NAME

elf_update – update an ELF descriptor

SYNOPSIS

cc [*flag* ...] *file* ... **–lelf** [*library* ...]

#include <libelf.h>

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

MT-LEVEL

Unsafe

DESCRIPTION

elf_update( ) causes the library to examine the information associated with an ELF descriptor, *elf*, and to recalculate the structural data needed to generate the file's image.

cmd may have the following values:

ELF_C_NULL This value tells **elf_update( )** to recalculate various values, updating only the ELF descriptor's memory structures. Any modified structures are flagged with the **ELF_F_DIRTY** bit. A program thus can update the structural information and then reexamine them without changing the file associated with the ELF descriptor. Because this does not change the file, the ELF descriptor may allow reading, writing, or both reading and writing (see **elf_begin**(3E)).

ELF_C_WRITE If *cmd* has this value, **elf_update( )** duplicates its **ELF_C_NULL** actions and also writes any ''dirty'' information associated with the ELF descriptor to the file. That is, when a program has used **elf_getdata**(3E) or the **elf_flagdata**(3E) facilities to supply new (or update existing) information for an ELF descriptor, those data will be examined, coordinated, translated if necessary (see **elf32_xlatetof**(3E)), and written to the file. When portions of the file are written, any **ELF_F_DIRTY** bits are reset, indicating those items no longer need to be written to the file (see **elf_flagdata**(3E)). The sections' data are written in the order of their section header entries, and the section header table is written to the end of the file.

When the ELF descriptor was created with **elf_begin( )**, it must have allowed writing the file. That is, the **elf_begin( )** command must have been either **ELF_C_RDWR** or **ELF_C_WRITE**.

If **elf_update( )** succeeds, it returns the total size of the file image (not the memory image), in bytes. Otherwise an error occurred, and the function returns –1.

When updating the internal structures, **elf_update( )** sets some members itself. Members listed below are the application's responsibility and retain the values given by the program.

The following table shows ELF Header members:

Member	Notes
e_ident[EI_DATA]	Library controls other **e_ident** values
e_type	
e_machine	
e_version	
e_entry	
e_phoff	Only when **ELF_F_LAYOUT** asserted
e_shoff	Only when **ELF_F_LAYOUT** asserted
e_flags	
e_shstrndx	

The following table shows the Program Header members:

Member	Notes
p_type	The application controls all
p_offset	program header entries
p_vaddr	
p_paddr	
p_filesz	
p_memsz	
p_flags	
p_align	

The following table shows the Section Header members:

Member	Notes
sh_name	
sh_type	
sh_flags	
sh_addr	
sh_offset	Only when **ELF_F_LAYOUT** asserted
sh_size	Only when **ELF_F_LAYOUT** asserted
sh_link	
sh_info	
sh_addralign	Only when **ELF_F_LAYOUT** asserted
sh_entsize	

The following table shows the Data Descriptor members:

Member	Notes
d_buf	
d_type	
d_size	
d_off	Only when **ELF_F_LAYOUT** asserted
d_align	
d_version	

Note that the program is responsible for two particularly important members (among others) in the ELF header. The **e_version** member controls the version of data structures written to the file. If the version is **EV_NONE**, the library uses its own internal version. The **e_ident[EI_DATA]** entry controls the data encoding used in the file. As a special case, the value may be **ELFDATANONE** to request the native data encoding for the host machine. An error occurs in this case if the native encoding doesn't match a file encoding known by the library.

Further note that the program is responsible for the **sh_entsize** section header member. Although the library sets it for sections with known types, it cannot reliably know the correct value for all sections. Consequently, the library relies on the program to provide the values for unknown section types. If the entry size is unknown or not applicable, the value should be set to zero.

When deciding how to build the output file, **elf_update( )** obeys the alignments of individual data buffers to create output sections. A section's most strictly aligned data buffer controls the section's alignment. The library also inserts padding between buffers, as necessary, to ensure the proper alignment of each buffer.

SEE ALSO

elf(3E), **elf32_fsize**(3E), **elf32_getehdr**(3E), **elf32_getshdr**(3E), **elf32_xlatetof**(3E), **elf_begin**(3E), **elf_flagdata**(3E), **elf_getdata**(3E)

NOTES

As mentioned above, the **ELF_C_WRITE** command translates data as necessary, before writing them to the file. This translation is *not* always transparent to the application program. If a program has obtained pointers to data associated with a file (for example, see **elf32_getehdr**(3E) and **elf_getdata**(3E)), the program should reestablish the pointers after calling **elf_update( )**.