ELFIO
Tutorial

Serge Lamikhov-Center
(to_serge@users.sourceforge.net)

July 5, 2001

1 Introduction

ELFIO is a C++ library that permits you to read and generate files in ELF binary format. This library is not based on any other products and is platform independent. It uses only standard ANSI C++ language constructios and supposed to run on wide range of architectures.

While the library's implementation does it's best to make your work easy, a basic knowledge of ELF binary format is needed. You may read about ELF format in TIS (Tool Interface Standards) documentation that you received together with the sources of this library.

2 Getting started with ELFIO

2.1 Initialization

ELFIO library consists of two independent parts: ELF file reader (ELFI) and producer (ELFO)[1]. Each is represented by its own set of interfaces. The library doesn't contain any classes that we should explicitly instantiate. Instead, ELFIO provides us a set of interfaces that we should use to access the library functionality.

To make our program recognize all ELFIO interface classes, we need to include ELFIO.h header file. Doing this, we also define all standard definitions from TIS documentation.

```
#include <ELFIO.h>
```

In this tutorial, we'll see how to work with the reader part of ELFIO library. The first step that we should do, is to get a pointer onto the reader through the provided global builder:

[1] ELF file producer (ELFO) is under development. There were no public releases yet.

```
IELFI* pReader;
g_pELFIOBuilder->CreateELFI( &pReader );
```

Now, we have a pointer on IELFI interface, we should initialize the object by loading ELF file:

```
char* filename = ''file.o'';
pReader->Load( filename );
```

From here, we have an access to an ELF header. We may "ask" such file parameters as encoding, machine type, entry point, etc... Lets get an encoding of our file:

```
unsigned char encoding = pReader->GetEncoding();
```

Please note, standard types and constants from TIS document are defined in ELFTypes.h header file. This file is included automatically into our project. For example ELFDATA2LSB and ELFDATA2MSB constant defines a value for little and big endian encoding respectively.

2.2 ELF file sections

ELF binary file consist from several sections. Each section has it's own responsibility. There are sections that contain executable code, and there are sections that describes you program dependencies, symbol tables and so on... Please see TIS documentation for the full description of all sections.

How can we know, how many sections our ELF file contains? What are their names? Their size? Let's see the next code:

```
int nSecNo = pReader->GetSectionsNum();
for ( int i = 0; i < nSecNo; ++i ) {    // For all sections
    const IELFISection* pSec = pReader->GetSection( i );
    std::cout << pSec->GetName() << '' ''
              << pSec->GetSize() << std::endl;
    pSec->Release();
}
```

First, we have got a number of sections. Next, we have received a pointer on IELFISection interface. Using this interface we may access different section's attributes, like it's size, type, flags. address. To get a buffer, that contains section's bytes, we'll use GetData() member function of this interface. Please see IELFISection declaration for a full description of this interface.

2.3 Section readers

When we have got section's data througt GetData() function call, we can manipulate this data according to our wish. But there are special sections that provide information in predefined forms and ELFIO library is ready to help us to process such sections. The library provides a set of section readers that "know"

predefined formats and how to process data. ELFIO.h header file curently defines the next types of readers:

```
enum ReaderType {
    ELFI_STRING,     // Strings reader
    ELFI_SYMBOL,     // Symbol table reader
    ELFI_RELOCATION, // Relocation table reader
    ELFI_NOTE,       // Notes reader
    ELFI_DYNAMIC,    // Dynamic section reader
    ELFI_HASH        // Hash
};
```

Let's see how we can use symbol table reader in our example.
First, we get symbol section:

```
const IELFISection* pSec = pReader->GetSection( ''.symtab'' );
```

Then, we create symbol section reader:

```
IELFISymbolTable* pSymTbl = 0;
pReader->CreateSectionReader( IELFI::ELFI_SYMBOL,
                              pSec,
                              (void**)&pSymTbl );
```

And finally, we use the section reader to process all entries (print operations omitted):

```
std::string   name;
Elf32_Addr    value;
Elf32_Word    size;
unsigned char bind;
unsigned char type;
Elf32_Half    section;
int nSymNo = pSymTbl->GetSymbolNum();
if ( 0 < nSymNo ) {
    for ( int i = 0; i < nSymNo; ++i ) {
        pSymTbl->GetSymbol( i, name, value, size,
                            bind, type, section );
    }
}
pSymTbl->Release();
pSec->Release();
```

2.4 Finalization

All interfaces that we get from ELFIO library should be freed after their use. Each interface has a Release() function. It's not enough to free only high level interface. If one of sections or readers will be held, resources will not be cleared.

While we freed our interfaces immediatly after their use, in this example, we should free only pReader object:

```
pReader->Release();
```

3 ELFDump utility

You may find source code of an ELF dumping utility in examples directory. There you will find more examples of using ELFIO reader interfaces.