAYOUT] Argument *elf* contained section descriptors that were incorrectly aligned or were too small for their data.

[ELF_E_LAYOUT] The flag ELF_F_LAYOUT was set on the Elf descriptor and the executable header overlapped with the program header table.

[ELF_E_LAYOUT] The flag ELF_F_LAYOUT was set on the Elf descriptor and the program

header table was placed at a misaligned file offset.

[ELF_E_LAYOUT] The flag ELF_F_LAYOUT was set on the Elf descriptor and the section header table overlapped an extent mapped by a section descriptor.

[ELF_E_LAYOUT] The ELF_F_LAYOUT flag was set on the Elf descriptor, and the *d_offset* field in an *Elf_Data* descriptor contained a value that was not a multiple of the descriptor's specified alignment.

[ELF_E_MODE] An ELF_C_WRITE operation was requested with an ELF descriptor that was not opened for writing or updating.

[ELF_E_SECTION] Argument *elf* contained a section with an unrecognized type.

[ELF_E_SECTION] The section header at index SHN_UNDEF had an illegal section type.

[ELF_E_SEQUENCE] An ELF_C_WRITE operation was requested after a prior call to **elf_cntl**(*elf*, *ELF_C_FDDONE*) disassociated the ELF descriptor *elf* from its underlying file.

[ELF_E_UNIMPL] Argument *elf* contained a section with an unsupported ELF type.

[ELF_E_VERSION] Argument *elf* had an unsupported version or contained an *Elf_Data* descriptor with an unsupported version.

SEE ALSO

elf(3), elf32_getehdr(3), elf32_getphdr(3), elf32_newehdr(3), elf32_newphdr(3), elf64_getehdr(3), elf64_getphdr(3), elf64_newehdr(3), elf64_newphdr(3), elf_begin(3), elf_cntl(3), elf_fill(3), elf_flagehdr(3), elf_flagelf(3), elf_getdata(3), elf_getscn(3), elf_newdata(3), elf_newscn(3), elf_rawdata(3), gelf(3), gelf_newehdr(3), gelf_newphdr(3), elf(5)