一、

在解析RM文件时我们首先需要了解RM文件的格式。在下面的内容中将介绍RM文件的格式。

RM文件格式（RealMedia File Format 简称RMFF），文件单元使用四字符编码。编码是32位的。

RMF的基本单元是chunk。是数据的逻辑单元。

每个chunk包含：

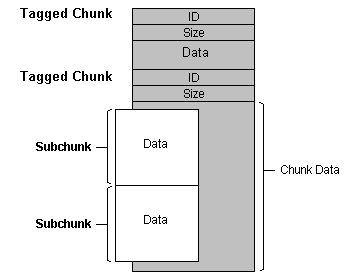
1.四字符编码的单元标识符；

2.每个chunk的32位的数据；

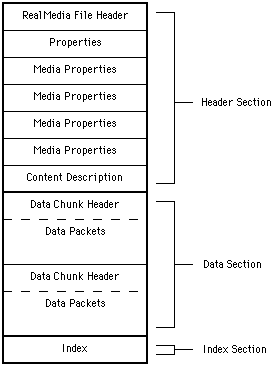
3.未识别的块。

RMFF的标签

RMFF的标签由一个头单元，一个数据单元和一个引导单元组成。



在头部中包括一下单元：RM的文件头，属性头，媒体属性头，内容描述头。



RM文件头：

每个RM文件都是以RM文件头开始的。

RealMedia\_File\_Header

{

UINT32 object\_id;

UINT32 size;

UINT16 object\_version

;

if ((object\_version == 0) || (object\_version == 1))

{

UINT32 file\_version;

UINT32 num\_headers;

}

}

RMFHeader的结构体

由于RMFH的内容要根据RMFF的版本来改变，所以头的结构体有一个版本信息为了扩展方便。

object\_id:  RMF唯一的ID号。是32位的无符号整型。相当于文件的标识符。

size:    RM头的大小。32位的无符号整型。

object\_version:   RMFH对象的版本。16位的无符号整型。所有文件依照这个规定有一个object\_version是0或1.

file\_version: 0或1.

num\_headers:   在头部中头的数量。32位无符号整型。

属性头：

描述RMF的一般媒体属性。

RM系统成员通过这个对象来配置如何处理RMF中或者流中的数据。在RMF中只有一个属性头。

结构体如下：

Properties

{

UINT32 object\_id;

UINT32 size;

UINT16 object\_version;

if (object\_version == 0)

{

UINT32 max\_bit\_rate;

UINT32 avg\_bit\_rate;

UINT32 max\_packet\_size;

UINT32 avg\_packet\_size;

UINT32 num\_packets;

UINT32 duration;

UINT32 preroll;

UINT32 index\_offset;

UINT32 data\_offset;

UINT16 num\_streams;

UINT16 flags;

}

}

object\_id:  属性头的唯一ID，相当于标识符。（PROP）

size：  属性头的大小。字节数

object\_version:  0或1.RMFH对象的版本。

当object\_version==0时由如下成员。

max\_bit\_rate: 网络传输这个文件所需要的最大的比特率。

avg\_bit\_rate:  网络传输这个文件所需要的平均比特率。

max\_packet\_size:  媒体数据包最大的大小。

avg\_packet\_size:  媒体数据包的平均大小。

num\_packets:  媒体信息中数据包数。

duration:  文件的位置。以毫秒为单位。

preroll:  在回放前缓冲区的毫秒数。

index\_offset:  从文件开头到索引头对象的字节数。

data\_offset:  从文件开头到数据单元的开头的字节数。

num\_streams:  在主要的头单元中媒体属性头的数量。

flags:   位屏蔽的一些这些文件的信息。

媒体属性头（Media Properties Header）

描述RM文件的每个流中特殊的媒体属性。作用和属性头差不多。

结构体描述如下：

Media\_Properties

{

UINT32 object\_id;

UINT32 size;

UINT16 object\_version;

if (object\_version == 0)

{

UINT16 stream\_number;

UINT32 max\_bit\_rate;

UINT32 avg\_bit\_rate;

UINT32 max\_packet\_size;

UINT32 avg\_packet\_size;

UINT32 start\_time;

UINT32 preroll;

UINT32 duration;

UINT8 stream\_name\_size;

UINT8[stream\_name\_size] stream\_name;

UINT8 mime\_type\_size;

UINT8[mime\_type\_size] mime\_type;

UINT32 type\_specific\_len;

UINT8[type\_specific\_len] type\_specific\_data;

}

}

object\_id:  该结构体的实例化ID（MDPR），标识符。

Size: 该结构体的大小。

object\_version:  媒体属性头的版本。

stream\_number:  标识一个物理流。是一个唯一的值。每个数据包都属于一个物理流，这个物理流和数据包的stream\_number相同。

start\_time:  在物理流中每个数据包的添加时间戳的开始时间。

stream\_name:流的名字

stream\_name\_size:stream\_name的字节数。

mime\_type:  一个非唯一索引的MIME风格的类型字符串是关联流的数据。

mime\_type\_size:mime\_type的字节数。

type\_specific\_data:  通常被用于数据类型描绘器的实例化。为处理物理流。

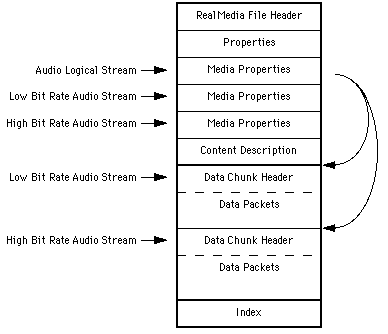
type\_specific\_len:    type\_specific\_data的字节数。

Logical Stream Organization

物理流组成逻辑流。逻辑流包含一下信息：

1.标识符：哪些物理流一起组成逻辑流

2.标识逻辑流属性的名字、值、属性。



逻辑流的结构体：

LogicalStream

{

ULONG32 size;

UINT16 object\_version;

if (object\_version == 0)

{

UINT16 num\_physical\_streams;

UINT16 physical\_stream\_numbers[num\_physical\_streams];

ULONG32 data\_offsets[num\_physical\_streams];

UINT16 num\_rules;

UINT16 rule\_to\_physical\_stream\_number\_map[num\_rules];

UINT16 num\_properties;

NameValueProperty properties[num\_properties];

}

};

size：结构体的大小（字节）

object\_version:  逻辑流结构体的版本。

如果object\_version==0；

则有如下元素：

num\_physical\_streams:  构成逻辑流的物理流数。

physical\_stream\_numbers[]:  存储num\_physical\_streams的数组。这个成眼是可变的。

data\_offsets[]:   标志每个物理流的数据单元开始位置。

num\_rules:   逻辑流的ASM rulse 的数量。//ASM:制动存储管理

rule\_to\_physical\_stream\_map[] :  物理流对应的每个规则的存储列表。

num\_properties:  NameValueProperty结构体的数量。

properties[]:  NameValueProperty结构体的存储列表。

**NameValueProperty Structure**

NameValueProperty

{

ULONG32 size;

UINT16 object\_version;

if (object\_version == 0)

{

UINT8 name\_length;

UINT8 name[namd\_length];

INT32 type;

UINT16 value\_length;

UINT8 value\_data[value\_length];

}

}

size: NameValueProperty结构体的大小（字节数）。

object\_version:  结构体的版本。

如果object\_version==0;则有如下元素。

name\_length: 名字的大小。

name[]： 名字信息。

type:  数据的类型。

value\_length: 数据信息的长度。

value\_data[]:  数据。

Content Description Header

Content\_Description

{

UINT32 object\_id;

UINT32 size;

UINT16 object\_version

;

if (object\_version == 0)

{

UINT16 title\_len;

UINT8[title\_len] title;

UINT16 author\_len;

UINT8[author\_len] author;

UINT16 copyright\_len;

UINT8[copyright\_len] copyright;

UINT16 comment\_len;

UINT8[comment\_len] comment;

}

}

展示文件的信息。是文本信息。包括文件的title.author.copyright和comments information.

二、

前面一章中介绍了RM文件的头。由于是初步理解所以文章比较凌乱。  
  
这一章节介绍RM文件的数据单元。  
  
RM文件的数据单元包括一个数据单元头。数据单元头描述数据单元的内容。  
  
接着就是一系列交叉存取的媒体信息包。  
  
Data Chunk Header  
  
标记数据块的开始位置。一般的RM文件通常只有一个数据块。很大的文件，可能由多个数据块。  
  
Data\_Chunk\_Header  
{  
  UINT32     object\_id;  
  UINT32     size;  
  UINT16      object\_version;  
  
  if (object\_version == 0)  
  {  
    UINT32    num\_packets;   
    UINT32    next\_data\_header;  
  }  
}  
  
object\_id: 数据块头的唯一ID（DATA）。  
  
size:  数据块的大小。  
  
object\_version:数据块头对象的版本.  
  
num\_packets: 数据块中包的数量。  
  
next\_data\_header:下一个数据块的开始位置。  
  
Data Packet Header  
  
Media\_Packet\_Header  
{  
  UINT16                object\_version;  
  
  if ((object\_version == 0) || (object\_version == 1))  
  {  
    UINT16        length;  
    UINT16        stream\_number;  
    UINT32        timestamp;  
    if (object\_version == 0)  
    {  
      UINT8        packet\_group;  
      UINT8        flags;  
    }  
    else if (object\_version == 1)  
    {  
      UINT16        asm\_rule;  
      UINT8          asm\_flags;  
    }  
  
    UINT8[length]        data;  
  }  
  else  
  {  
    StreamDone();  
  }

}

object\_version :  媒体数据包头对象的版本号。

length:   数据包的字节数。

stream\_number:  用于连接数据包和与之有关的媒体属性头。

timeStamp:  时间戳。

packet\_group: 属于哪个组。

flags:  描述这个数据包的属性。

asm\_rule: 将ASM rule赋值给这个数据包。

asm\_flags:  包含HX\_flags.指定流交换指针。

data:  程序指定的媒体数据。

三、

上面两章分别介绍了RM文件格式的头单元和数据单元的具体信息。

这一章主要学习的RM的引导单元。

RM文件的引导单元主要由两个部分组成。

Index Chunk Header:描述引导单元的内容。这个是引导单元的头。

接着是一系列的引导标签。

引导单元的头。

Index\_Chunk\_Header

{

u\_int32 object\_id;

u\_int32 size;

u\_int16 object\_version

;

if (object\_version == 0)

{

u\_int32 num\_indices;

u\_int16 stream\_number;

u\_int32 next\_index\_header;

}

}

object\_id:  index chunk header的唯一标识符。（“INDX”）

size:  引导单元的字节数。

object\_version:  引导块头部对象的版本号。

num\_indices:  在引导块中引导标签的数量。

stream\_number:在引导块中引导标签所对应的流。

next\_index\_header: 从文件的开始到下一个引导的距离。

index Record

在RM文件中的引导单元中有一系列的index record对象。每个对象包含 了快速找到一个特定物理流的时间戳信息。

IndexRecord

{

UINT16 object\_version;

if (object\_version == 0)

{

u\_int32 timestamp;

u\_int32 offset;

u\_int32 packet\_count\_for\_this\_packet;

}

}

object\_version:引导对象的版本号。

timestamp： 时间戳。

offset： 从文件的开始到数据包被找到的距离。

packet\_count\_for\_this\_packet:  在引导的数据包中数据包的数量。

四、

在RM文件的元数据单元包括一个标签、一个描述媒体文件元数据属性的集合等信息。

元数据单元的头（Metadata Section Header）

标识元数据单元的开始。一个RM文件中由一个元数据单元。预计是在文件的最后。

MetadataSectionHeader

{

u\_int32 object\_id;

u\_int32 size;

}

object\_id:元数据单元头的唯一标识符（RMMD）

size: 整个元数据的字节数。

元数据标签（Metadata Tag）

RM文件的元数据标签包含了一系列的属性。

MetadataTag

{

u\_int32 object\_id;

u\_int32 object\_version;

u\_int8[] properties;

}

object\_id: 元数据标签的唯一标签（“RJMD”）

object\_version: 元数据标签的版本号。

properties[ ]: 组成元数据标签的元数据属性。

元数据属性（Metadata Property Structure）

MetadataProperty

{

u\_int32 size;

u\_int32 type;

u\_int32 flags;

u\_int32 value\_offset;

u\_int32 subproperties\_offset;

u\_int32 num\_subproperties;

u\_int32 name\_length;

u\_int8[name\_length] name;

u\_int32 value\_length;

u\_int8[value\_length] value;

PropListEntry[num\_subproperties] subproperties\_list;

MetadataProperty[num\_subproperties] subproperties;

}

size: 元数据属性的字节数。

type: 数据值的类型。

flags: 描述属性。可以是只读（MPT\_READONLY）、私有（MPT\_PRIVATE）、描述符（MPT\_TYPE\_DESCRIPTOR）。