**ELFIO**

**User's Guide**

**Serge Lamikhov-Center**

**ELFIO: User's Guide**

by Serge Lamikhov-Center

#### Table of Contents

Introduction ....................................................................................................................... v

[ELF Section Data Accessors 3](#_Toc341455215)

[ELFDump Utility 4](#_Toc341455216)

[ELF File Writer 4](#_Toc341455217)

[Chapter 2. ELFIO Library Classes 5](#_Toc341455218)

[Class elfio 5](#_Toc341455219)

[Data members 5](#_Toc341455220)

[Member functions 5](#_Toc341455221)

[Class section 7](#_Toc341455222)

[Member functions 8](#_Toc341455223)

#### List of Tables

2.1. Class elfio member functions ...................................................................................... 5 2.2. Class elfio member functions ...................................................................................... 5

2.3. Class section member functions .................................................................................. 7

**Introduction**

ELFIO is a C++ library for reading and generating files in ELF binary format. This library is independent and does not require any other product. It is also cross-platform - the library uses standard ANSI C++ constructions and runs on wide variety of architectures.

While the library's implementation does make your work much easier: basic knowledge of the ELF binary format is required. Information about ELF format can be found widely on the web.

**Chapter 1. Getting Started With ELFIO**

**ELF File Reader**

The ELFIO library is a header only library. No preparatory compilation steps are required. To make your application be aware about the ELFIO classes and types declarations, just include elfio.hpp header file. All ELFIO library declarations reside in ELFIO namespace. So, this tutorial code starts from the following code:

#include <iostream> #include <elfio.hpp>

using namespace ELFIO;

int main( int argc, char\*\* argv ) { if ( argc != 2 ) { std::cout << "Usage: tutorial <elf\_file>" << std::endl; return 1; }

Include elfio.hpp header file

The ELFIO namespace usage

This chapter will explain how to work with the reader portion of the ELFIO library. The first step would be creation of the elfio class instance. The elfio constructor does not receive any parameters. After creation of a class object, we initialize the instance by invoking load function passing ELF file name as a parameter.

// Create an elfio reader elfio reader; // Load ELF data if ( !reader.load( argv[1] ) ) { std::cout << "Can't find or process ELF file " << argv[1] << std::endl; return 2; }

Create elfio class instance

Initialize the instance by loading ELF file. The function load returns true if the ELF file was found and processed successfully. It returns false otherwise.

From here, ELF header properties are accessible. This makes it possible to request file parameters such as encoding, machine type, entry point, etc. To get the class and the encoding of the file use:

// Print ELF file properties std::cout << "ELF file class : "; if ( reader.get\_class() == ELFCLASS32 ) std::cout << "ELF32" << std::endl; else

std::cout << "ELF64" << std::endl; std::cout << "ELF file encoding : "; if ( reader.get\_encoding() == ELFDATA2LSB ) std::cout << "Little endian" << std::endl; else

std::cout << "Big endian" << std::endl;

Member function get\_class() returns ELF file class. Possible values are ELFCLASS32 or ELF-

CLASS64.

Member function get\_encoding() returns ELF file format encoding. Possible values are ELFDATA2LSB and ELFDATA2MSB.

**Note**

Standard ELF types, flags and constants are defined in the elf\_types.hpp header file. This file is included automatically into the project. For example: ELFCLASS32, ELFCLASS64 constants define a value for 32/64 bit architectures. ELFDATA2LSB and ELFDATA2MSB constants define value for little and big endian encoding.

ELF binary files may consist of several sections. Each section has it's own responsibility: some contain executable code; others describe program dependencies; others symbol tables and so on. See ELF binary format documentation for a full description of each section.

The following code demonstrates how to find out the amount of sections the ELF file contains. The code also presents how to access particular section properties like names and sizes:

// Print ELF file sections info Elf\_Half sec\_num = reader.sections.size(); std::cout << "Number of sections: " << sec\_num << std::endl; for ( int i = 0; i < sec\_num; ++i ) { const section\* psec = reader.sections[i]; std::cout << " [" << i << "] " << psec->get\_name()

<< "\t"

<< psec->get\_size()

<< std::endl;

// Access to section's data

// const char\* p = reader.sections[i]->get\_data()

}

sections member of reader object permits to obtain number of sections the ELF file contains. It also serves for getting access to individual section by using operator[], which returns a pointer to corresponding section's interface.

Similarly, segments of the ELF file can be processed:

// Print ELF file segments info Elf\_Half seg\_num = reader.segments.size(); std::cout << "Number of segments: " << seg\_num << std::endl; for ( int i = 0; i < seg\_num; ++i ) { const segment\* pseg = reader.segments[i]; std::cout << " [" << i << "] 0x" << std::hex

<< pseg->get\_flags()

<< "\t0x"

<< pseg->get\_virtual\_address()

<< "\t0x"

<< pseg->get\_file\_size()

<< "\t0x"

<< pseg->get\_memory\_size()

<< std::endl;

// Access to segments's data

// const char\* p = reader.segments[i]->get\_data()

}

In this case, segments' attributes and data are obtained by using segments member of the reader.

The full text of this example comes together with ELFIO library distribution.

## ELF Section Data Accessors

To simplify creation and interpretation of the ELF sections' data, the ELFIO library comes with auxiliary classes - accessors. To the moment of this document writing, the following accessors are available:

* string\_section\_accessor
* symbol\_section\_accessor
* relocation\_section\_accessor
* note\_section\_accessor

Definitely, it is possible to extend the library by implementing additional accessors serving particular purposes.

Let's see how the accessors can be used with the previous ELF file reader example. For this example purposes, we will print out all symbols in a symbol section.

if ( psec->get\_type() == SHT\_SYMTAB ) { const symbol\_section\_accessor symbols( reader, psec ); for ( unsigned int j = 0; j < symbols.get\_symbols\_num(); ++j ) { std::string name; Elf64\_Addr value; Elf\_Xword size; unsigned char bind; unsigned char type; Elf\_Half section\_index; unsigned char other; symbols.get\_symbol( j, name, value, size, bind, type, section\_index, other ); std::cout << j << " " << name << std::endl; }

}

We create symbol\_section\_accessor instance first. Usually, accessors receive the elfio and section\* parameters for their constructors. get\_symbol is used to retrieve a particular entry in the symbol table.

## ELFDump Utility

The source code for the ELF Dumping Utility can be found in the "examples" directory; there also located more examples on how to use different ELFIO reader interfaces.

## ELF File Writer

TODO

# Chapter 2. ELFIO Library Classes

## Class elfio

### Data members

The ELFIO library's main class is elfio. The class contains the following two public data members: sections and segments:

**Table 2.1. Class elfio member functions**

|  |  |
| --- | --- |
| **Data member** | **Description** |
| sections | The container stores ELFIO library section instances. Implements operator[] and size(). operator[] permits access to individual ELF file section according to its index. |
| segments | The container stores ELFIO library segment instances. Implements operator[] and size(). operator[] permits access to individual ELF file segment according to its index. |

### Member functions

Here is the list of elfio public member functions. Most of the functions permit to retrieve or set ELF file properties.

**Table 2.2. Class elfio member functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **elfio** (); | The constructor. |
| **~elfio** (); | The destructor. |
| void **create** (*file\_class*, *encoding*);  unsigned char *file\_class* ; unsigned char *encoding* ; | Cleans and initializes empty elfio object. *file\_class* is either ELFCLASS32 or ELFCLASS64. *file\_class* is either ELFDATA2LSB or ELFDATA2MSB. |
| bool **load** (*file\_name*); const std::string& *file\_name* ; | Initializes elfio object by loading data from ELF binary file. File name provided in *file\_name*.  Returns true if the file was processed successfully. |
| bool **save** (*file\_name*); const std::string& *file\_name* ; | Creates a file in ELF binary format. File name provided in *file\_name*. Returns true if the file was created successfully. |
| unsigned char **get\_class** (); | Returns ELF file class. Possible values are ELFCLASS32 or ELFCLASS64. |
| unsigned char **get\_elf\_version** (); | Returns ELF file format version. |

ELFIO Library Classes

|  |  |
| --- | --- |
| **Function** | **Description** |
| unsigned char **get\_encoding** (); | Returns ELF file format encoding. Possible values are ELFDATA2LSB and ELFDATA2MSB. |
| Elf\_Word **get\_version** (); | Identifies the object file version. |
| Elf\_Half **get\_header\_size** (); | Returns the ELF header's size in bytes. |
| Elf\_Half **get\_section\_entry\_size**  (); | Returns a section's entry size in ELF file header section table. |
| Elf\_Half **get\_segment\_entry\_size**  (); | Returns a segment's entry size in ELF file header program table. |
| unsigned char **get\_os\_abi** (); | Returns operating system ABI identification. |
| void **set\_os\_abi** (*value*); unsigned char *value* ; | Sets operating system ABI identification. |
| unsigned char **get\_abi\_version** (); | Returns ABI version. |
| void **set\_abi\_version** (*value*); unsigned char *value* ; | Sets ABI version. |
| Elf\_Half **get\_type** (); | Returns the object file type. |
| void **set\_type** (*value*); Elf\_Half *value* ; | Sets the object file type. |
| Elf\_Half **get\_machine** (); | Returns the object file's architecture. |
| void **set\_machine** (*value*); Elf\_Half *value* ; | Sets the object file's architecture. |
| Elf\_Word **get\_flags** (); | Returns processor-specific flags associated with the file. |
| void **set\_flags** (*value*); Elf\_Word *value* ; | Sets processor-specific flags associated with the file. |
| Elf64\_Addr **get\_entry** (); | Returns the virtual address to which the system first transfers control. |
| void **set\_entry** (*value*); Elf64\_Addr *value* ; | Sets the virtual address to which the system first transfers control. |
| Elf64\_Off **get\_sections\_offset** (); | Returns the section header table's file offset in bytes. |
| void **set\_sections\_offset** (*value*); Elf64\_Off *value* ; | Sets the section header table's file offset. Attention! The value can be overridden by the library, when it creates new ELF file layout. |

ELFIO Library Classes

|  |  |
| --- | --- |
| **Function** | **Description** |
| Elf64\_Off **get\_segments\_offset** (); | Returns the program header table's file offset. |
| void **set\_segments\_offset** (*value*); Elf64\_Off *value* ; | Sets the program header table's file offset. Attention! The value can be overridden by the library, when it creates new ELF file layout. |
| Elf\_Half  **get\_section\_name\_str\_index** (); | Returns the section header table index of the entry associated with the section name string table. |
| void **set\_section\_name\_str\_index** (*value*);  Elf\_Half *value* ; | Sets the section header table index of the entry associated with the section name string table. |
| endianess\_convertor& **get\_convertor** (); | Returns endianess convertor reference for the specific elfio object instance. |
| Elf\_Xword **get\_default\_entry\_size**  (*section\_type*);  Elf\_Word *section\_type* ; | Returns default entry size for known section types having different values on 32 and 64 bit architectures. At the moment, only SHT\_RELA,  SHT\_REL, SHT\_SYMTAB and SHT\_DYNAMIC  are 'known' section types. The function returns 0 for other section types. |

## Class section

Class section has no public data members.

### Member functions

Here is the list of section public member functions. These functions permit to retrieve or set ELF file section properties.

**Table 2.3. Class section member functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| Elf\_Half **get\_index** (); | Returns section index within ELF file. |
| Elf\_Half **get\_index** (); | Returns section index within ELF file. |
| Elf\_Half **get\_index** (); | Returns section index within ELF file. |