**目录**

[**第一章** **sf2ascii.c程序的研究报告** 2](#_Toc468978852)

[**1.1、 sf2ascii .c程序主要函数** 2](#_Toc468978853)

[**1.2、 sf2ascii .c程序主要函数说明** 2](#_Toc468978854)

[1.2.1、 find\_ascii\_record\_type函数 2](#_Toc468978855)

[1.2.2、 find\_ascii\_data\_type函数 3](#_Toc468978856)

[1.2.3、 print\_bit\_array\_comments函数 3](#_Toc468978857)

[1.2.4、 main()主函数： 4](#_Toc468978858)

[**第二章** **C++ 语言程序研究报告** 10](#_Toc468978859)

[**2.1、 dbGDS2.h** 10](#_Toc468978860)

[**2.2、 dbGDS2.cc** 11](#_Toc468978861)

[**2.3、 dbGDS2ReaderBase.h** 12](#_Toc468978862)

[**2.4、 dbGDS2ReaderBase.cc** 14](#_Toc468978863)

[**2.5、 DbGDS2Reader.h** 16](#_Toc468978864)

[**2.6、 dbGDS2Reader.cc** 17](#_Toc468978865)

1. **sf2ascii.c程序的研究报告**
   1. **sf2ascii .c程序主要函数**

sf2ascii .c程序，除了main()主函数外，一共还有三个函数供main()来调用。

void find\_ascii\_record\_type (int numeric, char \*record\_name,

char \*record\_description, int \*expected\_data\_type);

void find\_ascii\_data\_type (int numeric, char \*data\_name, char \*data\_description);

void print\_bit\_array\_comments (char \*ascii\_record\_type,

int bit\_array, char \*indent\_string);

* 1. **sf2ascii .c程序主要函数说明**
     1. find\_ascii\_record\_type函数

void find\_ascii\_record\_type (int numeric, char \*record\_name,

char \*record\_description, int \*expected\_data\_type);

参数：Numeric：record类型都对应一个唯一的16进制的数（如0x00对应HEADER这种类型的record）;

record\_name：保存record类型名（如HEADER）;

record\_description：保存record类型的描述语言;

expected\_data\_type：record类型中的数据的输出形式（0x02表示以INTEGER\_2型格式输出数据）

函数功能：每种record类型对应一个唯一的16进制的数（如0x00对应HEADER类型的record），根据传入进来的numeric,可以找到相应的record所对应的16进制的数，并赋予一个名字（如HEADER）保存的record\_name指向字符串的指针变量中，对HEADER的描述解释字符串存储在record\_description指针，record中数据输出类型（如0x02表示以INTEGER\_2型格式输出数据）存储在expected\_data\_type指针变量中。

* + 1. find\_ascii\_data\_type函数

void find\_ascii\_data\_type (int numeric, char \*data\_name,

char \*data\_description);

参数 Numeric：record中的数据类型都对应一个唯一的16进制的数（如0x02对应INTEGER\_2）;

data\_name：保存输出数据类型名（如INTEGER\_2）;

data\_description:保存数据类型的描述语言如对INTEGER\_2的解释：

Bit array (2 bytes)）;

函数功能：相应的数据类型的命名和描述（如0x02命名为INTEGER\_2，描述为Bit array (2 bytes)）。

* + 1. print\_bit\_array\_comments函数

void print\_bit\_array\_comments (char \*ascii\_record\_type,

int bit\_array, char \*indent\_string);

参数：ascii\_record\_type：ascii码的record类型（如HEADER）

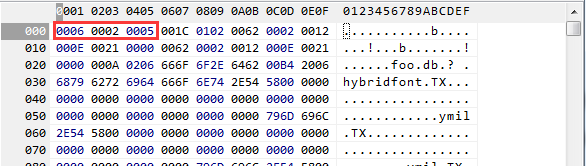
bit\_array：record（如record STRANS对应的16进制 0x1A01）两个字节对应的值

indent\_string：缩进标识。

函数功能：特殊的record（如STRANS）的某些bit有特殊的意义，这个函数指明这些bit的含义。

* + 1. main()主函数：

以下部分会以第一个record作为例子来说明。

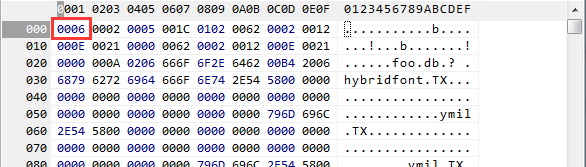


#define NO\_SPACES\_TO\_INDENT 2

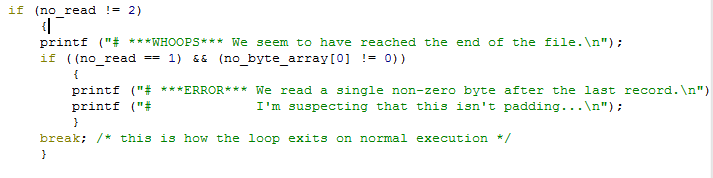
文本缩进两个字节而定义的一个宏

no\_read = read (0, no\_byte\_array, 2);

读取2个字节到no\_byte\_array数组中，返回值为实际写入到no\_byte\_array的字节数，传给no\_read。如

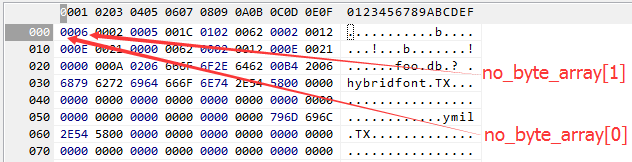


No\_byte\_array中存储了0x0006两个字节,no\_read为实际读取的字节数，在此no\_read=2。读取的字节数no\_read不为偶数的情况，给定的错误提示如下



no\_bytes = no\_byte\_array[0] \* 256 + no\_byte\_array[1];

no\_bytes：record的总的字节数。no\_bytes=0\*256+6=6

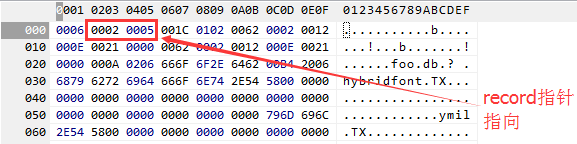


record = (unsigned char \*) malloc (no\_bytes - 2);

分配no\_bytes - 2个字节的空间,在此为4

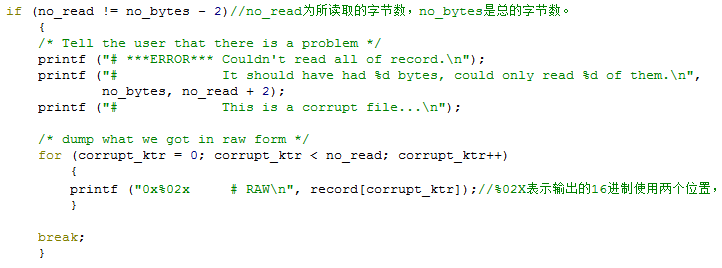
no\_read = read (0, record, no\_bytes - 2);

将这no\_bytes-2（即0002，0005）个字节存储到record中。0005



所实际存储的字节数不等于的总的字节数的错误提示。如0x0002，

0x0005，如果只存储了0x0002。no\_read=2不等于no\_bytes – 2=4，便会出现以上的错误提示如下。



record\_type = record[0];

find\_ascii\_record\_type(record\_type,ascii\_record\_type,ascii\_record\_description,&expected\_data\_type);

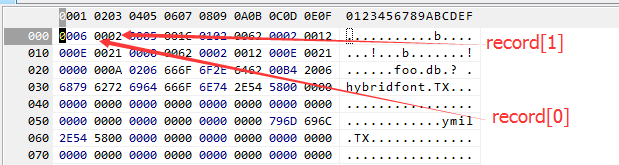
获取record类型，调用find\_ascii\_record\_type函数。

data\_type = record[1];

find\_ascii\_data\_type (data\_type, ascii\_data\_type, ascii\_data\_type\_description);

获取数据类型，调用find\_ascii\_data\_type函数。

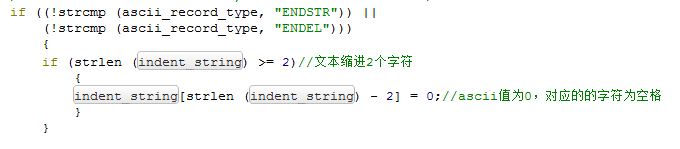
以上分别对应0x00，0x02



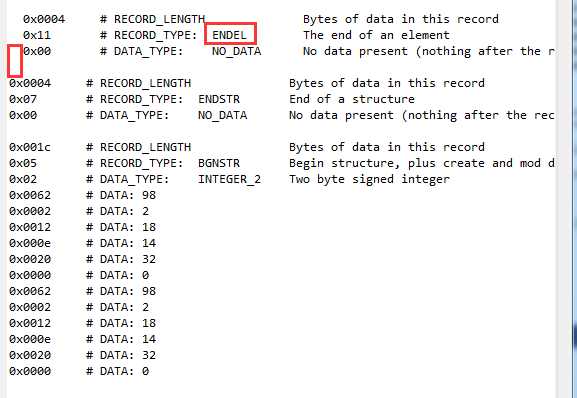
这两个函数分别给record中的RECORD\_TYPE,DATA\_TYPE命名和描述。

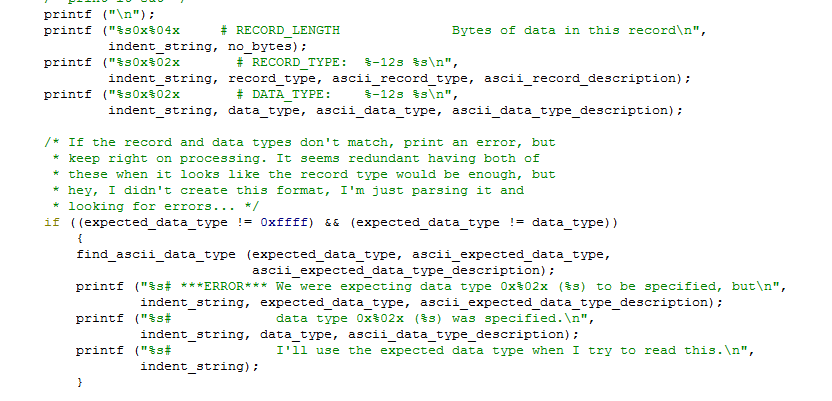
Expected\_data\_type会得到一个值，为后面相应的数据输出做准备。

如expected\_data\_type=0x02时，便会显示对应类型的数据5.

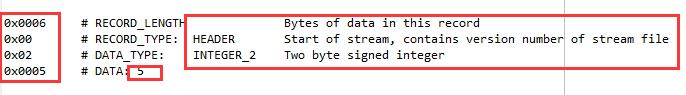


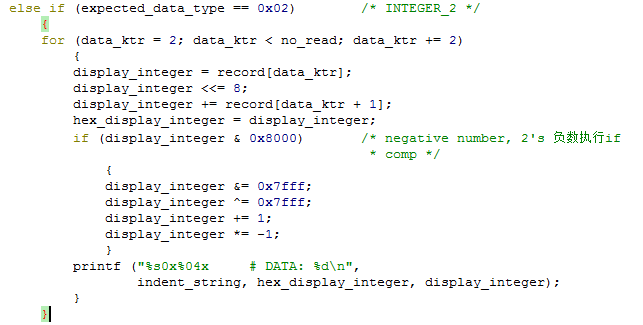
当record为ENDSTR，ENDEL时，文本减少缩进两个字节如图



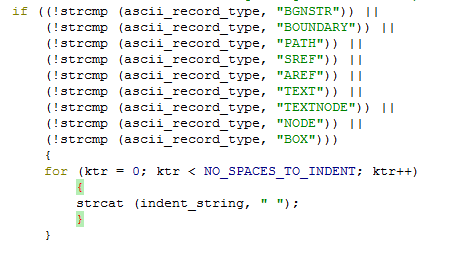


这是一些输出信息（如下图），以及一些错误提示。

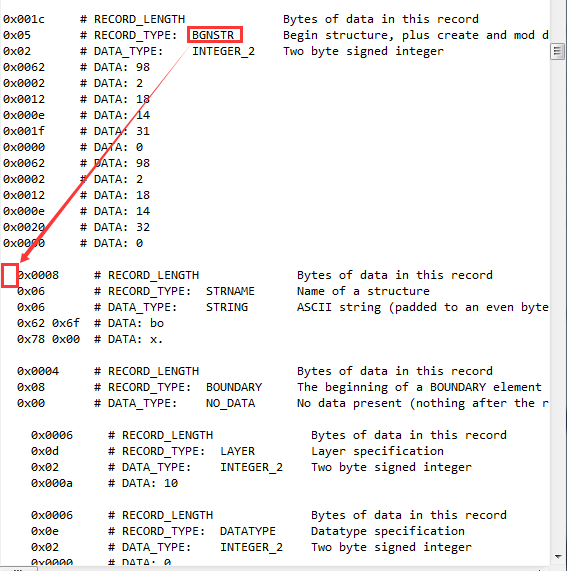




这是INTEGER\_2型数据的输出形式。其它的数据输出也是类似。



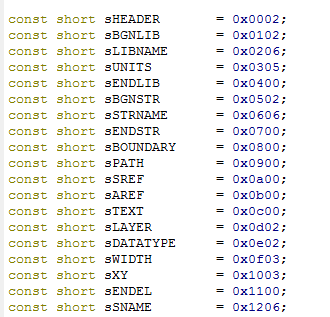
如果record是以上几种情况，将两个空格写入到indent\_string[]，以实现文本缩进两个字节，如下图



1. **C++ 语言程序研究报告**
   1. **dbGDS2.h**

dbGDS2.h文件中完成了两件事

* 给特定的标识符赋对应的值。



* 依据给定的要求，完成2字节，4字节的高低位转化或默认设置。

inline void gds2h (int16\_t &s)函数

参数：

s:short型数据。

函数功能：

defined(\_\_BYTE\_ORDER\_\_) && \_\_BYTE\_ORDER\_\_==\_\_ORDER\_LITTLE\_ENDIAN\_\_

低位优先成立，交换s的字节顺序

defined(\_\_BYTE\_ORDER\_\_) && \_\_BYTE\_ORDER\_\_==\_\_ORDER\_BIG\_ENDIAN\_\_

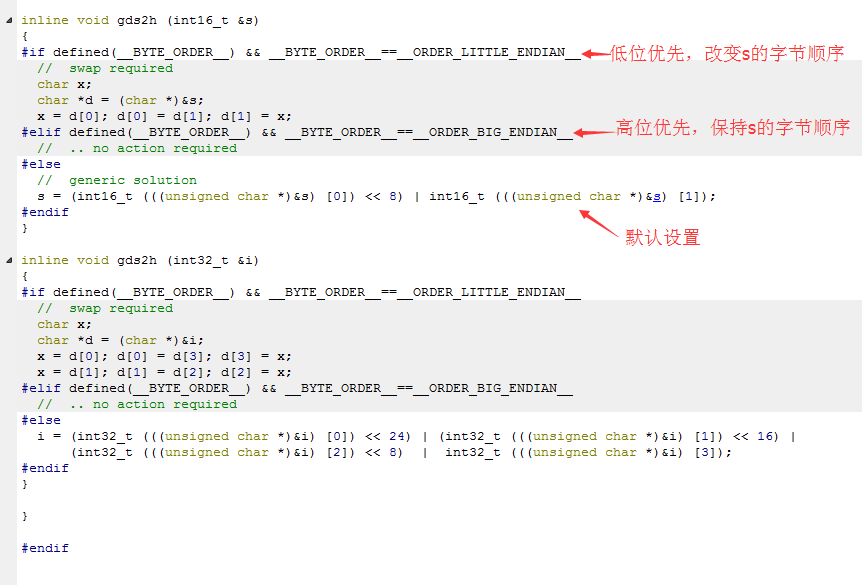
高位优先成立，保持s的字节顺序不变

默认的方式为

s = (int16\_t (((unsigned char \*)&s) [0]) << 8) |

int16\_t (((unsigned char \*)&s) [1])

inline void gds2h (int32\_t &i) 函数作用参考上面。



* 1. **dbGDS2.cc**

dbGDS2.cc完成了对GDS2格式的申明

class GDS2FormatDeclaration : public db::StreamFormatDeclaration

GDS2FormatDeclaration的格式申明类继承于StreamFormatDeclaration类

GDS2FormatDeclaration类主要实现了这四个函数的重载，分别对应读取文件，写入文件，能否读取，能否写入功能。

virtual ReaderBase \*create\_reader (tl::InputStream &s) const

virtual WriterBase \*create\_writer () const

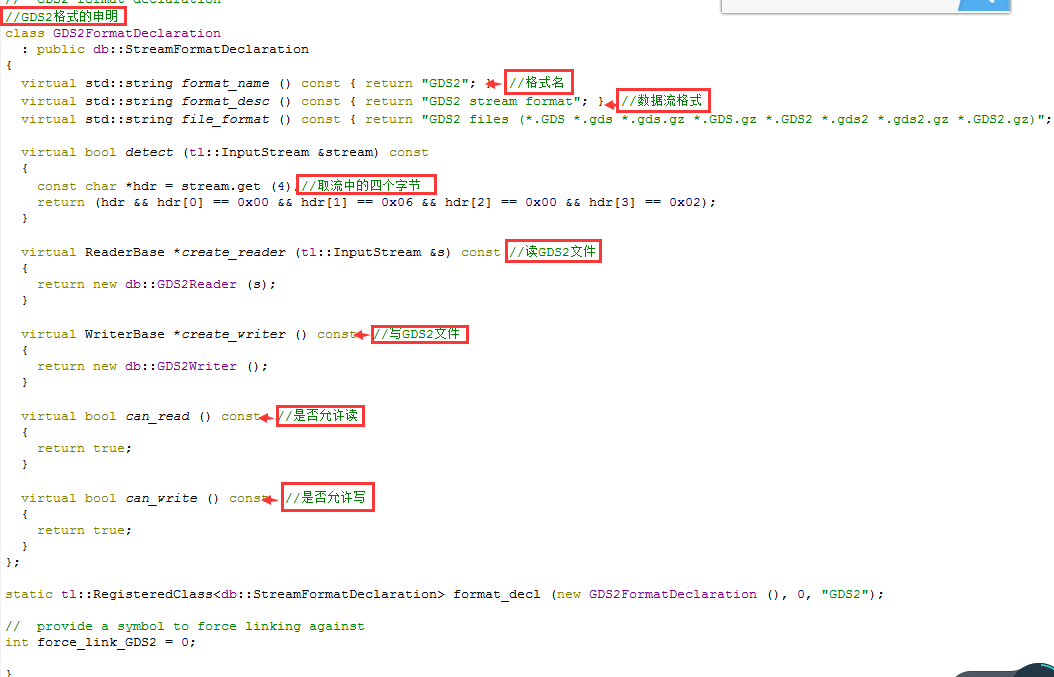
virtual bool can\_read () const

virtual bool can\_write () const

static tl::RegisteredClass<db::StreamFormatDeclaration>

format\_decl (new GDS2FormatDeclaration (), 0, "GDS2");

此语句应该是一个GDS2FormatDeclaration类注册，不太清楚这一个语句



* 1. **dbGDS2ReaderBase.h**

定义了一个用于存储坐标的结构体 struct GDS2XY{}

定义了一个类这

class KLAYOUT\_DLL GDS2ReaderBase: public ReaderBase{}

类中有以下几个函数申明

const std::string &libname () const { return m\_libname; } 获取库名

const LayerMap &basic\_read (db::Layout &layout, const LoadLayoutOptions &options, bool allow\_multi\_xy\_records, unsigned int box\_mode);

const tl::string &cellname () const { return m\_cellname; } 获取单元名

主要介绍下面的这个函数

const LayerMap &basic\_read (db::Layout &layout, const LoadLayoutOptions &options, bool allow\_multi\_xy\_records, unsigned int box\_mode);

参数：Layout：要写入的布局对象

Options:读取选项（不使用特定格式的选项）

allow\_multi\_xy\_records：如果是真的，试图检查边界元素XY记录

box\_mode：对BOX记录而言 (0: ignore, 1: as rectangles, 2: as boundaries, 3: error)

返回值 LayerMap: LayerMap对象告诉哪一层被加载

函数功能：调用这个函数后，将会读取数据，从返回值中知道哪一层被加载。

void read\_path (db::Layout &layout, db::Cell &cell); 读取path

void read\_text (db::Layout &layout, db::Cell &cell); 读取text

void read\_box (db::Layout &layout, db::Cell &cell); 读取box

void read\_ref (db::Layout &layout, db::Cell &cell, bool array, tl::vector<db::CellInstArray> &instances, tl::vector<db::CellInstArrayWithProperties> &insts\_wp); 读取引用

std::pair <bool, unsigned int> open\_dl (db::Layout &layout, const LDPair &dl, bool create);

std::pair <bool,db::properties\_id\_type> finish\_element

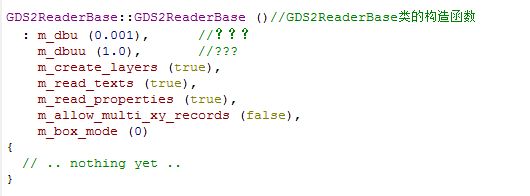
(db::PropertiesRepository &rep);一个元素的结束

void finish\_element ();一个元素的结束

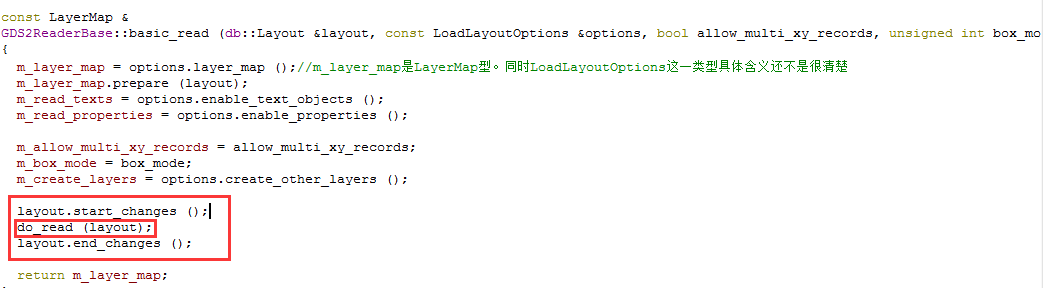
* 1. **dbGDS2ReaderBase.cc**



这个类不是很懂。



后面红色部分是变量？为何类继承变量了？



从这个函数当中do\_read()函数开始便读取数据流了。



这两个函数虽然都是结束一个元素，具体的区别还有知道？

接下来是













* 1. **DbGDS2Reader.h**

class KLAYOUT\_DLL GDS2Reader: public GDS2ReaderBase

GDS2Reader类中以下两个函数的作用参考前文

virtual const LayerMap &read (db::Layout &layout, const LoadLayoutOptions &options);

virtual const LayerMap &read (db::Layout &layout);

类中其它函数

virtual const char \*get\_string (); 以字符串的形式读取数据

virtual void get\_string (tl::string &s) const;

virtual int get\_int (); 以int型读取数据

virtual short get\_short ();以short型读取数据

virtual unsigned short get\_ushort ();以unsigned short型读取数据

virtual double get\_double ();以double型读取数据

virtual short get\_record ();获取record对应的值

virtual void unget\_record (short rec\_id); 以int型读取数据

virtual void get\_time (unsigned int \*mod\_time, unsigned int \*access\_time);

将数据转换成修改时间，以及访问时间

virtual GDS2XY \*get\_xy\_data (unsigned int &length); 获取坐标数据

virtual void progress\_checkpoint ();数据流的当前位置

* 1. **dbGDS2Reader.cc**

DbGDS2Reader.h中申明函数的实现。