编　　制	沈胜文
审　　核	
批　　准	
实施责任人	

Amendment history 修改历史记录

版本号	修改说明	修改批准人	修改人	日期	签收人
101	创建文档		沈胜文	2008-3-16	
102	修改		沈胜文	2008-4-11	
103	修改		王艳君	2008-5-5	

Table Of Contents 目录

1. Introduction 简介

1.1. Objective 编写目的

本文档是在《XML 解析概要设计》的基础上，就 XML 解析时所需要的数据结构及相应操作，进行的详细设计说明。因此本文档不涉及 XML 解析部分，仅仅涉及相关数据结构的创建、添加及删除操作；

本文档将尽可能详尽地说明相关数据结构的设计与开发，但是设计与开发不相符之处，需讨论决定，并且修改本文档；

最终设计以代码为准；

1.2. Background 背景

本程序是法电自动下载（MuFTAD）软件项目中的一部分，负责 XML（信令）内容相关的相关数据结构的操作；

设计的出发　点：使用 MuFTAD 项目中涉及的数据结构独立于 XML 解析函数，方便相关操作的更新与完善；
本软件的提出者：沈胜文
本软件的开发者：沈胜文
本软件的用　户：MuFTAD 项目；

1.3. Terms and Abbreviation 术语与缩写解释

Terms&Abbreviation 术语&缩写	Description 解释
XML	XML即可扩展标记语言（eXtensible Markup Language）。标记是指计算机所能理解的信息符号，通过此种标记，　计算机之间可以处理包含各种信息的文章等。
MuFTAD	软件名称 Microunit France-Telecom Auto-Download

FDDF	全称： File Downloading Description File (FDDF, fddfXML) FDDF keeps finfXML and downloading information.

1.4. Reference Material 参考资料

MuFTAD 需求说明书

2. Rules 规则

2.1. Name Rules 命名规则

相关数据结构的操作仅服务于 MuFTAD 项目。该程序中，所有的函数均以 Mu_的形式开头，而不是 MuFTAD_。因此在该程序内，所有的函数形如：Mu_XXXX()；

2.2. Illuminate 说明

针对每个程序，都必须注明其开发目的，开发者，开发时间，等等。以下字段必须被包含于程序的开头部分。

```
/*****************************************************************
*                    ====Microunit Techonogy Co,.LTD.====
* File Name:
*
*          XMLParse.c
*
* Description:
*
*          This file get the file name, which store the XML Contents.
*          The Functions Open the file, parse it, then return the Information we need.
*
* Revision History:
*
*          10-3-2008 ver1.0
*
*Author:
```

```
*
*          ssw    (fzqing@gmail.com)
*
*                          ***PROTECTED BY COPYRIGHT***
**********************************************************************/
```

2.3. Note Rules 注释规则

程序中的各个函数均需要明确注释其功能，并能简要描述其实现，及注意点。特别应该注意的是：在描述时，应该详细包括对锁，输入和输出进行详细说明。

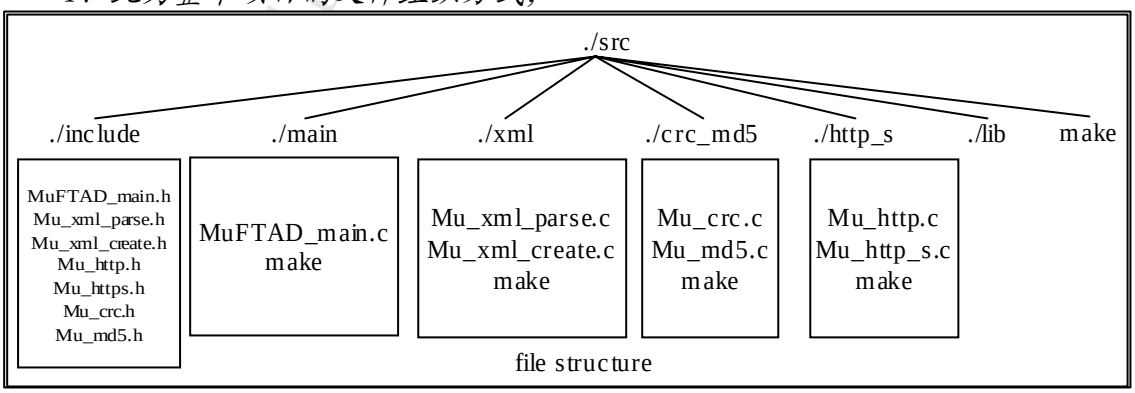
可参考模板

```
/**********************************************************************
*Description:
*          This Function is Parse the XML, return the Informations to caller use the
*          Value pointer;
*Input:
*          filename: the file name , which stored the XML contents
*Output:
*          Pointer: which is a pointer, point to the buffer stored the Informations\
*LOCK:
*          NONE
*Modify:
*          ssw (fzqing@gmail.com    10-3-2008)
**********************************************************************/
```

2.4. File Structure 文件结构

[注意]:

1、此为整个项目的文件组织方式;



file structure

./src/include：文件夹，包含该项目中的所有头文件；

./src/main：文件夹，包含所有按法电《do3c00_SoftProtocol_0.1.0_RC1》流程所开发的程序；

./src/xml：文件夹，包含项目中所需要的 xml 处理库函数源代码；

./src/crc_md5：文件夹，包含项目中所需要的校验函数源代码，包括 CRC 和 MD5 校验代码；

./src/http_s：文件夹，包含项目中所需要的与服务器交互的方式，包括 HTTP(s) GET、POST 方式；

./src/lib：文件夹，用于存储编译所生成的 xml、http 和 https、crc/md5 库。软件编译连接时使用该文件夹下的库；

./src/make：文件，总的编译入口；

[注意]:

1、各对应文件夹下的源文件按需要添加，但是所作修改必须对 makefile 文件作相应的修改，以正确编译;

3. Structure Of Routines 程序结构

3.1. Overview 总述

待开发的各程序之间是相互独立的，它们为项目 MuFTAD 中的其他部分服务，重点在相关数据结构的创建、添加及删除。

本项目中将会涉及的数据结构，及需要对其进行的操作均放于本文件中进行说明。

本文档主要分二部分：

一、相关数据结构的声明；

[注意]:

1、声明的数据结构可能与实际应用有所偏差，根据实际开发添加或是删减结构成员;

二、对数据结构的创建、添加与删除；

[注意]:

1、第一部分涉及到的所有数据结构均应提供上述三类操作;

3.2. Routines List 函数列表

```
int Mu_FddfStatusHeadCreate(FddfStatusHeadPtr *Ptr);
int Mu_FddfStatusNodeCreate(FddfStatusNodePtr *Ptr);
int Mu_FddfStatusHeadInit(FddfStatusHeadPtr Ptr);
int Mu_FddfStatusNodeInit(FddfStatusNodePtr Ptr);
int Mu_FddfStatusInsertNode(
       FddfStatusHeadPtr Head, FddfStatusNodePtr Node);
int Mu_FddfStatusNodeDelete(FddfStatusNodePtr Node);
int Mu_FddfStatusNodeStrncpy(FddfStatusNodePtr Node,
          const char *string, const char *name);

int Mu_DirStatusHeadCreate(DirStatusHeadPtr *Ptr);
int Mu_DirStatusNodeCreate(DirStatusNodePtr *Ptr);
int Mu_DirStatusAddrCreate(DirDownloadAddrPtr *Ptr);
int Mu_DirStatusHeadInit(DirStatusHeadPtr Ptr);
int Mu_DirStatusNodeInit(DirStatusNodePtr Ptr);
int Mu_DirDownloadAddrInit(DirDownloadAddrPtr Ptr);
int Mu_DirDownloadAddrInsert(
          DirStatusNodePtr Node, DirDownloadAddrPtr addr);
int Mu_DirDownloadAddrDeleteone(DirStatusNodePtr Node);
int Mu_DirDownloadAddrDeleteList(DirStatusNodePtr Node);
int Mu_DirStatusNodeDelete(DirStatusHeadPtr Head,
             DirStatusNodePtr Node);
int Mu_DirStatusNodeStrncpy(DirStatusNodePtr Node,
             const char *string, const char *name) ;
int Mu_DirDownloadAddrStrncpy(DirDownloadAddrPtr Node,
             char *string) ;

int Mu_ServiceInfoCreate(ServiceInfoPtr *Ptr);
int Mu_ServerNodeCreate(ServerNodePtr *Node);
int Mu_ServiceInfoInit(ServiceInfoPtr Ptr);
int Mu_ServerNodeInit(ServerNodePtr Ptr);
int Mu_ServiceInfoStrncpy(ServiceInfoPtr Ptr,
             const char *string, const char *name);
int Mu_ServerNodeStrncpy(ServerNodePtr Ptr,
             const char *string, const char *name);

int Mu_FddfListHeadCreate(FddfListHeadPtr *Ptr);
int Mu_FddfListNodeCreate(FddfListNodePtr *Ptr);
```

```
int Mu_FddfListHeadInit(FddfListHead Ptr);
int Mu_FddfListNodeInit(FddfListNodePtr Ptr);
int Mu_FddfListNodeInsert(FddfListHeadPtr Head,
                FddfListNodePtr Node);
int Mu_FddfListNodeDelete(FddfListHeadPtr Head);
int Mu_FddfLisitHeadStrncpy(FddfListHeadPtr Head,
                    const char *sring, const char *name);
int Mu_FddfListNodeStrncpy(FddfListNodePtr Node,
                    const char *string, const char *name);

int Mu_FddfFileCreate(FddfFilePtr *Ptr);
int Mu_SegmentHeadCreate(SegmentHeadPtr *Ptr);
int Mu_SegmentNodeCreate(SegmentNodePtr *Ptr);
int Mu_FddfFileInit(FddfFilePtr Ptr);
int Mu_SegmentHeadInit(SegmentHeadPtr Ptr);
int Mu_SegmentNodeInit(SegmentNodePtr Ptr);
int Mu_SegmentNodeInsert(SegmentHeadPtr Head, SegmentNodePtr Node);
int Mu_SegmentNodeDelete(SegmentHeadPtr head);
int Mu_FddfFileStrncpy(FddfFilePtr Ptr,
             const char *string, const char *name);
int Mu_SegmentNodeStrncpy(SegmentNodePtr Ptr,
             const char *string, const char *name);

int Mu_EyewearADPCreate(EyewearADPPtr *Ptr);
int Mu_EyewearADPInit(EyewearADPPtr Ptr);
int Mu_EyewearADPStrncpy(EyewearADPPtr Ptr,
             const char *string, const char *name);

int Mu_UpdateInfoCreate(UpdateInfoPtr *Ptr);
int Mu_UpdateInfoStrncpy(UpdateInfoPtr Ptr,
             const char *string, const char *name);

int Mu_DeviceInfoCreate(DeviceInfoPtr *Ptr);
int Mu_DeviceInfoStrncpy(DeviceInfoPtr Ptr,
             const char *string, const char *name);

int Mu_ResumeInfoInit(ResumeInfoPtr Ptr, int num);
int Mu_ResumeInsert(ResumeInfoPtr Ptr, int type,
             const char *name, int *num);

int Mu_FileInfoCreate(FileInfoPtr *Ptr);
```

```
int Mu_FileInfoStrncpy(FileInfoPtr Ptr,
                const char *string, const char *name);
```

4. Struct Definition 结构体定义

4.1. FDDF_Status

4.1.1. Definition

```
typedef struct _FddfStatusNode{
    int type;       //mark the Item type of the FDDF
    char *text;
    int size;
    int newer;      //index new
    int older;      //index older
    char *program;  //ID-program
    char *group;    //ID-group
    char *owner;    //ID-owner
    int priority;   //Download Priority
    struct _FddfStatusNode *next;
}FddfStatusNode, *FddfStatusNodePtr;

typedef struct{
    int type;       //Item Type of the FDDF
    int total;      //total number of this Item Type
    FddfStatusNodePtr next;
}FddfStatusHead, *FddfStatusHeadPtr;

FddfStatusHead ProgramDescriptor[4];
```

4.1.2. Illuminate

该数据结构用于存储 Query_Status 后，服务器返回的 2xx 类节目信息，因为总的 2xx 类型分为四种：201,202,203,204；

但是针对每一类型，又有若干个不同条款（Item），所以，采用四条链表来维持信息，并且四条链表的表头放在一个四元素的数组里；

4.2. DIR_Status

4.2.1. Definition

```
typedef struct _DirDownloadAddr{
    char * addr;            //pointer, point to some heap, store the address
    struct _DirDownloadAddr *next;
}DirDownloadAddr, *DirDownloadAddrPtr;

typedef struct _DirStatusNode{
    int type;           //Item Type of the DIR download
    int authmethod;  //Authorization Method
    int priority;       //Download priority
    char *DownloadID;  //ID-download
    int total;          //total numbers of address
    DirDownloadAddrPtr addrnext;
    DirDownloadAddrPtr addrlast;
    struct _DirStatusNode *next;
}DirStatusNode, *DirStatusNodePtr;

typedef struct{
    int type;       //item type
    int total;      //total number of the type(DirStatusNode)
    DirStatusNodePtr next;
}DirStatusHead, *DirStatusHeadPtr;

DirStatusHead DIRfileDownload[4];
```

4.2.2. Illuminate

该数据结构用于存储 Query_Status 后，服务器返回的 4xx 类节目信息，因为总的 4xx 类型分为四种：401, 402, 403, 404；

但是针对每一类型，又有若干个不同条款（Item），所以，采用四条链表来维持信息，并且四条链表的表头放在一个四元素的数组里；

4.3. Service_Info

4.3.1. Definition

```
typedef struct _ServerAddr{
    char *addr;                    //pointer, point to some heap, store the address
    struct _ServerAddr *next;
} ServerNode, * ServerNodePtr;

typedef struct _ServerHead{
    int total;                  //total number of the address
    ServerAddrPtr next;
    ServerAddrPtr last;
}ServerHead, * ServerHeadPtr;

typedef struct _Apipathlist{
    char *prelogin;
    char *login;
    char *querystatus;
    char *queryfddflist;
    char *updatestatus;
    char *logout;
}ApiPathList, *ApiPathListPtr;

typedef struct _ServiceInfo{
    int Protocolver;
    int stepping;
    ServerHeadPtr server;
    char *lastserver;
    char *ping;
    ApiPathListPtr apipathlist;
}ServiceInfo, *ServiceInfoPtr;
```

4.3.2. Illuminate

该结构用于存储解析 Server Information description XML 文件后的信息，在该结构中，定义了一个链表表头，用以存储<Addr>字段；

4.4. FDDF_List

4.4.1. Definition

```
typedef struct _FddfListNode{
        char *addr;
        struct _FddfListNode *next;
    }FddfListNode, *FddfListNodePtr;

    typedef struct _FddfListHead{
        int type;
        char *text;
        int total;
        int authmethod;
        int priority;
        FddfListNodePtr next;
        FddfListNodePtr last;
    }FddfListHead, *FddfListHeadPtr;
```

4.4.2. Illuminate

该数据结构用于存储解析 Query_FDDF 后， 服务器返回的信息。服务器返回的 FDDF_List 包括在一个条款（ Item）里的所有 FDDF XML 地址信息；

FDDF XML 文件的下载地址用 FddfListNode 节点存储，整个链表有一个头结点：FddfListHead；

该头结点有二个 FddfListNodePtr 指针，一个指向下一节点，另一个指向链表的最后一个结点，此仅方便将新解析得到的地址信息存储到链表的最后；

4.5. FDDF_File

4.5.1. Definition

```
typedef struct _SegmentNode{
    int index;              //index of the segment
    unsigned long size;     //size of the segment
    char *name;             //pointer, point to some buffer,
```

```
                                //store the segment name
        char *md5;              //pointer, point to some buffer,
                                //store the md5
        char *crc;              //pointer, point to some buffer,
                                //store the crc
        struct _SegmnetNode *next;
    }SegmentNode, *SegmentNodePtr;

    typedef struct _SegmentHead{
        int total;
        SegmentNodePtr next;
        SegmentNodePtr last;
    }SegmentHead, *SegmentHeadPtr;

    typedef struct _DatebaseInfo{
        int index;
        char * program;
        char * group;
        char * owner;
    }DatebaseInfo, *DatebaseInfoPtr;

typedef struct _EyewareFDDF{
        char *name;
        unsigned long size;
        char *md5;
        char *crc;
        int type;
        int rate;
        char *displayname;
        char *description;
        char *validbeforedate;
        char *validafterwatch;
        DatebaseInfoPtr datebaseinfo;
        char *NonDefaultServer;
        int Authorizemethod;
        char *serverfilepath;
        int filesegmentnum;
    }FddfFile, *FddfFilePtr;
```

4.5.2. Illuminate

该结构用于存储服务器返回的关于一个节目的描述 XML，该 XML 由二部分组成：

一、节目信息；

二、节目的分段信息；

因此，对于上述二部分，分别由二个结构来存储信息：

一、SegmentHead：存储分段信息，该部分信息由一个链表存储，链表有一个表头结点；

表头结点中有二个指针，一个指向链表的下一分段节点 SegmentNode，另一个指向链表的最后一个分段节点；

头结点中的二个指针是方便向链表表尾中加入新的节点；

二、节目的描述信息，由 FddfFile 存储；

对节目的描述字段，有些需要截断，比如：DisplayName，Desciption 等；

[注意]:

1、这部分的截断由解析函数完成，本文档内所涉及函数不提供截断操作;

4.6. Eyewear_ADP

4.6.1. Definition

```
typedef struct _PreLogin{
    int encrptionmethod;        // Device ID encryption method
    char *challenge;        // Challenge Data for encryption
}Prelogin, *PreloginPtr;

typedef struct _DeviceInfo{
    char *DeviceIDInt;
    char *DeivceIDApp;
    int devicestatus;
    int trigresource;
    char *SRVDateTime;
    int firewareversion;
    int hardwareversion;
    int serverinfostep;
}DeviceInfo, *DeviceInfoPtr;
```

```
typedef struct _Coversation{
    char *ConversationID;
    char *AuthUsername;
    char *AuthPassword;
}Conversation, *CoversatuonPtr;

typedef struct _EyewearADP{
    int Protocolver;
    int ProtocolOPC;
    PreloginPtr prelogin;
    DeviceInfoPtr device_info;
    ConversationPtr conversation;
}EyewearADP, *EyewearADPPtr;

typedef struct    _SRVDeviceInfo{
int type;      //Item Payload type
int method;      //Authorization Method
char * ServerInfo;    //File Path and Name (@Server)
int version1;      //File Stepping   or Firmware version
int version2;      //Reserved number or Hardware Version
char * Md5Num;      //Checksum (MD5)
}SRV_SvrInf,SRV_FirmSF;
typedef SRV_SvrInf *SRV_SvrInfPtr;
typedef SRV_FirmSF *SRV_FirmSFPtr;
```

4.6.2. Illuminate

该结构用于存储解析服务器与客户端交互的 CMD XML 文件中信息；

CMD XML 可以分为二部分：
一、固定部分，由 <Protocol_Ver>, <Protocol_OPC>, <PreLogin>,<DeviceInfo>,<Conversation>字段组成；
二、可变部分，由<payload>字段组成；

因此，对 CMD XML 的解析也需要由二个结构来存储，本结构只对固定部分进行存储，可变部分依具体情况解析；

4.7. Update_Info

4.7.1. Definition

```
typedef struct _SRVSvrInfo{
        int version;
        char *checksum;
    }SRVSvrInfo, *SRVSvrInfoPtr;

    typedef struct _SRVFireSF{
        int firewareversion;
        char *firewarechecksum;
        int hardwareversion;
    }SRVFireSF, *SRVFireSFPtr;

    typedef struct _UpdateInfo{
        SRVSvrInfoPtr srvsvrinfo;
        SRVFireSFPtr srvfiresf;
    }UpdateInfo, *UpdateInfoPtr;
```

4.7.2. Illuminate

该结构用于存储解析 Update_Info.xml 后得到的信息，该 XML 用于存储更新软固件的状态；

包括 Server Information Descriptor 和 Fireware Information 二个部分；

4.8. Device_Info

4.8.1. Definition

```
typedef struct _XmlDeviceInfo{
        int Int;
        int App;
        int Hardwareversion;
    }XmlDeviceInfo, *XmlDeviceInfoPtr;
```

4.8.2. Illuminate

该部分用于存储解析 Device_Info.xml 后得到的信息，该 XML 用于存储与设备相关的信息；

4.9. Resume_Info

4.9.1. Definition

```
typedef struct _ResumeInfo{
    int type;
    char name[MU_NAME_MAX + 1];
}ResumeInfo, *ResumeInfoPtr;

ResumeInfo Resume[10];
```

4.9.2. Illuminate

该结构用于存储解析 Resuming.xml 后得到的信息，该 XML 用于存储正在下载，或是未下载完成的节目信息，包括节目类型和节目名称；

4.10. File_Info

4.10.1. Definition

```
typedef struct _FileInfo{
        char *name;
        unsigned long size;
        char *md5;
        char *crc;
        int type;
        int rate;
        char *displayname;
        char *description;
        char *validbeforedate;
        char *validafterwatch;
```

```
    DatebaseInfoPtr datebaseinfo;
    int userrate;
    int userfreq;
}FileInfo, *FileInfoPtr;
```

4.10.2. Illuminate

该部分用于存储解析 File Description.xml 后的信息，该部分信息作为节目描述信息，在下载完成后，写入设备，与节目内容相关；

在重新开机后，需要上述信息，以确定文件的完整性，及播放时，需要得到上述各信息；

5. Operation Detail 函数细节

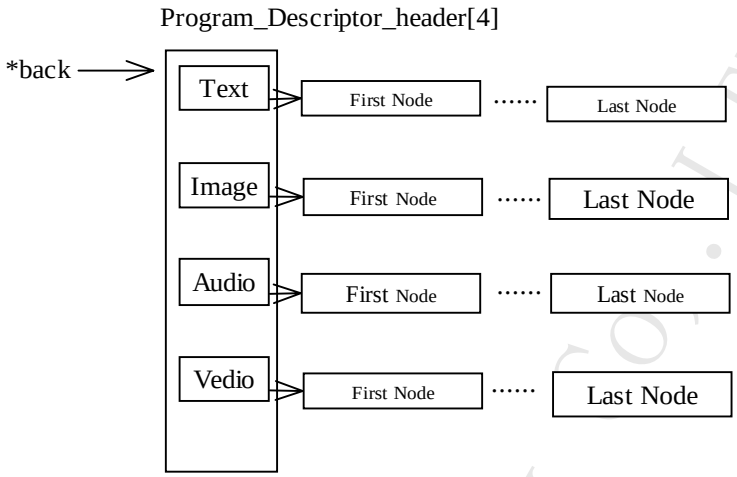
[注意]:
1、本文档提及的所有的创建函数在创建时，都已初始化;

5.1. FDDF_Status

5.1.1. Create

FDDF_Status 结构中由二部分组成：头节点和普通节点。因此，在创建节点时，需要分别提供上述二种节点的创建方式；

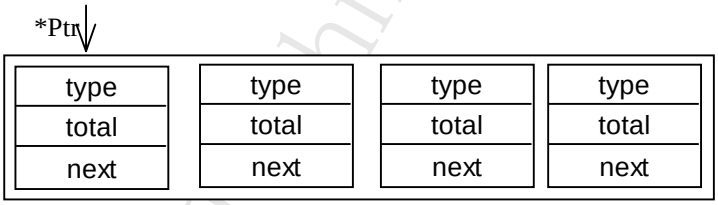
总结构图



5.1.1.1. FDDF_Status Create Head

本函数在初始化时，由调用者调用，用以初始化一个全局的 FDDF 头指针，在以后的应用中，该头结点需要被清空，但是不需要被删除；

■ **Head 结构图**



■ **伪代码**

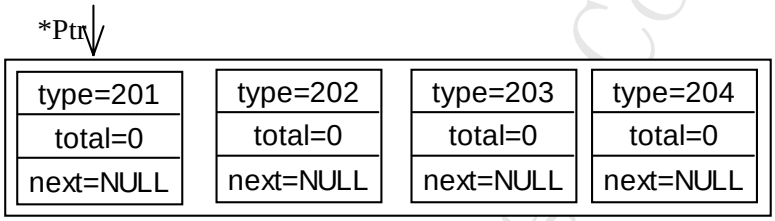
```
int Mu_FddfStatusHeadCreate(FddfStatusHeadPtr *Ptr)
{
    *Ptr = (FddfStatusHeadPtr)malloc(sizeof(FddfStatusHead) * 4);
    if(NULL == *Ptr){
        Mu_ErrorPrint()
        retrun MUNBUF;
    }

    //init the head
    Mu_FddfStatusHeadInit(*Ptr);
    return MUOK;
}
```

■ **返回值**

值	描述
MUOK	无错误
MUNBUF	无内存空间

■ **返回结构**

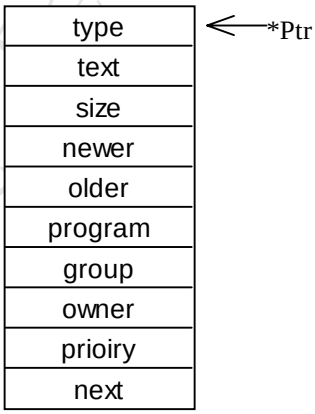
运行后，函数在堆空间中创建以下结构，其初始值如表内示：



5.1.1.2. FDDF_Status Create Node

该节点在 XML 解析或是在 MuFTAD 程序自动下载时，均需要对其进行操作；
在 XML 解析时，需要创建、初始、删除及赋值；
在下载时，需要删除节点；

■ **结构图**



■ **伪代码**

```
int Mu_FddfStatusNodeCreate(FddfStatusNodePtr *Ptr)
{
    int   back;
    *Ptr = (FddfStatusNodePtr)malloc(sizeof(FddfStatusNode));
    if(NULL  == *Ptr)
    {
```

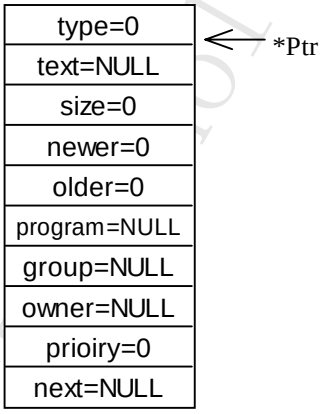
```
        Mu_ErrorPrint();
        return MUNBUF;
    }
    back = Mu_FddfStatusNodeInit(*Ptr);

    return back;
}
```

返回值

值	描述
MUOK	无错误
MUNBUF	无内存空间

■ **返回结构**

运行后，函数在堆空间中创建以下结构，其初始值如表内示：



5.1.2. Init

5.1.2.1. FDDF_Status Init Head

■ **说明**

在每次重新 Query_status 后，从服务器获得 XML 文件后，都需要对信令中的 2xx 类型进行解析，解析前，用于存储的链表头必须被初始化；

该函数用于清空 FDDF_Status 链表的头结点 FddfStatushead 中的值；

[注意]:

*1、 该函数中初始的内容不包括 next 指针所指向的链表中的节点，**该函数***

仅仅在空链表时（只有表头，无节点），被调用；

2、其节点的清除，在自动下载时，被边下载边清除；

3、FddfStatusHead 头结点创建时初始化与重用时初始化函数相同；

■ **伪代码**

```
int Mu_FddfStatusHeadInit(FddfStatusHeadPtr Ptr)
{

    //check Ptr
    int i;
    for(int i= 0; i<4; i ++){

        if(NULL = Ptr){
            Mu_ErrorPrint();
            return MUEERO;          }

        Ptr->type = 201 + i;
        Ptr->total = 0;
        Ptr->next = NULL;

        Ptr ++;
    }

    return MUOK;
}
```

5.1.2.2. FDDF_Status Init Node

■ **说明**

每创建一个新的节点后，都必须初始化其值；

[注意]:

1、清空 FDDF_Status 链表中的节点的信息；

2、区别清空其内容（初始优）与删除；

3、其在整个解析过程中不会被重用，保存在创建时初始化；

■ **伪代码**

```
int Mu_FddfStatusNodeInit(FddfStatusNodePtr Ptr)
```

```
{
    Ptr->type = 0;
    Ptr->size = 0;
    Ptr->newer = 0;
    Ptr->older = 0;

    Ptr->text = NULL;
    Ptr->program = NULL;
    Ptr->group = NULL;
    Ptr->owner = NULL;
    Ptr->priority = 0;
    Ptr->next = NULL;

    return MUOK;

}
```

5.1.3. Insert

■ 说明

将 FDDF_Status 节点插入到链表中时，按节点中 priority 字段安排插入点。将链表维护成一个有序链；

[注意]:

1、Head 为结构数组的首地址;

2、插入时，需要按待插入节点的类型进行链表选择（从四个链表头中选择一个表头）;

■ 伪代码

```
int Mu_FddfStatusNodeInsert (FddfStatusHeadPtr Head, FddfStatusNodePtr Node)
{
    FddfStatusNodePtr back = NULL;

    //find the list header
    Head += (Node->type-200);
    FddfStatusNodePtr Primer = Head->next;

    if(NULL == Primer)
        Head->next = Node;
```

```
    while((Primer != NULL) && (Primer->priority > Node->priority){
        back = Primer;
        Primer = Primer->next;
    }

    Node->next = Primer;
    back->next = Node;

    Head->total ++;
    return MUOK;
}
```

5.1.4. Delete

■ 说明

删除链表中的内容时，包括删除头结点和普通节点，在本项目中，头结点作为全局值放在静态区，对此不进行删除操作；

对普通节点，需要进行删除操作；

[注意]:

1、该项目中，不需要提供对整个链表的删除功能。因为节点间是相互独立的;

2、删除节点时，应该优先释放结构中指针所指的内存区域，否则会导致内存泄露;

■ 伪代码

```
int Mu_FddfStatusNodeDelete(FddfStatusNodePtr Node)
{
    FddfStatusNodePtr Primer = Head->next;
    FddfStatusNodePtr back = NULL;

    while((Primer != NULL) &&  (Primer != Node)){
        back = Primer;
        Primer= Primer->next;
    }
```

```
    if(Primer == NULL){
        Mu_ErrorPrint();
        return MUEERO;
    }

    back->next = Node ->next;
    Mu_Free(Node->text);
    Mu_Free(Node->Program);
    Mu_Free(Node->group);
    Mu_Free(Node->owner);

    free(Node);
    return MUOK;
}
```

5.1.5. Copy

■ 说明

该结构体内的拷贝不需要涉及头节点内信息的拷贝；

XML 解析时，我们需要将解析的值拷贝到节点中去或是分配内存空间，以存储相关信息；

[注意]:

1、拷贝时，用到了宏定义;

2、拷贝对整型，字符串采用不同的存储方式。对整型，直接转换后，赋给结构体内成员；对字符串，分配动态空间后，由结构体成员保存指针;

■ 伪代码

```
int Mu_FddfStatusNodeStrncpy(FddfStatusNodePtr Node,
                             const char *string, const char *name)
{
    if((NULL == string)||(NULL == name) || (NULL == Node))
    {
        Mu_ErrorPrint();
        return MUEERO;
    }
```

```
    Mu_StrncpyInt(Node->type, string, name, “Type”);
    Mu_StrncpyStr(Node->text, string, name, “Text”);
    Mu_StrncpyInt(Node->size, string, name, “Size”);
    Mu_StrncpyInt(Node->newer, string, name, “Dat1”);
    Mu_StrncpyInt(Node->older, string, name, “Dat2”);
    Mu_StrncpyStr(Node->program, string, name, “Dat3”);
    Mu_StrncpyStr(Node->group, string, name, “Dat4”);
    Mu_StrncpyStr(Node->owner, string, name, “Dat5”);
    Mu_StrncpyInt(Node->priority, string, name, “Dat6”);

    return MUOK;
}
```

5.2. DIR_Status

■ 说明

DIR_Status 结构由三部分组成：

1、链表表头结构数组；

可以参考 FDDF_Staus 中的结构组成，它们是一种类型的节点所组成的链表的表头，四个类型：401、402、403、404；

2、链表节点；

链表节点表示具体的一个条款（Item），它是一种类型节点的单独个体。在一个条款里，包含许多不确定的地址；

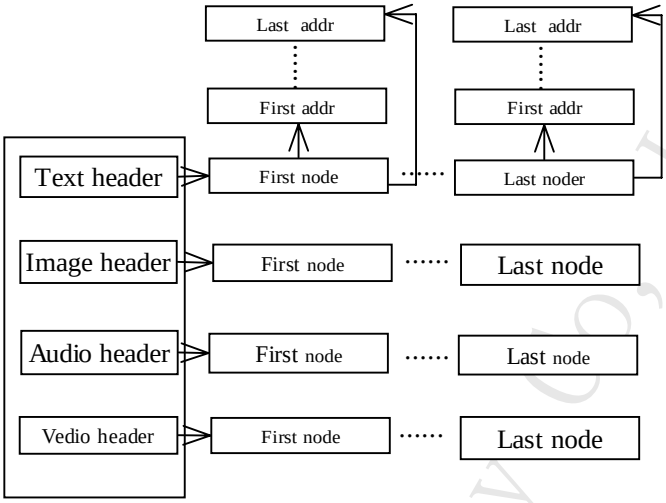
所包含的地址用地址链表来存储；

链表节点是这些地址节点的头节点；

3、地址节点；

如上所述；

■ 结构图



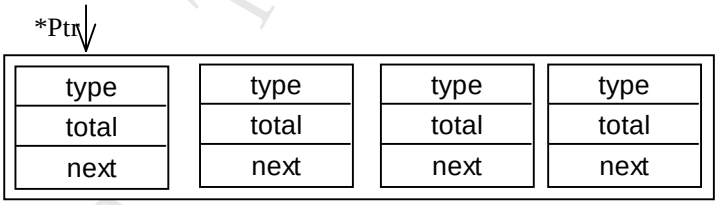
DIRfile_Download_header[4]

5.2.1. Create

该结构的创建不仅包括 DirStatusHead 数组的创建、还包括 DirStatusNode、DirDownloadNode 的创建；

5.2.1.1. DIR_Status Create Head

■ 结构图



■ 伪代码

```
int Mu_DirStatusHeadCreate(DirStatusHeadPtr *Ptr)
{
    *Ptr = (DirStatusHeadPtr)malloc(sizeof(DirStatusHead) * 4);
    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    //init head
```

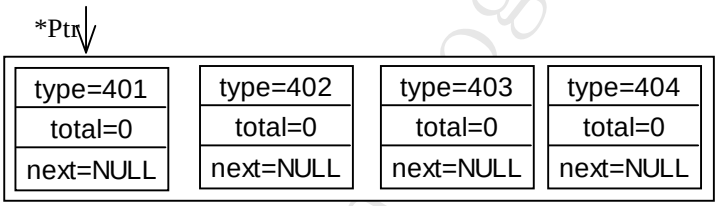
```
    Mu_DirStatusHeadInit(*Ptr);
    return MUOK;
}
```

■ 返回值

值	描述
MUOK	无错误
MUNBUF	无内存空间

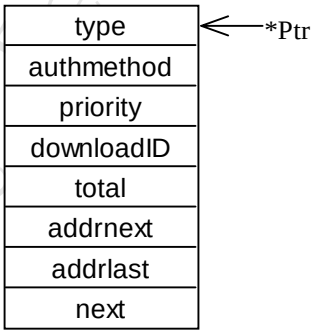
■ 返回结构

运行后，函数在堆空间中创建以下结构，其初始后，值如图示：



5.2.1.2 Dir_Status Create Node

■ 结构图



■ 伪代码

```
int Mu_DirStatusNodeCreate(DirStatusNodePtr *Ptr)
{
    int back;
    *Ptr = (DirStatusNodePtr)malloc(sizeof(DirStatusNode));
    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNBUF;
    }
```

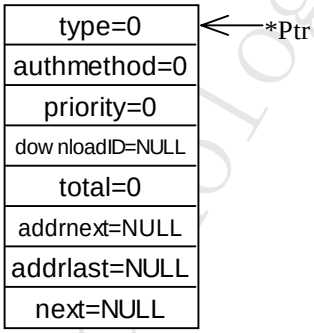
```
    back = Mu_DirStatusNodeInit(*Ptr);
    return back;
}
```

■ 返回值

值	描述
NO_MUERROR	无错误
MUERROR_BUFFER_EMPTY	无内存空间

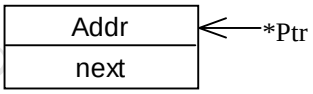
■ 返回结构

运行后，函数在堆栈中创建以下结构，其初始后，值如图示：



5.2.1.3. DIR_Status Addr Create

■ 结构图



■ 伪代码

```
int Mu_DirStatusAddrCreate(DirDownloadAddrPtr *Ptr)
{
    int back;
    *Ptr = (DirDownloadAddrPtr)malloc(sizeof(DirDownloadAddr));
    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    back = Mu_DirDownloadAddrInit(*Ptr);
    return back;
```

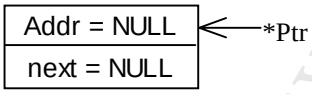
}

■ **返回值**

值	描述
MUOK	无错误
MUNBUF	无内存空间

■ **返回结构**

运行后，函数在堆中创建以下结构，其初始后，值如图示：



5.2.2. Init

对于每一个创建的结点，均需要进行初始化，因此必须提供三类初始化函数；

5.2.2.1. DIR_Status Head Init

■ **说明**

初始化结构数组；
[注意]:
1、DirStatusHead 头结点创建时初始化与重用时初始化函数相同;

■ **伪代码**

```
int Mu_DirStatusHeadInit(DirStatusHeadPtr Ptr)
{
    for(int i= 0; i<4; i ++){

        if(NULL = Ptr){
            Mu_ErrorPrint();
            return MUEERO;
        }

        Ptr->type = 401 + i;
        Ptr->total = 0;
        Ptr->next = NULL;
```

```
        Ptr ++;
    }

    return MUOK;
}
```

5.2.2.2. DIR_Status Node Init

■ 说明

初始化节点；

■ 伪代码

```
int Mu_DirStatusNodeInit(DirStatusNodePtr Ptr)
{
    if(NULL == Ptr)
    {
        Mu_ErrorPrint();
        return MUNNOD;
    }
    Ptr->type = 0;
    Ptr->authmethod = 0;
    Ptr->priority = 0;
    Ptr->DownloadID = NULL;
    Ptr->total = 0;
    Ptr->addrnext = NULL;
    Ptr->addrlast = NULL;
    Ptr->next = NULL;

    return MUOK;
}
```

5.2.2.3. DIR_DownloadAddr Init

■ 说明

初始化地址节点；

■ 伪代码

```
int Mu_DirDownloadAddrInit(DirDownloadAddrPtr Ptr)
{
    if(NULL == Ptr){
        Mu_ErrorPrint();
        return MUNNOD;
    }

    Ptr->addr= NULL;
    Ptr->next = NULL;

    return MUOK;
}
```

5.2.3 Insert

在解析 XML 中的 4xx 内容时，需要考虑到二类节点的插入操作：
一类是一个 Item 节点插入到其链表中去，同 2xx 节点内容的存储；
另一类是地址节点插入到地址链表中去，按解析优先顺序存储；

5.2.3.1. DIR_Status Node Insert

参考 5.2.2.1

5.2.3.2.DIR_Status Addr Insert

■ 说明

将解析得到的地址节点存储到地址链表中去，存储时将新解析的节点放入链表尾部；
在其头结点中，提供尾指针，方便插入；

■ 伪代码

```
int Mu_DirDownloadAddrInsert(DirStatusNodePtr Node, DirDownloadAddrPtr addr)
{
    if((NULL == Node) ||(NULL == addr))
    {
        Mu_ErrorPrint();
        return MUNNOD;
```

```
    }
    if(NULL == Node->addrlast)
    {
        Node->addrnext = addr;
        Node->addrlast = addr;
        return MUOK;
    }
    Node->addrlast->next = addr;
    Node->addrlast = addr;
    Node->total ++;

    return MUOK;
}
```

5.2.4. Delete

在本项目中，对节点的删除有二种情况：
1、当一个地址节点上的内容下载完成，需要从地址链表中删除相应节点；

下载是从地址链表的头节点开始依次下载的，因此这种情况下的删除也是按这种方式进行的；
每次删除一个地址节点；

2、若在解析地址节点时出错，那么需要删除整修链表，但是并不删除其头结点；
一次删除所有的节点，一个链表中，除表头的所有节点；

3、当一个地址链表中所有地址处的内容均被下载完成后，需要删除其头结点，也就是 DirStatuseNode 节点；
一次删除一个 DirStatusNode 节点；

5.2.4.1. DIR_Status Addr Delete one

■ **说明**

从头结点的 next 指针所指开始删除，一次删除一个节点；
[注意]:
1、应该先释放地址节点中所指向的地址内存(addr)，然后再释放地址节点本身;

■ 伪代码

```
int Mu_DirDownloadAddrDeleteone(DirStatusNodePtr Node)
{
    DirDownloadAddrPtr address = NULL;

    if((address= Node->addrnext) == NULL)
    {
        Mu_ErrorPrint();
        return MUNNOD;
    }
    Node->addrnext = address->addrnext;

    if(address == Node->addrlast)
        Node->addrlast = NULL;

    Mu_Free(address->addr);

    free(address);

    return MUOK;
}
```

5.2.4.2. DIR_Status Addr Delete List

■ 说明

从头结点的 next 指针所指开始删除，一次删除链表中，除头结点外的所有节点；

[注意]:

1、应该先释放地址节点中所指向的地址内存(addr)，然后再释放地址节点本身;

■ 伪代码

```
int Mu_DirDownloadAddrDeleteList(DirStatusNodePtr Node)
{
    DirDownloadAddrPtr address = NULL;

    while(Node->addrnext != NULL){
```

```
        if((address= Node->addrnext) == NULL)
        {
            Mu_ErrorPrint();
            return MUNNOD;
        }
        Node->addrnext = address->addrnext;

        if(address == Node->addrlast)
            Node->addrlast = NULL;

        Mu_Free(address->addr);
        free(address);
    }
    return MUOK;
}
```

5.2.4.3. DIR_Status Node Delete

■ 说明

删除一个 item 元素信息结点 DriStatusNode；
[注意]:
1、实现中调用 Mu_DirDownloadAddrList 函数释放一个地址链表;

■ 伪代码

```
int Mu_DirStatusNodeDelete(DirStatusHeadPtr Head, DirStatusNodePtr Node)
{
    Head += (Node->type -400);

    DirStatusNodePtr Primer = Head->next;
    DirStatusNodePtr back = NULL;

    while((Primer != NULL) &&  (Primer != Node)){
        back = Primer;
        Primer= Primer->next;
    }

    if(Primer == NULL){
        Mu_ErrorPrint();
        return MUNNOD;
```

```
    }

    back->next = Node ->next;
    Mu_DirDownloadAddrDeleteList(Node);

    Mu_Free(Node->downloadID);
    free(Node);
    return MUOK;
}
```

5.2.5. Copy

XML 解析过程中，需要将所解析的信息存储到相关结构中去，在本结构中，对该结构的赋值操作可分为二种：

1、拷贝到 DirStatusNode 节点

这些节点中维护着一个条款(Item)中的公有消息；

2、拷贝到地址节点

这些节点中仅仅保存一个地址信息；

5.2.5.1. Dir_Status Node Copy

■ 说明

将一个条款中的公有消息存储到 DirStatusNode 节点中；

■ 伪代码

```
int Mu_DirStatusNodeStrncpy(DirStatusNodePtr Node,
                            const char *string, const char *name)
{
    if((NULL == string)||(NULL == name) || (NULL == Node))
    {
        Mu_ErrorPrint();
        return MUNNOD;
    }

    Mu_StrncpyInt(Node->type, string, name, “Type”);
    Mu_StrncpyInt(Node->authmethod, string, name, “Dat1”);
    Mu_StrncpyInt(Node->priority, string, name, “Dat2”);
```

```
    Mu_StrncpyStr(Node->DownloadID, string, name, “Dat3”);

    return MUOK;
}
```

5.2.5.2. Dir_Download Addr Copy

■ 说明

将一个条款中的公有消息存储到 DirDownloadAddr 节点中；

■ 伪代码

```
int Mu_DirDownloadAddrStrncpy(DirDownloadAddrPtr Node, char *string)
{
    if((NULL == string) || (NULL == Node))
    {
        Mu_ErrorPrint();
        return MUNNOD;
    }

    if(NULL == (Node->addr = malloc(strlen(string) + 1)))
    {
        Mu_ErrorPrint();
        return MUEERO;
    }
    memset(Node->addr, 0, strlen(string)+1)
    strncpy(Node->addr, string, strlen(string));

    return MUOK;
}
```

5.3. Service_Info

■ 说明

Service_Info 结构大体由二部分组成：
1、ApipathList
该结构内存储了服务器提供的可访问的接口，接口函数名称全部存储于动态

内存区；

2、ServerHead

该结构体是一个服务器地址的链表表头，在协议中，服务器的地址个数是不确定的；

该头结点中提供二个节点指针，一个指向链表首，另一个指向链表尾节点；

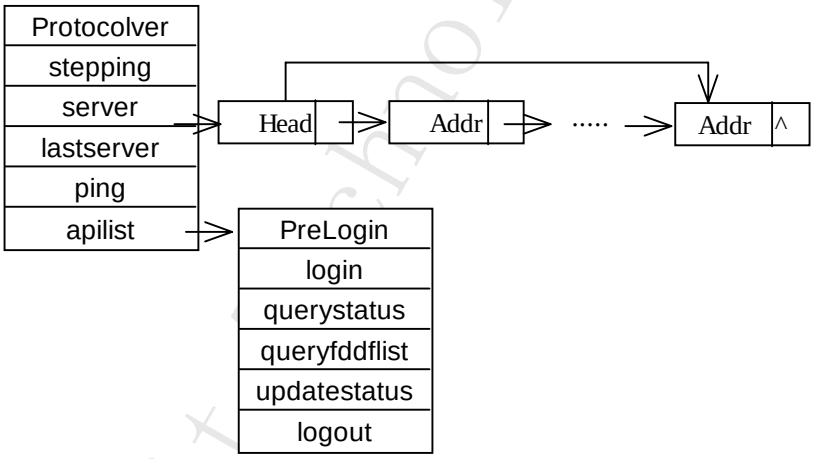
[注意]:

1、《MuFTAD 需求文档中》对地址存储结构设计为循环链表，具体设计时，仍采用单向链表存储;

2、新建地址节点插入链表表尾;

3、该结构不会在整个下载过程中重复解析，它只会在开机时，一次性解析出所有信息，存储于动态内存区，方便以后的使用;

■ **结构图**



5.3.1. Create

因为该结构由 Server 链表和 ApiPathlist 结构结成，所以，在创建该结构时，应该一并创建该信息；

但是对于地址结点，必须提供创建函数，方便解析时动态地创建节点；

5.3.1.1. ServiceInfo Create

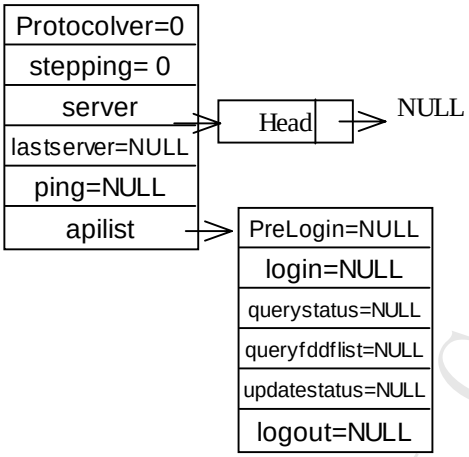
■ **说明**

创建 ServiceInfo 结构，包括 ServerHead 和 ApiPathList 结构；

■ 伪代码

```
int Mu_ServiceInfoCreate(ServiceInfoPtr *Ptr)
{
    ServerHeadPtr head = NULL;
    ApiPathListPtr api = NULL

    head = (ServerHeadPtr)malloc(sizeof(ServerHead));
    if(NULL == head){
        Mu_ErrorPrint();
        return MUNNOD;
    }

    api = (ApiPathListPtr)malloc(sizeof(ApiPathList));
    if(NULL == api){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    *Ptr = (ServiceInfoPtr)malloc(sizeof(ServiceInfo));
    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    (*Ptr)->server = head;
    (*Ptr)->apipathlist = api;

    Mu_ServiceInfoInit(*Ptr);
    return MUOK;
}
```

■ 创建后的结构



5.3.1.2. ServerNode Create

■ 说明

在解析 server Information Description 时，需要动态地创建服务器地址节点，以存储服务器地址；

■ 伪代码

```
int Mu_ServerNodeCreate(ServerNodePtr *Node)
{
    *Node = (ServerNodePtr)malloc(sizeof(ServerNode));

    if(NULL == *Node){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    Mu_ServerNodeInit(*Node);

    return MUOK;
}
```

5.3.2. Init

由于 Service_Info 结构分为二部分组成，所以必须相应地提供二个初始化函

数；

5.3.2.1. Service_Info Init

■ 说明

初始化 Service Info 结构中的所有项，包括 ServerHead 头结点和 ApiPathList 结构;

该结构不会被重用，在整个过程中 Sever Information Descriptor 只会被调用一次，所以该函数可以被忽略；

[注意]:
1、DirStatusHead 在设计时不考虑被重用问题;

■ 伪代码

```
int Mu_ServiceInfoInit(ServiceInfoPtr Ptr)
{
    //Init
    Ptr->Protocolver = 0;
    Ptr->stepping = 0;
    Ptr->lastserver = NULL;
    Ptr->ping = NULL;

    //ServerHead Head
    Ptr->server->total = 0;
    Ptr->server->next = NULL;
    Ptr->server->last = NULL;

    //ApiPathList
    Ptr->apipathlist->prelogin = NULL;
    Ptr->apipathlist->login = NULL;
    Ptr->apipathlist->querystatus = NULL;
    Ptr->apipathlist->updatestatus = NULL;
    Ptr->apipathlist->queryfddflist = NULL;
    Ptr->apipathlist->logout = NULL;

    returnMUOK;
}
```

5.3.2.2. Server Node Init

■ 说明

该节点用于存储 Server Information Description XML 文件中的 addr 字段信息；

■ 伪代码

```
int Mu_ServerNodeInit(ServerNodePtr Ptr)
{
    Ptr->addr = NULL;
    Ptr->next = NULL;
    return MUOK;
}
```

5.3.3. Insert

■ 说明

该函数的功能是将解析出来的地直接点加入到链表中去；
[注意]:
1、加入地址链表的顺序按解析顺序完成，解析出的值加入到链表的尾部;

■ 伪代码

参考 5.2.3.2.DIR_Status Addr Insert

5.3.4. Delete

在本项目中，该 XML 文件解析，及整个应用中，均不会涉及到删除函数。因此文档省略该部分；

5.3.5. Copy

将从 Server Information Description XML 文件中解析的值存储入 Service_info 结构中的相应项中；
按该结构中的类型组成，以及我们创建的节点，我们需要创建二个拷贝函数；

[注意]:

1、在解析 Server Information Description 时，需要区分 addr 字段与非 addr 字段;

5.3.5.1.Service_Info Copy

■ **说明**

从 Server Information Description XML 文件中解析的非 addr 字段值存储入 ServiceInfo 结构中的相应项中；

■ **伪代码**

```
int Mu_ServiceInfoStrncpy(ServiceInfoPtr Ptr, const xmlChar *string, const xmlChar *name)
{
    if((NULL == string)||(NULL == name) || (NULL == Node))
    {
        Mu_ErrorPrint();
        return MUNNOD;
    }

    Mu_StrncpyInt(Ptr->Protocolver, string, name, “Protocol_Ver”);
    Mu_StrncpyInt(Ptr->stepping, string, name, “Stepping”);
    Mu_StrncpyStr(Ptr->ping, string, name, “Ping”);
    Mu_StrncpyStr(Ptr->lastserver, string, name, “Last”);

    Mu_StrncpyStr(Ptr->apipathlist->prelogin, string, name, “Pre_Login”);
    Mu_StrncpyStr(Ptr->apipathlist->login, string, name, “Login”);
    Mu_StrncpyStr(Ptr->apipathlist->querystatus, string, name, “Query_Status”);
    Mu_StrncpyStr(Ptr->apipathlist->queryfddflist,
                    string, name, “Query_FDDF_List”);
    Mu_StrncpyStr(Ptr->apipathlist->updatestatus, string,name,”Update_Status”);
    Mu_StrncpyStr(Ptr->apipathlist->logout, string, name, “Logout”);

    return MUOK;

}
```

5.3.5.2. ServerNode Copy

■ 说明

将从 Server Information Description XML 文件中解析的 addr 字段值存储入 ServerNode 结构中的相应项中；

■ 伪代码

```
int Mu_ServerNodeStrncpy(ServerNodePtr Ptr, const xmlChar *string, const xmlChar *name)
{
    if((NULL == string)||(NULL == name) || (NULL == Ptr))
    {
        Mu_ErrorPrint();
        return MUNNOD;
    }

    Mu_StrncpyStr(Ptr->addr, string, name, “Addr”);

    return MUOK;

}
```

5.4. FDDF_List

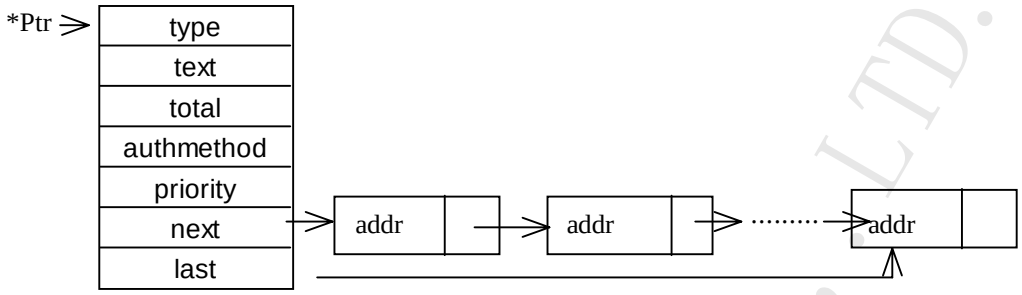
■ 说明

在设备访问服务器的 query_Status API 后，得到了 2xx 类节目的总体信息，但是不包括详细的 FDDF XML 文件。

根据 query_Status 后得到的信息，访问服务器，得到相关节目的 FDDF List，也即 FDDF XML 文件的下载地址。

对应于一个条款 Item，可能包含多个下载地址，并且地址在 XML 文件中的元素名是可变的，因此采用单独的数据结构来存储该地址；

■ 结构图



5.4.1. Create

FDDF_List 由二部分组成，FddfListHead 和 FddfListNode；

FddfListHead 对应于上述链表的头部结点，而 FddfListNode 则对应于普通节点，用于存储 FDDF XML 文件的下载路径；

因此在创建该结构的节点时，需要提供二种类型结构的创建函数；

5.4.1.1. FDDF_List Head Create

■ **说明**

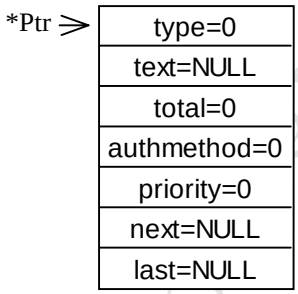
新节点在被创建时，会被初始化，该节点应该被存储在动态存储区，在整个下载过程中，不会被删除；

■ **伪代码**

```
int Mu_FddfListHeadCreate(FddfListHeadPtr *Ptr)
{

    *Ptr = (FddfListHeadPtr)malloc(sizeof(FddfListHead));
    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNNOD;
    }

    //Init
    (*Ptr)->type = 0;
    (*Ptr)->text = NULL;
```

```
    (*Ptr)->total = 0;
    (*Ptr)->authmethod= 0;
    (*Ptr)->priority = 0;
    (*Ptr)->next = NULL;
    (*Ptr)->last = NULL;

    return MUOK;
}
```

■ **创建后的结构**



5.4.1.2. FDDF_List Node Create

```
int Mu_FddfListNodeCreate(FddfListNodePtr *Ptr)
{
    参考 5.2.1.3.
    //Mu_FddfListNodeInit(*Ptr);
}
```

5.4.2. Init

该结构由二个部分组成，因此需要提供二个单独的初始化函数以完成相应工作；

5.4.2.1. FDDF_List Head Init

■ **说明**

该函数用于初始化该结构的头部结点，在每次解析 FDDF List XML 文件时，都会调用此函数初始化该结构；

[注意]:

1、FddfListHead 在整个过程中会被多次使用，头结点需要被重新初始化;
*2、FddfListHead 重新初始化与创建时初始化是**不相同**的;*

■ 伪代码

```
int Mu_FddfListHeadInit(FddfListHeadPtr Ptr)
{
    Ptr->type = 0;
    Mu_Free(Ptr->text);
    Ptr->total = 0;
    Ptr->authmethod= 0;
    Ptr->priority = 0;
    Ptr->next = NULL;
    Ptr->last = NULL;

    return MUOK;
}
```

5.4.2.2. FDDF_List Node Init

■ 说明

该函数用于初始化 FDDF XML 文件在服务器上的路径消息，在存储时需要单独创建节点以存储该信息;

在每个节点被创建时，均需要被初始化;

[注意]:
1、FddfListNode 在整个过程中不会被重用，只存在创建时初始化的情况;

■ 伪代码

```
int Mu_FddfListNodeInit(FddfListNodePtr Ptr)
{
    参考 5.3.2.1.或是 5.3.2.3.
}
```

5.4.3. Insert

■ 说明

对于每一个 FDDF_List 节点，在创建并被赋值后，需要插入到链表中去；

[注意]:

1、该链表按解析顺序排列，即按解析出的信息依次添加到链表的尾部;

■ **伪代码**

```
int Mu_FddfListNodeInsert(FddfListHeadPtr Head, FddfListNodePtr Node)
{
    参考 5.2.3.2.
    [注意]:
    1、虽然本函数提供了 Head 指针，但是其是为了同其他函数相统一，在使用
       时，Head 指针将会是一个全局变量;

}
```

5.4.4. Delete

■ **说明**

对于本结构，只涉及到地址节点的删除，本且，执行删除操作时，均从链表表头开始删除；

■ **伪代码**

```
int Mu_FddfListNodeDelete(FddfListHeadPtr Head)
{
    参考 5.2.4.1.
}
```

5.4.5. Copy

本结构中存在二种类型的节点，因此在解析 FDDF List XML 文件时，需要提供二种类型的数据拷贝函数，以将信息存储到相应的子项中；

5.4.5.1. FDDF_List Head Copy

■ **说明**

该函数将 FDDF List XML 文件<payload>字段中的一些公共信息，存储到头节点中；

■ 伪代码

```
int Mu_FddfLisitHeadStrncpy(FddfListHeadPtr Head, const char *sring, const char *name)
{
    参考 5.3.5.1.
}
```

5.4.5.2. FDDF_List Node Copy

■ 说明

该函数将 FDDF List XML 文件<payload>字段中的路径信息，存储到节点中；

■ 伪代码

```
int Mu_FddfListNodeStrncpy(FddfListNodePtr Node, char *string, char *name)
{
    参考 5.3.5.2.
}
```

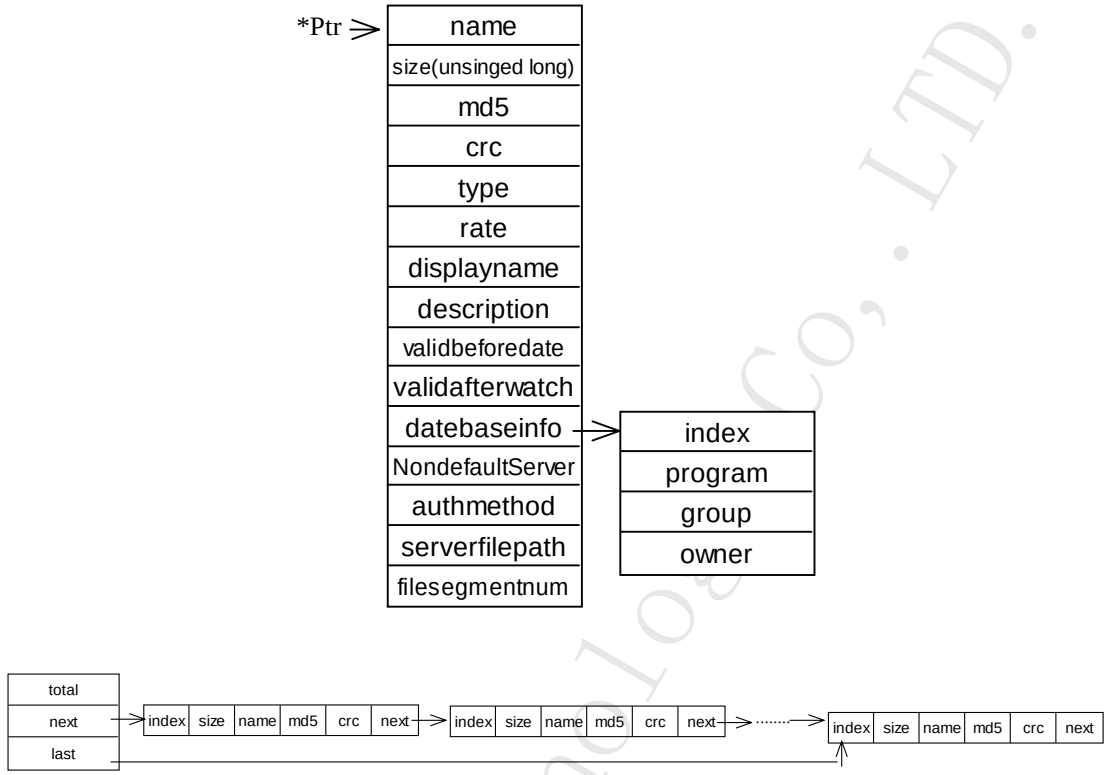
5.5. FDDF_File

■ 说明

定义的结构用于存储，从 FDDF 中解析出的关于节目的信息，这个 XML 文件从服务器获得，Device 只有在取得该 XML 文件，并解析出其中的信息后，才可以下载该文件所描述的节目；

FDDF 分为二个部分：
一部分为节目整体描述性信息，该信息由结构体 FddfFile 定义；
另一部分为节目的分段信息，该部分由以单独的结构体 SegmentHead 为头结点的链表存储，该部分还包括分段节点 SegmentNode；

■ 结构图



5.5.1. Create

节点的创建分为二个部分：

一、节目总体信息描述结构 FddfFile 的创建，该结构在整个软件工作过程中被置为全局变量，会被多次重新初始化；

二、节目分段信息 SegmentHead 与 SegmentNode 结构的创建，在工作过程中，SegmentHead 会被置为全局变量，并且会被多次初始化；

5.5.1.1. FddfFile Create

■ 说明

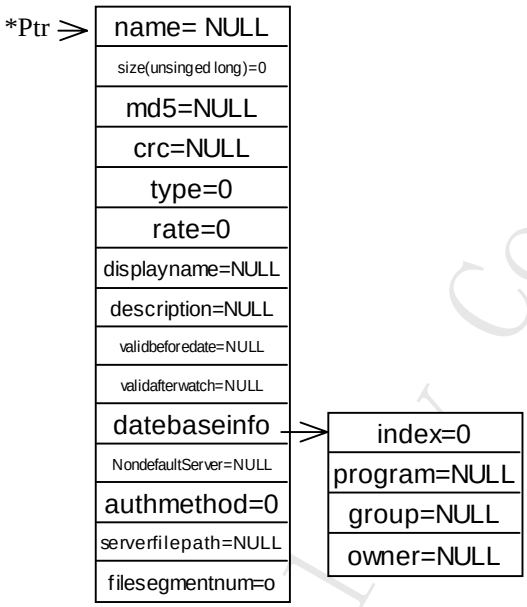
用于存储 FDDF XML 文件中关于一个节目的整体性描述信息，这部分信息会因不同节目而导致内容的不一样，因此会汲及到多次初始化；

■ 伪代码

int Mu_FddfFileCreate(FddfFilePtr *Ptr)

```
{

    DatebaseInfoPtr database = NULL;
    database = (DatebaseInfoPtr)malloc(sizeof(DatebaseInfo))
    if(NULL == database){
    {
        Mu_ErrorPrint();
        return MUNBUF;
    }

    *Ptr = (FddfFilePtr)malloc(sizeof(FddfFile));
    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    (*Ptr)->databaseinfo = database;

    //init
    (*Ptr)->name = NULL;
    (*Ptr)->size = 0;
    (*Ptr)->md5 = NULL;
    (*Ptr)->crc = NULL;
    (*Ptr)->type = 0;
    (*Ptr)->rate = 0;
    (*Ptr)->displayname = NULL;
    (*Ptr)->description = NULL;
    (*Ptr)->validbeforedate = NULL;
    (*Ptr)->description = NULL;
    (*Ptr)->nondefaultserver = NULL;
    (*Ptr)->authmethod = 0;
    (*Ptr)->serverfilepath = NULL;
    (*Ptr)->filesegmentnum = 0;

    database->index = 0;
    database->program = NULL;
    database->group = NULL;
    database->owner = NULL;

    return MUOK;
}
```

■ 创建的结构



5.5.1.2. SegmentHead Create

■ 说明

用于维护一个存储节目的所有分段信息的链表，该表头结构包含二个指针，分别指向下一个分段信息结构和最后一个分段信息结构；

该头结点 SegmentHead 被存放于动态存储区，并且被设置为全局变量，在整个工作过程中，需要被多次初始化；

[注意]:

1、该节点被重复初始化时，该链表中的节点都应该已经被释放;

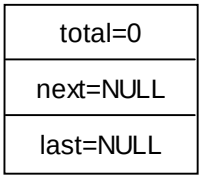
2、该节点被重复初始化时，函数相同;

■ 伪代码

```
int Mu_SegmentHeadCreate(SegmentHeadPtr *Ptr)
{
    *Ptr = (SegmentHeadPtr)malloc(sizeof(SegmentHead));

    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNBUF;
    }
```

```
    //init
    Mu_SegmentHeadInit(*Ptr);
    return MUOK;

}
```

■ 创建后的结构



5.5.1.3. SegmentNode Create

■ 说明

用于存储一个节目的所有分段信息；

该节点只会被初始化一次，每次都是新建该节点时初始化；

■ 伪代码

```
int Mu_SegmentNodeCreate(SegmentNodePtr *Ptr)
{
    *Ptr = (SegmentNodePtr)malloc(sizeof(SegmentNode));

    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    //init
    Mu_SegmentNodeInit(*Ptr);
    return MUOK;
}
```

■ 创建后的结构

5.5.2. Init

根据上述说明，整个结构分为三个部分组成，因此需要相应的函数来初始化结点；

5.5.2.1. FddfFile Init

■ 说明

该结构被申明为一个指针型全局变量，在创建时会被初始化，在以后重复使用时，仍需要将其进行初始化，但是这二次的初始化函数是不相同的；

[注意]:
1、不要释放 DatabaseInfo 结构;

■ 伪代码

```
int Mu_FddfFileInit(FddfFilePtr Ptr)
{
    Mu_Free(Ptr->name);
    Ptr->size = 0;
    Mu_Free(Ptr->md5);
    Mu_Free(Ptr->crc);
    Ptr->type = 0;
    Ptr->rate = 0;
    Mu_Free(Ptr->displayname);
    Mu_Free(Ptr->description);
    Mu_Free(Ptr->validbeforedate);
    Mu_Free(Ptr->description);
    Mu_Free(Ptr->nondefaultserver);
    Ptr->authmethod = 0;
    Mu_Free(Ptr->serverfilepath);
    Ptr->filesegmentnum = 0;

    Ptr->databaseInfo->index = 0;
    Mu_Free(Ptr->databaseinfo->program);
    Mu_Free(Ptr->databaseinfo->group);
    Mu_Free(Ptr->databaseinfo->owner);

    return MUOK;
}
```

5.5.2.2. SegmentHead Init

■ 说明

该函数用于初始化分段链表的表头结点，该表头也被申明为一个指针型的 SegmentHead 结构，在创建时，或是在被重用前，均需要被初始化；

[注意]:
1、虽然该结构在创建时，在被重用时，均需要被初始化，但是二种情况下，所使用的初始化函数是相同的;

■ 伪代码

```
int Mu_SegmentHeadInit(SegmentHeadPtr Ptr)
{
    Ptr->total = 0;
    Ptr->next = NULL;
    Ptr->last = NULL;

    return MUOK;
}
```

5.5.2.3. SegmentNode Init

■ 说明

该函数用于初始化链表节点，该节点中存储着节目的分段信息，该函数在创建新的节点时被用来初始化结构内的值；

■ 伪代码

```
int Mu_SegmentNodeInit(SegmentNodePtr Ptr)
{
    Ptr->index = 0;
    Ptr->size = 0;
    Ptr->name = NULL;
    Ptr->md5 = NULL;
    Ptr->crc = NULL;
    Ptr->next = NULL;
```

return **MUOK**;
}

5.5.3. Insert

■ **说明**

对于每一个 SegmentNode 节点，在创建后，都应该插入到以 SegmentHead 为首结点的链表中去；

链表是以解析顺序来维护的，对每个新建的结点，都被放到链表的最尾部，不需要进行额外的排序；

■ 伪代码

```
int Mu_SegmentNodeInsert(SegmentHeadPtr Head, SegmentNodePtr Node)
{
    参考 5.4.3.

    [注意]:
    1、虽然本函数提供了 Head 指针，但是其是为了同其他函数相统一，在使用时，Head 指针将会是一个全局变量;

}
```

5.5.4. Delete

■ 说明

对于本章节中所述的节点，只有分段节点会在使用过程中被释放掉；
释放时是从头结点开始的，考虑到后期下载时使用多线程，该链表在整个节目下载完成或是下载失败后，将整个链表一并删除；

■ 伪代码

```
int Mu_SegmentNodeDelete(SegmentHeadPtr Head)
{
    参考 5.2.4.2.
    [注意]:
    1、本函数执行的时间;
}
```

5.5.5. Copy

本章节涉及到的结构由二部分组成，在进行 XML 解析时，需要保存获得的值，因此需要二个函数对结构进行赋值。

[注意]:
1、结构中的 size 字段是 unsigned long 类型；

5.5.5.1. FddfFile Copy

■ 说明

将解析得到的节目整体描述信息存入 FddfFile 结构中，对于字符串信息，需要分配动态内存区域以存储；

■ 伪代码

```
int Mu_FddfFileStrncpy(FddfFilePtr Ptr, const xmlChar *string, const xmlChar *name)
{
    参考 5.3.5.1.
    [注意]:
    1、结构中的 size 字段设置为 unsigned long 类型，格式转换时需要特别注意;

}
```

5.5.5.2. SegmentNode Copy

■ 说明

将解析得到的分段信息存入 SegmentNode 结构中，对于字符串信息，需要分配动态内存区域以存储；

■ 伪代码

```
int Mu_SegmentNodeStrncpy(SegmentNodePtr Ptr, const xmlChar *string, const xmlChar *name)
{
    参考 5.3.5.1.
    [注意]:
    1、结构中的 size 字段设置为 unsigned long 类型，格式转换时需要特别注意;

}
```

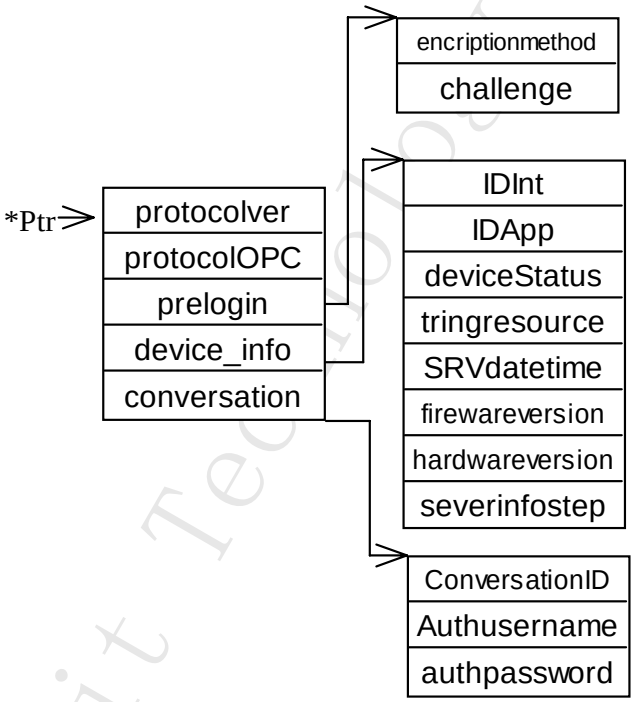
5.6. Eyewear_ADP

■ 说明

Eyewear_ADP 是用于存储解析 CMD XML 中的内容，CMD XML 是 Device 与服务器交互时使用的信令的一部分，其包含除<payload>字段外的所有字段；

Eyewear_ADP 由三个结构体构成，按无素进行分类；

■ 结构图



5.6.1. Create

■ 说明

结构创建时，在动态内存区域申请区域存储该结构所有的项，并对其初始化，由于该结构在整个运行过程中，是重复利用的，每次使用时，需要对其进行初始化；

软件将会使用全局 EyewearADP 指针指向该区域；

■ 伪代码

```
int Mu_EyewearADPCreate(EyewearADPPtr *Ptr)
{

    PreLoginPtr Pre_log = NULL;
    DeviceInfoPtr device = NULL;
    ConversationPtr conv = NULL;

    Pre_log = (PreLoginPtr)malloc(sizeof(PreLogin));
    if(NULL == Pre_log){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    //init
    Pre_log->encryptionmethod = 0;
    Pre_log->challenge = NULL;

    device = (DeviceInfoPtr)malloc(sizeof(DeviceInfo));
    if(NULL == device){
        Mu_ErrorPrint();
        return MUNBUF;
    }
    //init
    device->IDInt = NULL;
    device->App = NULL;
    device->devicestatus = 0;
    device->trigsource = 0;
    device->SRVdatetime = NULL;
    device->firewareversion = 0;
    device->hardwareversion = 0;
    device->serverinfostep = 0;

    conv = (ConversationPtr)malloc(sizeof(Conversation));
    if(NULL == conv){
        Mu_ErrorPrint();
        return MUNBUF;
    }
     //init
    conv->ConversationID = NULL;
```

```
    conv->authusername = NULL;
    conv->authpassword = NULL;

    *Ptr = (EyeweaerADPPtr)malloc(sizeof(EyewearADP));
    if(NULL == *Ptr){
        Mu_ErrorPrint();
        return MUNBUF;
    }

    (*Ptr)->protocolver = 0;
    (*Ptr)->protocolOPC = 0;
    (*Ptr)->prelogin = Pre_log;
    (*Ptr)->device_info = device;
    (*Ptr)->conversation = conv;

    return MUOK;
}
```

5.6.2. Init

■ 说明

初始化用于该结构被重用，在创建该结构时，是不能用该函数来初始化其结构的；

[注意]:
1、不能释放 PreLogin、Device_info、conversation 本身的内存区域;

■ 伪代码

```
int Mu_EyewearADPInit(EyewearADPPtr Ptr)
{
    //init
    Ptr->prelogin->encriptionmethod = 0;
    Mu_Free(Ptr->prelogin->challenge);

    //init device
    Mu_Free(Ptr->device_info->IDInt );
    Mu_Free(Ptr->device_info ->App );
```

```
    Ptr->device_info ->devicestatus = 0;
    Ptr->device_info ->trigresource = 0;
    Mu_Free(Ptr->device_info ->SRVdatetime );
    Ptr->device_info ->firmwareversion = 0;
    Ptr->device_info ->hardwareversion = 0;
    Ptr->device_info ->serverinfostep = 0;

    //init
    Ptr->conversation->conversationID = 0;
    Mu_Free(Ptr->conversation ->authusername);
    Mu_Free(Ptr->conversation ->authpassword );

    Ptr->protocolver = 0;
    Ptr->protocolOPC = 0;

    return MUOK;
}
```

5.6.3. Insert

该结构不存在链表等其他存储结构，不需要插入操作；

5.6.4. Delete

该结构不存在链表等其他存储结构，不需要删除操作，每次重用时，只需要调用 5.6.2.中函数，就可以释放结构所指向的所有内存区域；

5.6.5. Copy

■ **说明**

当解析 CMD 信令获得的信息后，必须存储到本章节所介绍的结构中去，具体操作由本函数完成；

■ **伪代码**

```
int Mu_EyewearADPStrncpy(EyewearADPPtr Ptr, const xmlChar *string, const
```

xmlChar *name)
{
　　参考 5.3.5.1.
}

5.7. Update_info

　　参考 5.6.
　　针对本结构的函数有：
int Mu_UpdateInfoCreate(UpdateInfoPtr *Ptr);
int Mu_UpdateInfoStrncpy(UpdateInfoPtr Ptr, const xmlChar *string, const xmlChar *name);

　　[注意]:
　　1、本结构用于存储 Update XML 文件中的信息，该结构仅仅在开机检验更新固件是否完整;
　　2、针对本结构没有 Insert 和 Delete 操作函数;

5.8. Device_Info

　　参考 5.7.
　　针对本结构的函数有：

int Mu_DeviceInfoCreate(DeviceInfoPtr *Ptr);
int Mu_DeviceInfoStrncpy(DeviceInfoPtr Ptr, const xmlChar *string, const xmlChar *name);

　　[注意]:
　　1、本结构用于存储 Device info XML 文件中的信息;
　　2、针对本结构没有 Insert 和 Delete 操作函数;

5.9. Resume_Info

■ **说明**

在下载过程中，程序一下地维护一个未下载或是未载完成节目的信息，以方便设备在工作时断电，或是发生其他故障后，记录上述信息；

在开机后，程序检测上述节目，以确定是否完整，并按相关操作删除或是重新下载节目；

Resume 是一结构数组；

5.9.1. Create

由于此结构在程序工作时一直处理内存，其创建与分配不需要另外函数完成；

[注意]:

1、该结构虽然不需要创建，但是在使用时，必须调用初始化函数时置初值;

5.9.2. Init

■ **说明**

初始化 Resume 数组中的一个结构体；

■ **伪代码**

```
int Mu_ResumeInfoInit(ResumeInfoPtr Ptr, int num)
{
    Ptr += num;
    if(Ptr == NULL){
        Mu_ErrorPrint();
        return MUNNOD;
    }

    Ptr->type = 0;
    memset(Ptr->name, 0, MU_NAME_MAX + 1);

    return MUOK;
}
```

[注意]:

1、Ptr 为 Resume 数组的首地址;

2、只提供单一结构的初始化，在程序运行之初，若要对数组内所有结构初始化，由调用者完成;

5.9.3. Insert

■ **说明**

将从 resuming xml 文件中，解析得到的，正在下载或是未下载完成的节目信息放入该数组的空闲结构体内；

整型指针 num 用于向调用者传回插入点位置，便于下载完成后清除该结构内的信息；

[注意]：

1、若 num 用全局值，可以不使用该参数；

■ **伪代码**

```
int Mu_ResumeInsert(ResumeInfoPtr Ptr, int type, const xmlChar *name，int num, const xmlChar* string)
{
    int i;
    char str[100];
    for(i = 0; i<10; i++){
        if((Ptr+i)->type)
            break;
    }

    if(i == 10){
        Mu_ErrorPrint();
        return MUEOVR;
    }

    Mu_ResumeInfoInit(Ptr, i);
    (Ptr + i)->type = type;
    Convert(type,str);
    if(xmlStrcmp((xmlChar*)str,name))
    {
        strncpy((Ptr+ i)->name, name, xmlStrlen(name)+1);
    }
```

num = i;
return **MUOK**;
}

5.9.4. Delete

对该结构的删除并不是删除结构体本身，而是清空一个结构体内的内容，该操作可以由初始化函数完成；

5.9.5. Copy

该操作已由 Insert 函数完成；

5.10. File_Info

参考 5.7.
针对本结构的函数有：

int Mu_FileInfoCreate(FileInfoPtr *Ptr);
int Mu_FileInfoStrncpy(FileInfoPtr Ptr, xmlChar *string, const xmlChar *name);

[注意]:
1、本结构用于存储 File info XML 文件中的信息;
2、针对本结构没有 Insert 和 Delete 操作函数;

6. Appendix 附录

6.1. macro 宏定义

#define MU_MAX_FILE_NAME 200

6.1.1. Mu_Free

```
#define Mu_Free(note)\
do{\
    if(note != NULL){\
        free(note);\
        note = NULL;\
}while(0)
```

6.1.2. Mu_StrncpyInt

```
#define Mu_StrncpyInt(note, string, name, place)\
do{\
        if(!xmlStrcmp(name, (const xmlChar*)place))\
        note = atoi(string);\
}while(0)
```

6.1.3. Mu_StrncpyStr

```
#define Mu_StrncpyStr(note, string, name, place)\
do{\
    if(!xmlStrcmp(name, (const xmlChar*)place))\
    {\
        if(NULL != ((note = malloc(xmlStrlen(string) + 1))))\
        {\
            memset(note, 0, xmlStrlen(string)+ 1);\
            strncpy(note, (const xmlChar*)string,xmlStrlen(string));\
        }\
    }while(0)
```

6.1.4. Mu_StrncpyLong

```
#define Mu_StrncpyLong(note, string, name, place)\
do{\
    if(!xmlStrcmp(name, (const xmlChar*)place))\
```

```
        note = (unsigned long)atoi(string);\
}while(0)
```

6.2. XMLs

6.2.1. Server Info rmation Description

00_ServerInfo.XML

6.2.2. Pre_Login

01_PreLogin.xml

该 XML 文件作为信令，作为服务器与设备间的交互信息，在设备调用 PreLogin 函数后，返回的 XML。

6.2.3. Login

03_After_Login.xml

该 XML 作为信令，在设备登录服务器后，服务器返回的信息；

6.2.4. Query_Status

05_After_Query.xml

该 XML 作为信令，在设备查看其状态时，由服务器返回的信息；

6.2.5. Query_FDDF

07_After_QueryFDDF.xml

该 XML 作为信令，返回设备查看 2xx 类型节目的 FDDF 信息；

6.2.6. Update_Status

09_After_Update.xml

该 XML 作为信令，返回设备查看 2xx 类型节目的 FDDF 信息；

6.2.7. FDDF

FDDF_1.xml
该 XML 描述待下载节目的分段；

6.2.8. FileInfo

file_info.xml
该 XML 用于描述文件信息；

6.2.9. Resuming

resuming.xml
该文件存储正在下载或是未下载完成的文件名，用于断点续传时，确定节目；

6.2.10. DeviceInfo

deviceinfo.xml
存储设备相关信息；

6.2.11. Update_info

update_SRV.xml
该文件用于存储更新固件的信息，在开机启动时，将用该信息确定更新固件是否完整；