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	2008-3-11	
102	修改		沈胜文	2008-3-12	
103	修改		沈胜文	2008-3-16	
104	修改		沈胜文	2008-4-11	
105	修改		王艳君	2008-5-23	

Table of Content 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 解析进行的详细设计说明。本文档将尽可能详尽地说明 XML 解析的设计与开发，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）的解析。但是本次所开发的软件，并不局限于本项目。

设计的出发 点：该解析部分独立于 MuFTAD 项目，可以方便地服务于其他项目。

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 和其他需要利用 XML 解析的项目；

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。
XPath	XPath 是为寻址 XML 文档部件而提供的一种语法和数据模型的语言。
MuFTAD	软件名称 Microunit France-Telecom Auto-Download
libxml	Libxml 是一个有免费许可的用于处理 XML、可以轻松跨越多个平台的 C 语言

	库。

1.4. Reference Material 参考资料

Xmltutorial.pdf
Libxml 简单教程.pdf
XML Parse tutaril.pdf

2. Rules 规则

2.1. Name Rules 命名规则

XML 解析是立足于 MuFTAD 项目，但是着眼于服务所有待开发的项目。因此该程序的设计考虑使用扩展库的形式来实现。

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. Libaries 库

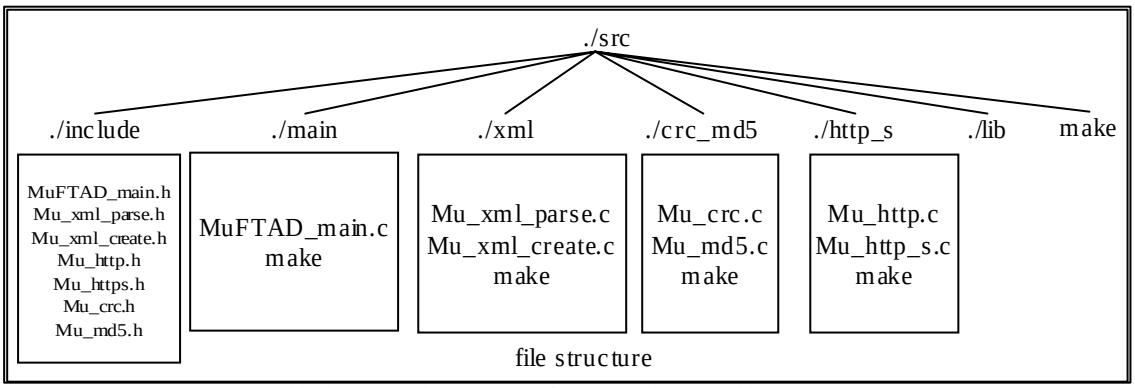
待开发的扩展库是在 libxml 的基础上进行的，该库是一个有免费许可的用于处理 XML、可以轻松跨越多个平台的 C 语言库。

在大多数 Linux 操作系统上均已安装该库。

2.5. File Structure 文件结构

[注意]:

1、此为整个项目的文件组织方式;



./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

该程序由二部组成：

一、主体函数部分：该部分获取文件名参数及返回指针参数。在 XML 文件中按 XPath 进行查找，定位到复合要求的待解析部分；

二、信息存储部分：该部分取得主体函数部分解析得到的信息，将其存储在特定的结构中，并以指针的形式返回；

[注意]:

1、上述二个部分可以参考《XML 解析概要设计》;

3.2. Routines List 函数列表

```
int Mu_Parse (int type, const int fd, const char *xpath, void * *back);

int Mu_ParseXML(int type, xmlDocPtr Doc, void * *back);

int Mu_ParseXpathXML(int type,    xmlDocPtr Doc,
                                  const char *xpath,    void * *back);

int Mu_ParseSingle(xmlXPathObjectPtr Ptr, void **back);
int Mu_ParseFDDFList(xmlXPathObjectPtr Ptr, FddfListHeadPtr back);
int Mu_ParseMulti(xmlXPathObjectPtr Ptr, ServerHeadPtr back);
int Mu_ParseMulti2xx(xmlXPathObjectPtr Ptr, FddfStatusHeadPtr back);
int Mu_ParseMulti4xx(xmlXPathObjectPtr Ptr, DirStatusHeadPtr back);
int Mu_ParseMultiFDDF(xmlXPathObjectPtr Ptr, FddfHeadPtr back);
int Mu_ParseEyewearADP(xmlDocPtr Doc, EyeWearADPPtr back);
int Mu_ParseEyewearFDDF(xmlDocPtr Doc, EyewearFddfPtr back);
int Mu_ParseServiceInfo(xmlDocPtr Doc, ServiceInfoPtr back);
int Mu_ParseUpdateInfo(xmlDocPtr Doc, UpdateInfoPtr back);
int Mu_ParseDeviceInfo(xmlDocPtr Doc, DeviceInfoPtr back);
int Mu_ParseResumeInfo(xmlDocPtr Doc, ResumeInfoPtr back);
int Mu_ParseFileInfo(xmlDocPtr Doc, FileInfoPtr back);
```

4. Global Description 全局描述

4.1. Global Type 全局类型

[注意]:
1、该类型可以被定义在 xml.h 中;

4.1.1. Macro 宏定义

```
#define MUXML_LENGTH_MAX 250
```

用于在申请内存时，或是在存储已解析的信息时，最多可存储的空间。限制该长度是防止某个描述字段占用过多空间，而耗尽内存。

[注意]:
1、该定义不能过短，以防止对正常的、不可截断信息进行误操作;

#define MUMULTI_LIST_MAX 20
用于定义解析 2xx 和 4xx 信息时，一次可解析的最多 item 个数。

#define PARSE_SINGLE 1
#define PARSE_ FDDF_LIST 2

#define PARSE_MULTI 3
#define PARSE_MULTI_2XX 4
#define PARSE_MULTI_4XX 5
#define PARSE_MULTI_FDDF 6

#define PARSE_EYEWEAR_ADP 101
#define PARSE_EYEWEAR_FDDF 102
#define PARSE_SERVICE_INFO 103
#define PARSE_UPDATE_INFO 104
#define PARSE_DEVICE_INFO 105
#define PARSE_RESUME_INFO 106
用于确定解析时对存储函数的选择；

#define CHECK_2XX(num) (((num) > 200) && ((num) < 205))
#define CHECK_4XX(num) (((num) > 400) && ((num) < 405))
#define CHECK_XPATH(num) ((num) > 100)

volatile int FDDF_2xx_total = 0;
volatile int DIR_4xx_total = 0;
在解析过多字段时，每次只解析一定量的字段，用上述两个值定位 XML 中待解析的位置；

```
#define Mu_xmlFree(name)
do{\
    if(name){\
        xmlFree(name);\
        name = NULL;\
    }\
}while(0)
```

4.1.2. Struct 结构体

参见《MuFTAD 数据结构详细设计》

4.2. Global Error 全局错误码

[注意]:
1、该错误码应该被放在一个单独的，只用来定义错误的头文件中；
2、该头文件定义为：mu_error.h

```
#define MUOK 0
#define MUNBUF -1

#define MUEXDOC -2
#define MUEXNOD -3
#define MUEXCNT -4
……
```

5. Routines Details 函数细节

5.1. Mu_ParseSingle

5.1.1. Name 函数名称

```
int Mu_ParseSingle(xmlXPathObjectPtr Ptr, void **back);
```

5.1.2. Description 函数描述

该函数由解析的处理程序 Mu_ParseXpathXML 函数调用，并且传递参数给本函数，当函数处理完成后，返回处理的状态值；

本函数处理的字段为：XML 文件中的某个字段是独一无二的，不可重复的，不可变的，解析时不会与其他字段冲突的字段，否则不可使用本函数解析；

[注意]:

1、此处说的独一无二是指，在用 xPath 路径描述 XML 中字段时，路径唯一确定一信息;

本函数取得的字符串信息，由 malloc 或是 calloc 分配的内存空间存储。该内存空间将会在函数调用结束后，**一直常驻内存，除非显式地调用 free 函数释**

放该空间；

5.1.3. Function 功能

本函数从 XPath 指针 Ptr 处取出需要的信息，并且分配静态地址空间，以指针的地址 back 返回。

若所求内容超出 MUXML_LENGTH_MAX，需要将其截断，在截断时，应该考虑到对中文的支持。**截断的对像，通常为描述信息，该部分若采用中文，则应该避免对一个中文码的一半的截断**；

back 是一个无类型的二级指针，在该类型中，为字符串空间的地址。

[注意]:

1、返回的指针处的内容，为一个字符串地址，不管待解析的部分是数字还是字母。如果待解析部分是数字，由调用者进行字符串到数字的转换;

5.1.4. Capability 性能

该函数申请的内存空间不能超过 MUXML_LENGTH_MAX；

5.1.5. Input 输入

该函数接受二个输入项：ptr 指针，和一个无类型二级指针。

ptr 指针：该指针是 Mu_ParseXpathXML 函数找到的与待查询内容相符合的结构，通过该结构，可以取得待查询内容的相关信息。

[注意]:

1、具体 xmlXPathObjectPtr 结构，可以在 http://xmlsoft.Org 中查询。

```
struct _xmlXPathObject {
    xmlXPathObjectType type
    xmlNodeSetPtr nodesetval
    int boolval
    double floatval
    xmlChar * stringval
    void * user
    int index
```

```
    void * user2
    int index2
    }

struct _xmlNodeSet {
    int nodeNr : number of nodes in the set
    int nodeMax : size of the array as allocated
    xmlNodePtr * nodeTab : array of nodes in no particular order @
    }

struct _xmlNode {
    void * _private : application data
    xmlElementType type : type number, must be second !
    const xmlChar * name : the name of the node, or the entity
    struct _xmlNode * children : parent->childs link
    struct _xmlNode * last : last child link
    struct _xmlNode * parent : child->parent link
    struct _xmlNode * next : next sibling link
    struct _xmlNode * prev : previous sibling link
    struct _xmlDoc * doc : the containing document End of common p
    xmlNs * ns : pointer to the associated namespace
    xmlChar * content : the content
    struct _xmlAttr * properties : properties list
    xmlNs * nsDef : namespace definitions on this node
    void * psvi : for type/PSVI informations
    unsigned short line : line number
    unsigned short extra : extra data for XPath/XSLT
    }
```

back：无类型二级指针，该指针由调用者指定，该指针返回查找到的信息；

[注意]:

*1、该**指针必须为二级指针**，否则会造成内存空间泄露;*

2、本函数中，back 是一个指向字符串的地址，对其内容的操作与释放由调用者完成;

5.1.6. Output 输出

函数运行状态码；

在程序运行出错时，以错误状态返回，并应该能输出相应的说明信息，建议使用 write 函数输出。

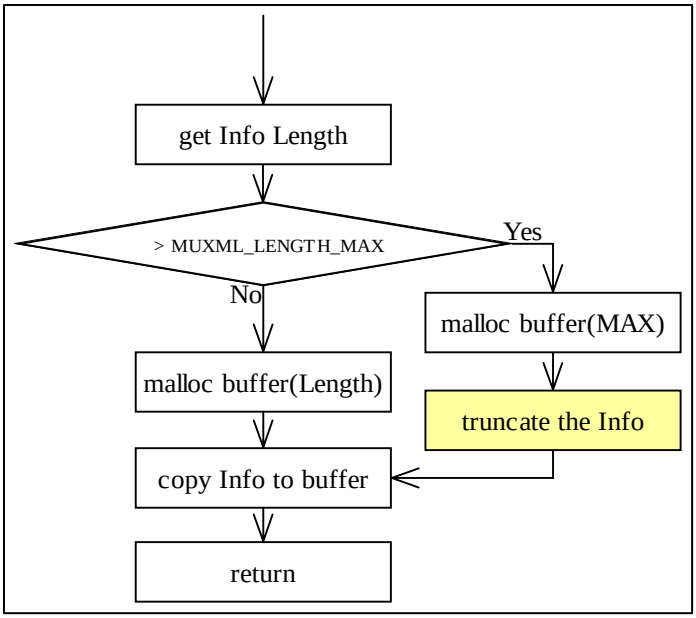
要求能处理的错误，包括取值错、申请内存空间错和拷贝错等情况。

back 指向的内存区域存储一串字符串信息，该区域是由函数静态分配的，占用系统的堆内存空间；

5.1.7. Arithmetic 算法

略

5.1.8. Process Flow 处理流程



[注意]:

1、黄色部分由其他函数提供;

get info Length：取得 XPath 指针所指向地址的信息的长度；

malloc buffer：申请一块足够的地址空间存储 get info length 长度的字串，通常该申请长度为要存储的最长字段长度加 1；

copy Info to buffer：将查找的信息拷贝至申请的地址空间；

truncate the info：对字段项截断，该功能由函数 Mu_TruncateInfo 提供；

5.1.9. Pseudocode 伪代码

```
int Mu_ParseSingle(xmlXPathObjectPtr Ptr, void **back)
{
    xmlNodePtr curNode = NULL;
    xmlChar *muvalue = NULL;
    int length;
    //backvalue can use global value;

    xmlNodeSetPtr munodeset = Ptr->nodesetval;

    Mu_ErrorPrint(). Because we parse the xml element is single!!!
    //if the munode is not NULL, we also just process ONE
    if((munodesetptr == NULL)
        ||((munodeset->nodeNr) == NULL)
        ||(( curNode = munodeset->nodeTab[0]) == NULL)
        ||(( muvalue = xmlNodeGetContent(curNode)) == NULL){
        xmlFree(muvalue);
        return MUEXSIN;
    }

    length = strlen(muvalue);
    if((length > MUXML_LENGTH_MAX){
        &&(length = Mu_TruncateInfo(muvalue)) < 0)){
        return MUEXSIN;
    }

    if(((char *)*back = (char *)malloc(length + 1)) == NULL){
        Mu_ErrorPrint();
        return MUNBUF;
    }
    memset(*back, 0, length + 1);

    strncpy(*back, muvlaue, length);
    xmlFree(muvalue);

    return backvalue;
}
```

5.1.10. Interface 接口

本函数为主体解析函数提供解析接口，同时，在解析的时候也需要利用其他函数：

Mu_Truncate（）。

[注意]:

1、详细说明请参考附录;

5.1.11. Malloc 存储分配

在函数运行过程中，需要分配一段内存区域，用于存储已解析的信息。分配的长度由所解析的字串长度与头文件定义的 MUXML_LENGTH_MAX 共同决定。

对于超过 MUXML_LENGTH_MAX 的字段，应该对其进行截断。

5.1.12. Restrict 限制

略

5.1.13. Test 测试

对本函数的测试，必须借助于主函数，并且使用正确的调用参数；

测试时，需要存在完整的 XML 文件；

对于解析的结果存储于内存中，因此为了直观地了解解析结果，应该写一个小的测试程序，输出内存中的内容；

需要测试程序对内存的影响；

5.14. Unsolve 未解决情况

略

5.2. Mu_ParseFDDFList

5.2.1. Name 函数名称

int Mu_ParseFDDFList(xmlXPathObjectPtr Ptr, FddfListHeadPtr back);

5.2.2. Description 函数描述

该函数由解析的 Mu_ParseXpathXML 程序调用，并且传递参数给本函数，当函数处理完成后，返回处理的状态值；

本函数处理的字段为：XML 文件中的某个字段是独一无二的、不重复的，解析时不会与其他字段冲突的字段,但是该字段中子项是可变的；

[注意]:

1、此处说的独一无二是指，在用 xPath 路径描述 XML 中字段时，路径唯一确定一信息;

2、具体的待解析 XML 文件如 07_After_QueryFDDF.xml 所示;

本函数取得的信息，由 malloc 或是 calloc 分配的内存空间存储，并被构建成一个链表形式存储；

链表节点的内存空间将会在函数调用结束后，一直常驻内存，除非显式地调用 free 函数释放该空间；

back 指针指向一个头节点，该头结点由调用者分配空间；

5.2.3. Function 功能

本函数从 XPath 指针 Ptr 处取出需要的信息，并且分配静态地址空间，添加到 back 所指的链表结构中；

若所求内容超出 MUXML_LENGTH_MAX，直接放弃；

back 是一个结构指针，参考 FddfListHeadPtr 结构；

5.2.4. Capability 性能

该函数申请的内存空间不能超过 MUXML_LENGTH_MAX；

5.2.5. Input 输入

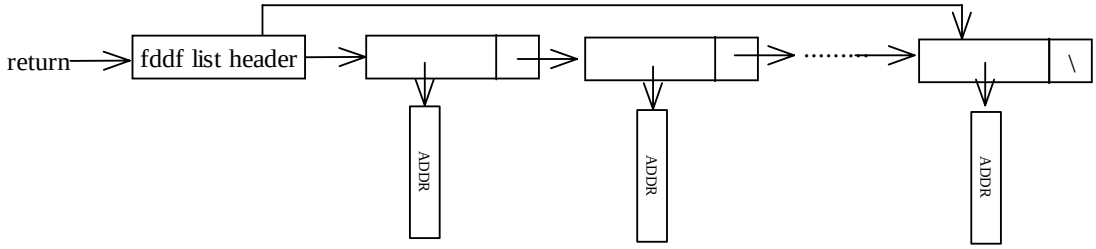
同 5.1.5.中说明

5.2.6. Output 输出

同 5.1.6 中说明

5.2.7. Arithmetic 算法

函数在完成解析时，需要充分考虑节点的存储，简图如下：



新节点插入时，应该将该节点插入到链表的尾部。

[注意]:

1、该链表的具体结构请参考全局类型(4.1 节)中的 Fddf_Lis_Head 和 Fddf_List_Node;

2、头结点中的指针 last，方便插入节点;

3、上述结构简略与实际简略有一些出入，完整结构图参考《MuFTAD 数据结构详细设计》中的 FDDF_List （5.2. 章节);

5.2.8. Process Flow 处理流程

同 5.3.8.中说明

5.2.9. Pseudocode 伪代码

[注意]:

1、调用者必须先调用 Mu_FddfListHeadCreate()函数创建头结点;

2、头结点地址作为全局值存储;

```
int Mu_ParseFDDFList(xmlXPathObjectPtr Ptr,   FddfListHeadPtr back)
{
```

```
xmlNodePtr curNode = NULL;
xmlChar *muvalue = NULL;
int length;
int i = 0;
FddfListNodeptr node = NULL;

xmlNodeSetPtr munodeset = Ptr->nodesetval;
if(munodeset == NULL){
    Mu_ErrorPrint();
    Return MUEEOR;
}

//init the header of fddf list
Mu_FddfListHeadClear(back);

if(((curNode = munodeset->nodeTab[0])== NULL)
    ||((curNode = curNode->child) == NULL)){
    Mu_ErrorPrint();
    return MUEXFDL;
}
if(((muvalue = xmlNodeGetContent(curNode)) == NULL)
    ||(strlen(muvalue) != 3)){
    Mu_ErrorPrint();
    return MUEXFDL;
}

int num = atoi(muvalue);
if(!(CHECK_FDDF_LIST(num))){
    retun MUEXFDL;
}

//stored the value of 'Type'
if(Mu_FddfHeadStrncpy(back, muvalue, curNode-> name) < 0)
    return MUEXFDL;
Mu_xmlFree(muvalue);

//get 6 elements: text, data, data1,data2,data3,data4
while(i < 6){
    if(((curNode = curNode->child) == NULL)
        ||(muvalue = xmlNodeGetContent(curNode) == NULL)
        ||( Mu_FddfHeadStrncpy(back, muvalue, curNode-> name) < 0){
```

```
            Mu_xmlFree(muvalue);
            return MUEXFDL;
        }

        Mu_xmlFree(muvalue);
        i ++;
    }

    //get the path of the FDDF,stored in the server
    int count = back->size - i + 2;
    while(count - i -1){
        if(((curNode = curNode->child) == NULL)
            ||((muvalue = xmlNodeGetContent(curNode)) == NULL)
            ||( Mu_FddfListNodeCreate (&Node) < 0)
            ||( Mu_FddfListNodeStrncpy (Node, muvalue, curNode->name) < 0)
            ||( Mu_FddfListNodeInsert (back, Node) < 0 ){
            Mu_xmlFree(muvalue);
            return MUEXFDL;
        }

        Node = NULL;
        Mu_xmlFree(muvalue);
    }

    retrun MUOK;
}
```

5.2.10. Interface 接口

本函数为主体解析函数提供解析接口，同时，在解析的时候也需要利用其他函数：

Mu_FddfListHeadCreate (back);
Mu_FddfListHeadInit(back);
Mu_FddfListHeadStrncpy (back, muvalue, curNode-> name);
Mu_FddfListNodeCreate (&Node);
Mu_FddfListNodeStrncpy (Node, muvalue, curNode-> name);
Mu_FddfListNodeInsert (back, Node);

[注意]:

1、详细说明请参考《MuFTAD 数据结构详细设计》;

5.2.11. Malloc 存储分配

在本函数运行的过程中，**不需要**申请内存空间以存储 FddfListHead 结构、该结构由用户自己初始化；

但是函数**必须**申请内存空间以存储链表节点 FddfListNode 结构和各获得的信息段；

以上各部分均在动态内存区存储，本函数不会自动释放该空间，需要调用者显示地调用 free 释放相应的信息（包括节点本身）；

[注意]:
1、头结点在整个项目中将会是一个全局变量，其内存空间不会被释放;

5.2.12. Restrict 限制

略

5.2.13. Test 测试

对本函数的测试，必须借助于主函数，并且使用正确的调用参数；

测试时，需要存在完整的 XML 文件；

对于解析的结果存储于内存中，因此为了直观地了解解析结果，应该写一个小的测试程序，输出内存中的内容；

需要测试程序对内存的影响；

5.2.14. Unsolve 未解决情况

略

5.3. Mu_ParseMulti(DELAY)

5.3.1. Name 函数名称

int Mu_ParseMulti(xmlXPathObjectPtr Ptr, ServerHeadPtr back);

5.3.2. Description 函数描述

该函数由解析的主处理程序调用，并且传递参数给本函数，当函数处理完成后，返回处理的状态值；

本函数处理的字段，在 XML 文件中存是重复的、可变的，解析时会得到一连串的相同字段信息；

[注意]:
1、此处说的重复是指，在用 xPath 路径描述 XML 中字段时，某个路径可以确定多个信息;
2、本函数的提出主要是针对 Server Information description XML 文件中的<addr>字段;
3、本函数并不一定会用于解析<addr>字段信息，但是扩展库仍可提供该函数以供其他用处;

本函数取得的信息，由头结点 back 中的指针维护，头结点由调用者初始化；

存储信息的链表节点，由 malloc 或是 calloc 分配内存空间。该内存空间将会在函数调用结束后，一直常驻内存，除非显式地调用 free 函数释放该空间。

5.3.3. Function 功能

本函数从 XPath 指针 Ptr 处，开始取出一连串需要的信息，并且分配内存空间，构建链表结构，存储信息，并将节点添加到以 back 指针为头结点的链表中；

函数在解析中，也会更新头结点的信息；

若所求内容超出 MUXML_LENGTH_MAX，为了安全上的考虑，应该**直接放弃**存储该信息。

back 是一个服务器地址链表的头结点指针。

5.3.4. Capability 性能

该函数针对每一项信息申请的内存空间不能超过 MUXML_LENGTH_MAX；

5.3.5. Input 输入

同 5.1.5 中说明

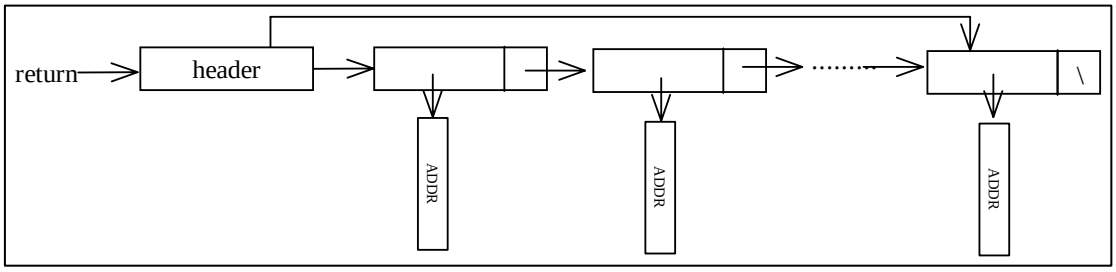
5.3.6. Output 输出

同 5.1.6.说明

back 为一链表头结点，该表头指向一个存储地址的链表节点（ServerHead）。用于存储服务器地址普通节点及地址信息的区域是由用户动态分配的，占用系统的堆空间；

5.3.7. Arithmetic 算法

函数在完成解析时，需要充分考虑节点的存储，简图如下：



新节点插入时，应该将该节点插入到链表的尾部。

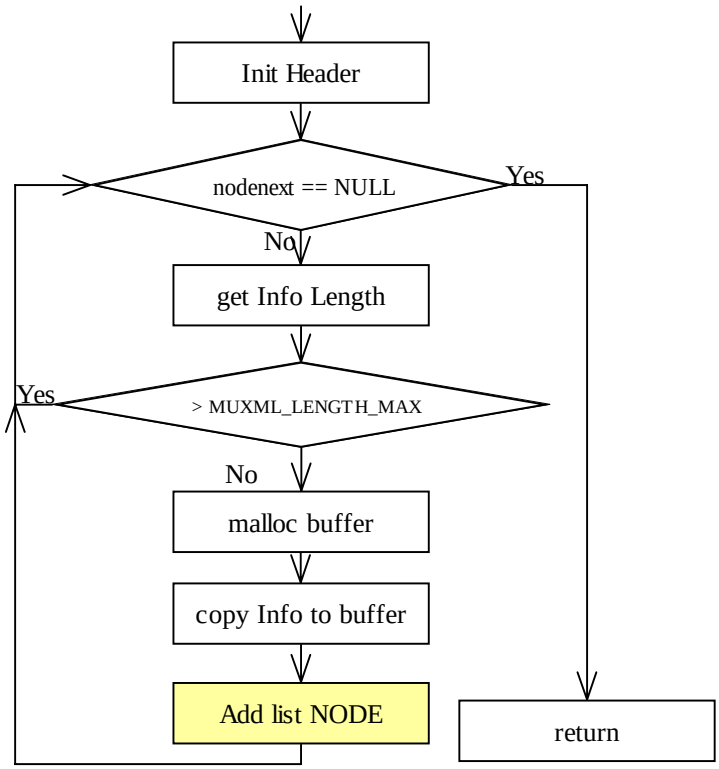
[注意]:

1、该链表的具体结构请参考全局类型(4.1 节)中的 SERVER_HEAD 和 SERVER_NODE;

2、头结点中的指针 last，方便插入节点;

3、该结构简图与实际的存储结构图有一些出入，详细可参考《MuFTAD 数据结构详细设计》;

5.3.8. Process Flow 处理流程



[注意]:

1、黄色部分由其他函数完成;

Init header：初始化链表头结点；

[注意]:

1、对于存储的服务器地址信息，不需要进行本操作，因为整个程序运行过程中，链表结构只会操作一次，不会被重复操作;

get info Length：取得 XPath 指针所指向地址的信息的长度；

add list NODE：因为所需要解析的内容在 XML 中由多个相同字段组成，解析时，需要创建链表结构来存储；

malloc buffer：申请一块足够的地址空间存储 get info length 长度的字串，通常该申请长度为 get info Length 长度加 1；

copy Info to buffer：将查找的信息拷贝至申请的地址空间；

[注意]:

1、因为该字段在 XML 文件中包含许多相同部分，所以需要遍历 XPath 所指的所有节点;

2、对于长度超过 MUXML_LENGTH_MAX 的字段，直接放弃，以防止内存耗尽;

5.3.9. Pseudocode 伪代码

```
Int Mu_ParseMulti(xmlXPathObjectPtr Ptr,    ServerHeadPtr back)
{
    xmlNodePtr curNode = NULL;
    xmlChar *muvalue = NULL;
    int length;
    int i = 0;
    int backvalue = NO_MUERROR;
    ServerNodePtr node = NULL;
    //backvalue can use global value;

    //clear the server header of list
     back->total = 0;
    back->next = back->last = NULL;

    xmlNodeSetPtr munodeset = Ptr->nodesetval;
    if(munodesetptr == NULL){
        Mu_ErrorPrint()
        Return backvaule;
    }

    Mu_ServerHeadInit(back);
    //may can be omit
    for(i; i < munodeset->nodeNr; i++){
        curNode = munodeset->nodeTab[i];
        if(curNode != NULL){
            muvalue = xmlNodeGetContent(curNode);
            if(muvalue == NULL){
                Mu_ErrorPrint();
                Backvalue = ....;
                continue;
            }

            length = strlen(muvalue);
            if(length > MUXML_LENGTH_MAX){
                Backvalue = ....;
                Break;
            }

            //create node
```

```
            Mu_ServerNodeCreate(&node);

            //copy length to buffer
            Mu_ServerNodeStrncpy(node->addr, muvalue, length);
            xmlFree(muvalue);

            Mu_ServerNodeInsert(back, node);
        }

    return backvalue;
}
```

5.3.10. Interface 接口

本函数为主体解析函数提供解析接口，同时，在解析的时候也需要利用其他函数：

Mu_ServerHeadInit();
Mu_ServerNodeCreate();
Mu_ServerNodeStrncpy();
Mu_ServerNodeInit();

[注意]:

1、详细说明请参考《MuFTAD 数据结构详细设计》;

5.3.11. Malloc 存储分配

在本函数运行的过程中，**不需要**申请内存空间以存储链表头结点 ServerHead，该区域由调用者创建；

需要申请内存空间以存储存储已查找得到的信息节点 ServerNode；

对于用于分配得到的内存空间，本函数不会自动释放该空间，需要调用者显示地调用 free 释放结点的相应的信息地址；

5.3.12. Restrict 限制

略

5.3.13. Test 测试

本函数的测试的前提条件：待解析 XML 文件中存在相同的字段项，可以查看

《doc00_SoftProtocol_0.1.1》中涉及的 Server Information Descriptor XML 文件中的 addr 字段；

必须借助于主函数，并且使用正确的调用参数；

对于解析的结果均存储于链表中，因此为了直观地了解解析结果，应该写一个小的测试程序，输出链表中的内容；

需要测试程序对内存的影响；

5.3.14. Unsolve 未解决情况

略

5.4. Mu_ParseMulti2XX

5.4.1. Name 函数名称

int Mu_ParseMulti2XX(xmlXPathObjectPtr Ptr, FddfStatusHeadPtr back);

5.4.2. Description 函数描述

该函数由 Mu_ParseXpathXML 函数调用，并且传递参数给本函数，函数处理完成后，返回处理的状态值；

本函数取得的信息，由 malloc 或是 calloc 分配的内存空间存储，再添加到 back 所指的链表中；
该内存空间将会在函数调用结束后，一直常驻内存，除非显式地调用 free 函数释放该空间；

待解析的内容在原 XML 文件中可能包含有多个相同的字段；
[注意]:
1、此处说的相同是指，在用 xPath 路径描述 XML 中字段时，某个路径可以确定多个信息;

5.4.3. Function 功能

本函数从 XPath 指针 Ptr 处，开始取出一连串需要的信息，并且分配内存空

间，构建链表节点，存储到相应的链表中；

若所解析出的内容超出 MUXML_LENGTH_MAX，为了安全上的考虑，应该直接放弃存储该信息；

函数每次解析的字段数最多为 MUMULTI_LIST_MAX；

调用者初始化一块连续的内存区域以存储 FddfStatusHead 数组 ProgramDescriptor[4]；
本函数动态分配内存空间以存储 FddfStatusNode 节点；

[注意]:
1、back 是 ProgramDescriptor 数组地址;

5.4.4. Capability 性能

该函数针对每一项信息申请的内存空间不能超过 MUXML_LENGTH_MAX；
函数每次解析的字段数最多为 **MUMULTI_LIST_MAX**；

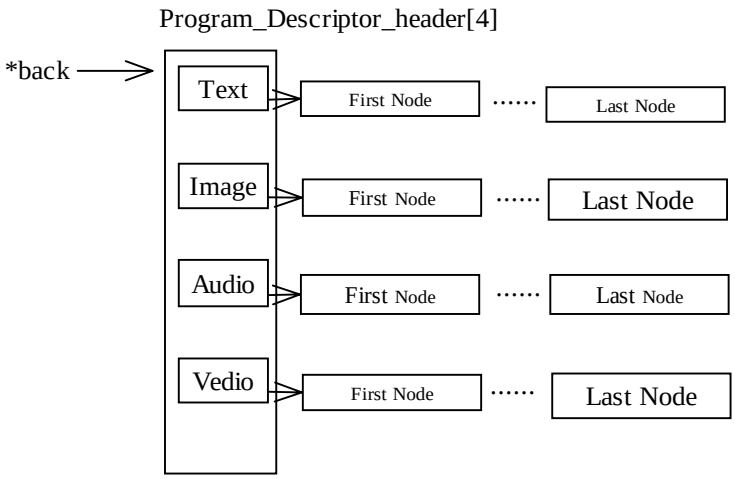
5.4.5. Input 输入

同 5.1.5 中说明

5.4.6. Output 输出

同 5.1.6.说明
back 指向在内存区域为一数组分配的地址，该数组由四个头结点组成。该区域是调用者初始化的，占用系统的堆空间；

5.4.7. Arithmetic 算法



ProgramDescriptor[4]由四个 FddfStatusHead 结构组成;

解析出的信息，构建新的节点，节点按优先级插入到链表中，头结点（FddfStatusHead）中包括相应链表的结点总数;

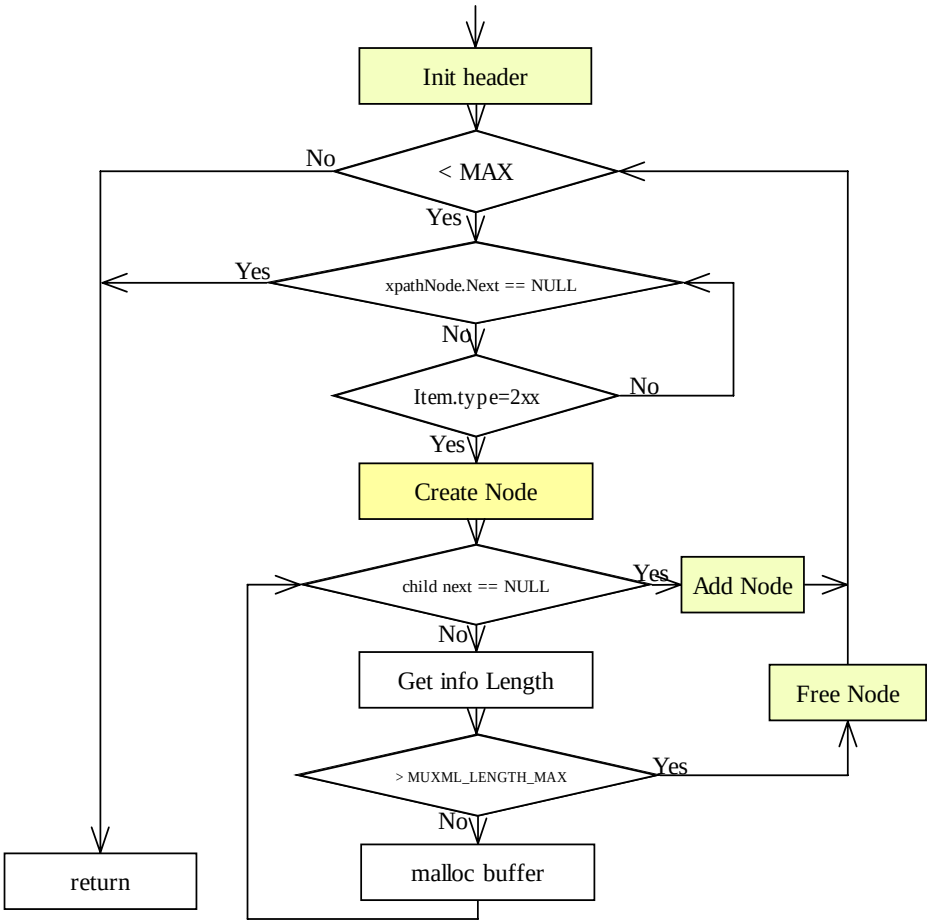
[注意]:

1、该链表的具体结构请参考《MuFTAD 数据结构详细设计》中的 FddfStatusHead 和 FddfStatusNode;

2、新建的节点按类别插入到相应链表，按优先级插入到链表中的具体位置;

该结构中的所有结点，最多只有 MUMULTI_LIST_MAX 个;

5.4.8. Process Flow 处理流程



[注意]:

1、黄色部分为调用其他函数;

2、若 Query_Status 时，返回 2xx 类中的字段返回超过可解析最大长度 MUXML_LENGTH_MAX，该字段将永远无法解析;（BUG）

Init header：初始化头结点，功能由函数 Mu_FddfStatusHeadInit 提供;

create node：创建节点，功能由函数 Mu_FddfStatusNodeCreate 提供;

add Node：将结点添加到链表中去，功能由函数 Mu_FddfStatusNodeInsert 提供;

free Node：由于解析字段里某些项不可解析时，需要将已存储的信息释放掉，并且将结点也释放掉，具体功能由函数 Mu_FddfStatusNodeDelete 提供;

5.4.9. Pseudocode 伪代码

调用者必须先用 Mu_FddfStatusHeadCreate 创建结构;

int FDDF_2xx_total

```
int Mu_ParseMultiDel2XX(xmlXPathObjectPtr Ptr,    FddfStatusHeadPtr back)
{
```

```
xmlNodePtr curNode = NULL;
xmlChar *muvalue = NULL;

int length;
int total_parse = 0;
volatile int insertmark;
FddfStatusNodePtr node = NULL;

//backvalue can use global value;

xmlNodeSetPtr munodeset = Ptr->nodesetval;
if(munodesetptr == NULL){
    Mu_ErrorPrint()
    return MUEX2XX;
}

Mu_FddfStatusHeadClear(back);
for(insertmark = 1;
    ((total_parse < MUMULTI_LIST_MAX)
    && (FDDF_2xx_total < munodeset->nodeNr));
    FDDF_2xx_total ++, total_parse ++, insertmark = 1)
{
    if((curNode = munodeset->nodeTab[FDDF_2xx_total]) == NULL){
        Mu_ErrorPrint()
        continue;
    }

    //jump to the first child
    if(((curNode = curNode->child) == NULL)
        ||(( muvalue = xmlNodeGetContent(curNode)) == NULL)
        ||(strlen(muvalue) != 3))){

        return MUEX2XX;
    }

    if(!(num = atoi((char*)muvalue)) || (!(CHECK_2XX(num))))
    {
                Mu_xmlFree(muvalue);
                continue;
    }
    if(Mu_FddfStatusNodeCreate(&node) < 0
```

```
            || Mu_FddfStatusNodeStrncpy(node, muvalue, curNode-> name) < 0){
            Mu_xmlFree(muvalue))
            return MUEX2XX;
        }
        Mu_xmlFree(muvalue);
        //get ride of one Item.Type = 2xx
        while(curNode->next != NULL){
            curNode = curNode->next;
            if(NULL == (muvalue = xmlNodeGetContent(curNode))){
                return MUEX2XX;
            }

            if(strelne(muvalue) > MUXML_LENGTH_MAX){
                Mu_FddfStatusNodeDelete(Node);
                insertmark = 0;
                Mu_xmlFree(muvalue);
                //Other processes!!!According to need!!!
                break;
            }

            if( Mu_FddfStatusNodeStrncpy(Node, muvalue, curNode->name)< 0){
                Mu_FddfStatusNodeDelete(Node);
                Mu_xmlFree(muvalue);
                //Other processes!!!According to need!!!
                return MUEX2XX;
            }
            Mu_xmlFree(muvalue);
        }

        if((insertmark)&& (Mu_FddfStatusNodeInsert(back, Node) < 0))
            return MUEX2XX;
    }

    //clear the counter for FDDF_2xx_total
    if(FDDF_2xx_total == munodeset->nodeNr)
        FDDF_2xx_total = 0;

    retrun MUOK;
}
```

5.4.10. Interface 接口

本函数为主解析函数提供解析接口，同时，在解析的时候也需要利用其他函数：

Mu_FddfStatusHeadInit();
Mu_FddfStatusNodeCreate();
Mu_FddfStatusNodeStrncpy();
Mu_FddfStatusNodeDelete();
Mu_FddfStatusNodeInsert();

[注意]:

1、详细说明请参考《MuFTAD 数据结构详细设计》;

5.4.11. Malloc 存储分配

在本函数运行的过程中，**不需要**申请内存空间以存储 FddfStatus_Head 结构数组，但是**需要**申请内存空间以存储链表节点 FddfStatusNode 结构和各查找的信息段;

以上各部分均以动态内存区存储，本函数不会自动释放该空间，需要调用者显示地调用 free 释放结点的相应的信息地址;

5.4.12. Restrict 限制

略

5.4.13. Test 测试

本函数的测试的前提条件：待解析 XML 文件中存在相同的字段项，并且该字段项还应该有子项存在。

[注意]:

1、测试时可以参考《doc00_SoftProtocol_0.1.1》中 Query_status 后返回的 XML 文件中的 Item.Type = 2xx 字段;

借助于主函数，并且使用正确的调用参数;

对于解析的结果均存储于链表中，因此为了直观地了解解析结果，应该写一个小的测试程序，输出链表中的内容;

需要测试程序对内存的影响；

5.4.14. Unsolve 未解决情况

略

5.5. Mu_ParseMulti4xx

5.5.1. Name 函数名称

int Mu_ParseMulti4XX(xmlXPathObjectPtr Ptr, DirStatusHeadPtr back);

5.5.2. Description 函数描述

同 5.3.2 中说明

5.5.3. Function 功能

本函数从 XPath 指针 Ptr 处，开始取出一连串需要的信息，并且分配内存空间，构建链表节点，存储到相应的链表中。

函数每次解析的字段数最多为 MUMULTI_LIST_MAX；

调用者分配一块连续的内存区域以存储 DirStarusHead 数组 DIRfileDownload[4]，并初始化其元素；
函数在运行过程中，动态分配内存空间以存储 DirStatusNode 节点和 DirDownloadAddr 节点；

[注意]:
1、back 是 DirStatusHead 类型数组 DIRfileDownload 的地址;

5.5.4. Capability 性能

同 5.3.4. 中说明

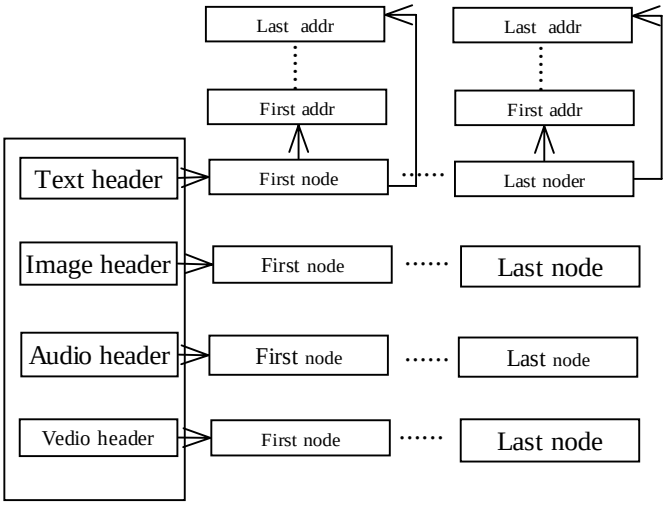
5.5.5. Input 输入

同 5.1.5 中说明

5.5.6. Output 输出

同 5.3.6.说明

5.5.7. Arithmetic 算法



DIRfile_Download[4]

DIRfileDownload4]由四个 DirStatusHead 结构组成；

解析出的信息，构建新的节点按类型插入相应类型的链表中，按优先级插入到链表中去，头结点(header)中包括相应链表的结点总数。

各节点（node）同时也是地址链表的头结点，该节点中包括地址总数和和二个指针，其中一个指向地址链表第一个节点，另一个指向地址链表的最后一个节点；

地址节点（Addr）用来存储下载文件的方式和地址信息，地址链表按解析顺序依次存放在地址链表的尾部；

[注意]:

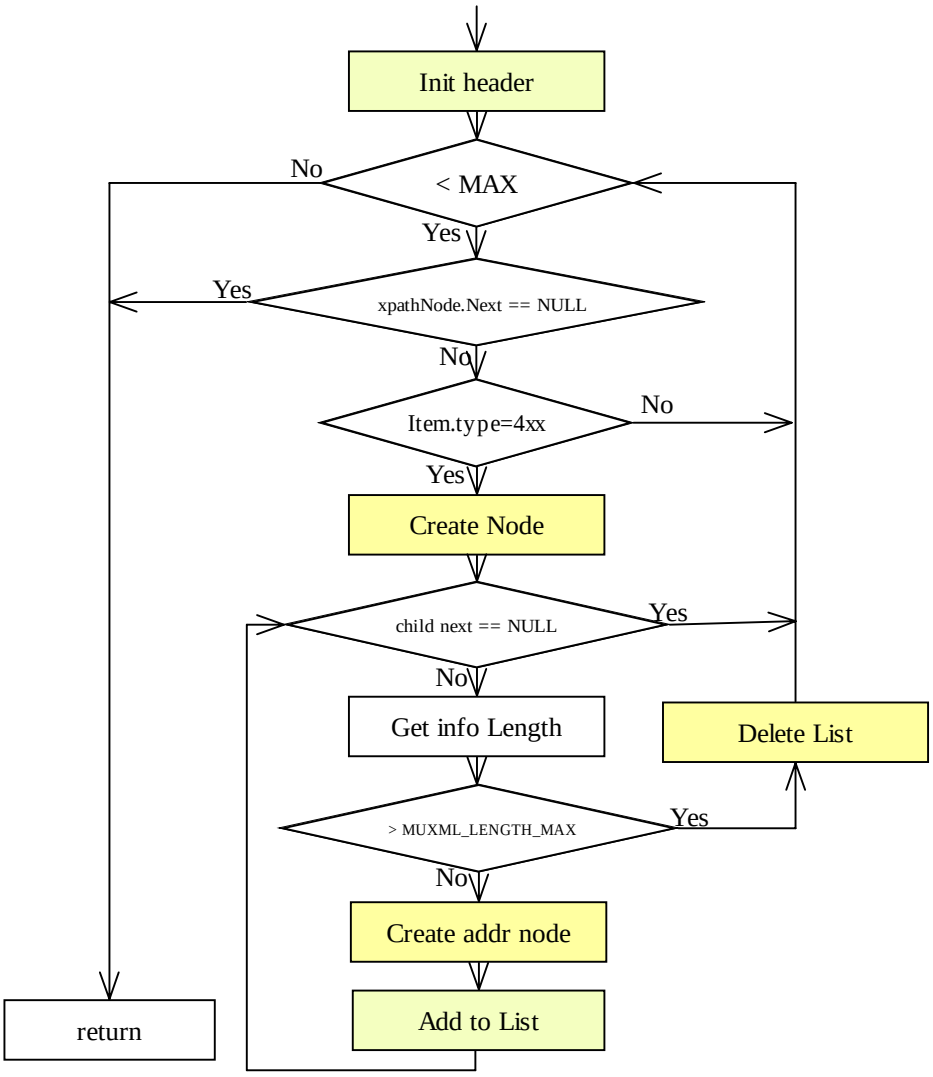
1、该链表的具体结构请参考全局类型(4.1 节)中的 DirStatusHead、DirStatusNode 和 DirDownloadAddr;

2、新建的类型节点按类别，按优先级插入到链表中;

3、新建地址节点，按解析顺序依次加入到链表尾部;

对于一种类型的链表，其结点最多只有 MUMULTI_LIST_MAX 个。

5.5.8. Process Flow 处理流程



[注意]:

1、黄色部分为调用其他函数;

Init header：初始化 DirStatusHead 数组，该功能由函数 Mu_DirStatusHeadInit 完成；

crete node：创建节点，以存储与一个 item.type=4xx 相关的信息，该功能由函数 mu_DirStatusNodeCreate 完成；

create addr node：创建地址节点，以存储一个路径或是地址相关信息，该功能由函数 Mu_DirDownloadAddrCreate 完成；

add to list：将新创建的 Addr 节点加入到地址链表中去，该功能由函数 Mu_DirDownloadAddrInsert 实现；

delete List：删除链表，当解析一个 addr 发生错误时，释放掉整个链表，但是不释放与之相关的头节点，以方便在确认时，向服务器发送失败确认。该功能由函数 Mu_DirDownloadAddrDeleteList 完成；

5.5.9. Pseudocode 伪代码

调用者必须先用 Mu_DirStatusHeadCreate 创建结构
DIR_4xx_total

```
int Mu_ParseMulti4XX(xmlXPathObjectPtr Ptr, DirStatusHeadPtr back)
{
    int length;
    int total_parse;
    //backvalue can use global value;

    xmlNodePtr curNode = NULL;
    xmlNodePtr delNode = NULL;
    xmlChar *muvalue = NULL;
    DirStatusNodePtr node = NULL;
    DirDownloadAddrPtr addrnode = NULL;

    xmlNodeSetPtr munodeset = Ptr->nodesetval;
    if(munodesetptr == NULL){
        return MUEX4XX;
    }

    Mu_DirStatusHeadClear(back);
    for(total_Parse = 0;
        (total_parse < MUMULTI_LIST_MAX)
        && (DIR_4xx_total < munodeset->nodeNr);
        DIR_4xx_total ++, total_parse ++){
        if((curNode = munodeset->nodeTab[DIR_4xx_total]) == NULL){
            continue;
        }
        delNode = curNode;

        //jump to the first child
```

```
if(((curNode = curNode->child) == NULL)
    ||((muvalue = xmlNodeGetContent(curNode)) == NULL)
    ||(strlen(muvalue) != 3)
    ||(!( num = atoi(muvalue)))
    ||(!(CHECK_4XX(num))){

    fprintf(stdout, “the Type is ERROR, jump to next!\n”);
    Mu_xmlFree(muvalue);
    continue;
}

if(Mu_DirStatusNodeCreate(&node) < 0
    || Mu_DirStatusNodeStrncpy(node, muvalue, curNode->name) < 0){

    fprintf(stdout, “copy OR create node error!\n”);
    Mu_xmlFree(muvalue);
    Mu_DirStatusNodeDelete(Node);
    return MUEX4XX;
}

Mu_xmlFree(muvalue);
 //get ride of one Item.Type = 4xx
 curNode = curNode->next->nextt;
 //skip two novalid segment
 //jump “size” and “text”
 int mark = 0; //get node parameters

 while(curNode->next != NULL){
    curNode = curNode->next;

    if(mark < 3){
        if((muvalue = xmlNodeGetContent(curNode)) == NULL)
            || (strelne(muvalue)) > MUXML_LENGTH_MAX)){

            Mu_xmlFree(muvalue);
            Mu_DirStatusNodeDelete(Node);
            //Other processes!!!According to need!!!
            break;
        }
        if(Mu_DirStatusNodeStrncpy(Node, muvalue, curNode-> name) <
    0){
```

```
                Mu_xmlFree(muvalue);
                Mu_DirStatusNodeDelete(Node);
                //Other processes!!!According to need!!!
                return MUEX4XX;
            }

            Mu_xmlFree(muvalue);
            mark ++;
            continue;
        }

        if((muvalue = xmlNodeGetContent(curNode)) == NULL)
            ||( Mu_DirStatusNodeInsert (back, Node) < 0)
            ||((length = strelne(muvalue)) > MUXML_LENGTH_MAX){

            Mu_xmlFree(muvalue);
            Mu_DirDownloadAddrDeleteList(Node);
            //can't free the header, just free node;
            //Other processes!!!According to need!!!
            break;
        }

        if(( Mu_DirDownloadAddrCreate(&addrnode) < 0)
            ||(  Mu_DirDownloadAddrStrncpy(addrnode,  muvalue,  curNode->
    name) < 0)
            ||( Mu_DirDownloadAddrInsert(Node, addrnode) < 0)){

            Mu_xmlFree(muvalue);
            Mu_DirDownloadAddrDeleteList(Node);
            return MUEX4XX;
        }

        Mu_xmlFree(muvalue);
        Node = NULL;
        addrnode = NULL;
    }
}

//clear the counter for DIR_4xx_total
if(DIR_4xx_total == munodeset->nodeNr)
    DIR_4xx_total = 0;
```

retrun **MUOK**;
}

5.5.9. Interface 接口

本函数为主解析函数提供解析接口，同时，在解析的时候也需要利用其他函数：

Mu_DirStatusHeadCreate();
Mu_DirStatusHeadInit();
Mu_DirStatusNodeCreate();
Mu_DirStatusNodeStrncpy();
Mu_DirDownloadAddrStrncpy();
Mu_DirStatusNodeDelete();
Mu_DirStatusNodeInsert();
Mu_DirDownloadAddrDeleteList();
Mu_DirDownloadAddrCreate();
Mu_DirDownloadAddrInsert();

[注意]:

1、详细说明请参考《MuFTAD 数据结构详细设计》;

5.5.10. Malloc 存储分配

在本函数运行的过程中，**不需要**申请内存空间以存储指针数组(DirStatus_Head)，该部分由调用者完成并初始化；

但是本函数**需要**申请内存区域以存储链表节点（DirStatusNode 与 DirDownloadAddr）和已查找得到的信息；

对于以上信息，本函数不会自动释放该空间，需要调用者显示地调用 free 释放结点的相应的信息地址；

5.5.11. Restrict 限制

略

5.5.12. Test 测试

本函数的测试的前提条件：待解析 XML 文件中存在相同的字段项，并且该字段项还应该有子项存在。

[注意]:

1、 测试时可以参考《doc00_SoftProtocol_0.1.1》中 Query_status 后返回的 XML 文件中的 Item.Type = 4xx 字段。

对本函数的测试，必须借助于主函数，并且使用正确的调用参数；

对于解析的结果均存储于链表中，因此为了直观地了解解析结果，应该写一个小的测试程序，输出链表中的内容；

需要测试程序对内存的影响；

5.5.13. Unsolve 未解决情况

如果 4xx 中某一类型在 MUXML_LIST_MAX 内解析得到的 addr 链表地址很多，也可能造成内存耗尽的可能。

但是该问题没有在本函数中解决，可能对以后的应用带来 BUG。该部分提到的字段的多少由服务器端来保证不会过量。

5.6. Mu_ParseMultiFDDF(DELAY)

5.6.1. Name 函数名称

```
int Mu_ParseMultiFDDF (xmlXPathObjectPtr Ptr,
                       FddfHeadPtr back);
```

5.6.2. Description 函数描述

该函数由解析的主处理函数调用，并且传递参数给本函数，函数处理完成后，返回处理的状态值。

本函数取得的信息，由 malloc 或是 calloc 分配的内存空间存储。该内存空间将会在函数调用结束后，一直常驻内存，除非显式地调用 free 函数释放该空间。

待解析的内容在原 XML 文件中包含有多个相同的字段；

[注意]:

1、此处说的相同是指，在用 xPath 路径描述 XML 中字段时，某个路径可以确定多个信息;

5.6.3. Function 功能

本函数从 XPath 指针 Ptr 处，开始取出一连串需要的信息，并且分配内存空间，构建链表节点，存储到链表中。

函数**不需要**分配一块动态内存区域以存储 FddfHead 头结点，对该头结点的创建以及初始化，由调用者完成；
函数在运行过程中，**需要**分配内存空间以存储 FddfNode 节点；

5.6.4. Capability 性能

同 5.3.4.中说明

5.6.5. Input 输入

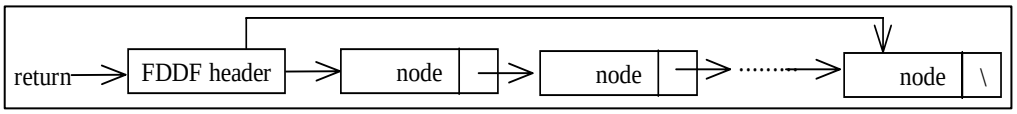
同 5.1.5 中说明

5.6.6. Output 输出

同 5.1.6.说明

back 为一链表表头(FddfHead)，该表头指向一个存储地址的链表节点（FddfNode)。后者的存储区域是由函数动态分配的，占用系统的堆空间；

5.6.7. Arithmetic 算法



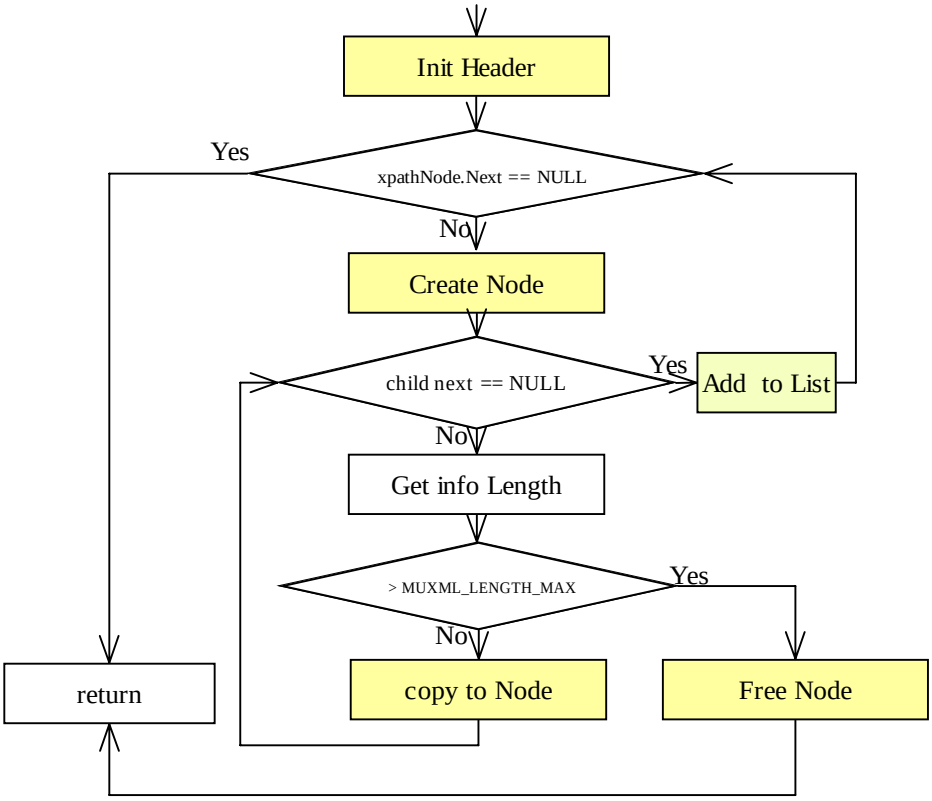
解析出的信息，构建新的节点，地址链表按解析顺序依次放在地址链表的尾部；

[注意]:

1、该链表的具体结构请参考全局类型(4.1 节)中的 Fddf_Head 和 Fddf_Node;

2、新建地址节点，按解析顺序依次加入到链表尾部;

5.6.8. Process Flow 处理流程



[注意]:

1、黄色部分由其他函数完成;

Init header：初始化头结点，该功能由函数 Mu_FDDFHeadInit 完成;

Create Node：创建 Fddf_Node 节点，初始化结点内容。该功能由函数 Mu_FDDFNodeCreate 完成;

Add to List：将解析完成的节点加入到链表中去，该操作按解析的先后顺序完成，具体操作由函数 Mu_FDDFNodeInsert 完成;

Free Node：对于解析中出现错误的节点，应该释放已经存储的内容，并且释放内存空间，具体的操作由函数 Mu_FDDFNodeDelete 完成;

5.6.9. Pseudocode 伪代码

调用者必须用 Mu_FDDFHeadCreate 创建结构

```
int Mu_ParseMultiDelFDDF(xmlXPathObjectPtr Ptr,    Fddf_Head_Ptr back)
{
    int length;
    int i = 0;
    int mark = 0;
    int backvalue = NO_MUERROR;
    //backvalue can use global value;

    xmlNodePtr curNode = NULL;
    xmlChar *muvalue = NULL;
    FDDF_NODE *node = NULL;

    xmlNodeSetPtr munodeset = Ptr->nodesetval;
    if(munodesetptr == NULL){
        Mu_ErrorPrint()
        return backvaule;
    }

    Mu_FDDFHeadInit(back);

    for(i;    i < munodeset->nodeNr;    i++){
        curNode = munodeset->nodeTab[i];
        if(curNode == NULL){
            Mu_ErrorPrint()
            break;
        }

        //jump to child segment
        curNode = curNode->child;
        if(curNode == NULL){
            Mu_ErrorPrint()
            break;
        }
        Mu_FDDFNodeCreate (&node);

        while(curNode != NULL){
            muvalue = xmlNodeGetContent(curNode);
            if(muvalue == NULL){
                Mu_ErrorPrint();
                mark = 1;
```

```
                break;
            }

            if((length = strelne(muvalue)) > MUXML_LENGTH_MAX){
                Mu_ErrorPrint();
                Mark = 1;
                Break;
            }

            Mu_FDDFNodeStrncpy (Node, curNode->name, muvalue);
            Mu_xmlFree(muvalue);
        }

        if(mark)
            Mu_FDDFNodeDelete(node);
        else
            Mu_FDDFNodeInsert (back, Node);

        curNode = curNode->next;
    }
    retrun backvalue;
}
```

5.6.10. Interface 接口

本函数为主体解析函数提供解析接口，同时，在解析的时候也需要利用其他函数：

Mu_FDDFHeadCreate ();
Mu_FDDFHeadInit();
Mu_FDDFNodeCreate ();
Mu_FDDFNodeStrncpy();
Mu_FDDFNodeDelete();
Mu_FDDFNodeInsert ();

[注意]:

1、详细说明请参考《MuFTAD 数据结构详细设计》;

5.6.11. Malloc 存储分配

在本函数运行的过程中，**不需要**申请内存空间以存储链表表头(FddfHead)，该操作由调用者完成;

但是函数**需要**申请内存空间以存储链表节点（FddfNode）和已查找得到的信息；

对于以上信息，本函数不会自动释放该空间，需要调用者显示地调用 free 释放结点的相应的信息地址；

5.6.12. Restrict 限制

略

5.6.13. Test 测试

本函数的测试的前提条件：待解析 XML 文件中存在相同的字段项，并且该字段项还应该有子项存在。

[注意]:

1、测试时可以参考《doc00_SoftProtocol_0.1.1》中 Query_FDDF 后返回的 FDDF XML 文件中的 segment 字段。

对本函数的测试，必须借助于主函数，并且使用正确的调用参数；

对于解析的结果均存储于链表中，因此为了直观地了解解析结果，应该写一个小的测试程序，输出链表中的内容；

需要测试程序对内存的影响；

5.6.14. Unsolve 未解决问题

略

5.7. Mu_ParseXpathXML

5.7.1. Name 函数名称

```
int Mu_ParseXpathXML(int type,
                     xmlDocPtr Doc,
                     const char *xpath,
                     void ** back);
```

5.7.2. Description 函数描述

该函数由解析的主解析函数调用，并且传递参数给本函数，函数处理完成后，返回处理的状态值。

函数中的各参数参考本章 5.6.5；

函数的返回值是 int 型，以标识函数的运行状态；

5.7.3. Function 功能

该函数对调用接口提供了总的、唯一的 XML 解析接口，用于解析重复、可变字段。在解析 XML 时，从接口函数处获得相关参数；

该函数不对所需要的信息进行存储和组织，只是在 XML 文件指针所指内容（doc）中查找适合规则（xpath）的字段，然后通过类型（type）决定调用具体的结构体操作函数对所解析的信息进行组织；

解析得到的信息通过无类型二级指针 back 返回，back 的具体类型由调用者决定；

[注意]:
1、本文档所说明的重复、可变字段是指：XML 文档中相同字段个数不确定，依具体情况而定；

5.7.4. Capability 性能

略

5.7.5. Input 输入

doc：XML 文件的文档指针；

type：类型字段，该值在全局类型（4.1.节）中描述，在解析时，必须明确指明所需要解析的类型；

xpath：为了使用 xpath 对 XML 文件中的字段进行定位，必须指明待解析字段的位置；

[注意]:
1、该字串的定义可以参考《xpath 手册》;

back：二级指针，可用于返回解析的信息，根据不同的类型，信息的组织方式也不一样，但均提供该二级指针；

5.7.6. Output 输出

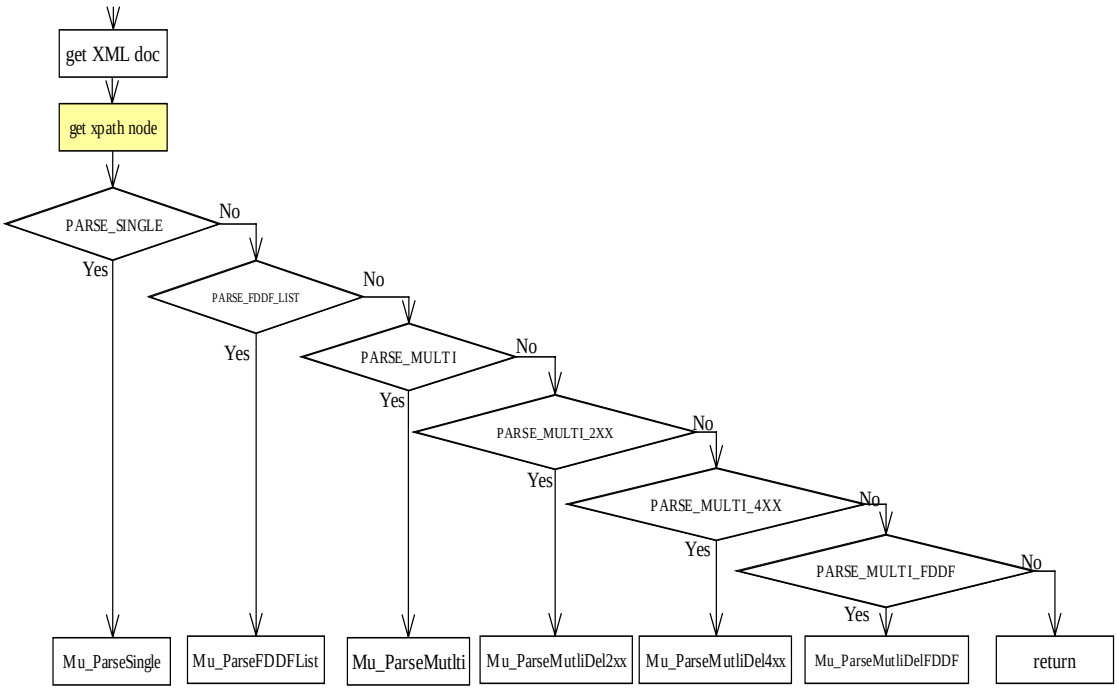
同 5.1.6. 中说明

5.7.7. Arithmetic 算法

略

5.7.8. Process Flow 处理流程

[注意]:
1、黄色部分由其他函数完成;



Get xpathnode：取得相应的字段点，该部分由函数 Mu_GetNode 完成;

5.7.9. Pseudocode 伪代码

```
int Mu_ParseXpathXML(int type,    xmlDocPtr Doc,    char *xpath, void ** back);
{
    xmlXpathObjectPtr xpath_result = NULL;
    int backvalue = NO_MUERROR;

    xmlChar *muXpath = (xmlChar *)xpath;
    Mu_GetXMLNode(doc, muXpath, &xpath_result);
    if(xpath_result == NULL){
        Mu_ErrorPrint();
        return MUEEOR;
    }

    switch(type){
    case PARSE_SINGLE:
        backvalue = Mu_parseSingle(xpath_result, back);
        break;
    case PARSE_FDDF_LIST:
        backvalue = Mu_ParseFDDFList(xpath_result, *back);
        break;
    case PARSE_MULTI:
        backvalue = Mu_ParseMulti(xpath_result, *back);
        break;
    case PARSE_MULTI_2XX:
        backvalue = Mu_ParseMulti2xx(xpath_result, *back);
        break;
    case PARSE_MULTI_4XX:
        backvalue = Mu_ParseMulti2xx(xpath_result, *back);
        break;
    case PARSE_MULTI_FDDF:
        backvalue = Mu_ParseMultiFDDF(xpath_result, *back);
        break;
    default:
        fprintf(stdout, "Error format!\n");
        backvalue = MUEEOR;
        break;
    }
    xmlXPathFreeContext(context);
    xmlXPathFreeObject(xpath_result);
    return backvalue;
```

}

5.7.10. Interface 接口

本函数是在解析包含重复、可变字段 XML 文件时，调用的函数接口；

同时，本函数还利用其他函数来完成相应的功能：
Mu_GetXMLNode()：该函数从 XML 的上下文中，获得与所求字段相符的节点信息；
Mu_ParseXXX()：该系列函数为具体的存储函数，如前面章节所述；

5.7.11. Malloc 存储分配

无

5.7.12. Restrict 限制

略

5.7.13. Test 测试

在测试上面章节函数时，对本节函数的功能已经测试完毕。

[注意]:
1、在测试需要删除字段的函数时，应该检查测试 XML 在程序运行后的结构;

5.7.14. Unsolve 未解决问题

略

5.8. Mu_ParseXML

5.8.1. Name 函数名称

```
int Mu_ParseXML(int type, xmlDocPtr Doc, void ** back);
```

5.8.2. Description 函数描述

该函数由解析的主解析函数调用，并且传递参数给本函数，函数处理完成后，返回处理的状态值。

函数中的各参数参考本章 5.7.5；

函数的返回值是 int 型，以标识函数的运行状态；

5.8.3. Function 功能

该函数对主函数提供了总的、唯一的 XML 解析接口，用于解析 XML 文件中不重复、不可变字段。在解析 XML 时，从主函数处获得相关参数；

该函数不对所需要的信息进行存储和组织，只是按类型（type）决定调用具体的解析函数对待解析的信息进行解析和存储；

解析得到的信息通过无类型二级指针 back 返回，back 的具体类型由调用者决定；

5.8.4. Capability 性能

略

5.8.5. Input 输入

type：类型字段，该值在全局类型（4.1.节）中描述，在解析时，必须明确指明所需要解析的类型；

doc：XML 文件的文档指针；

back：二级指针，用于返回解析的信息，根据不同的类型，信息的组织方式也不一样；

5.8.6. Output 输出

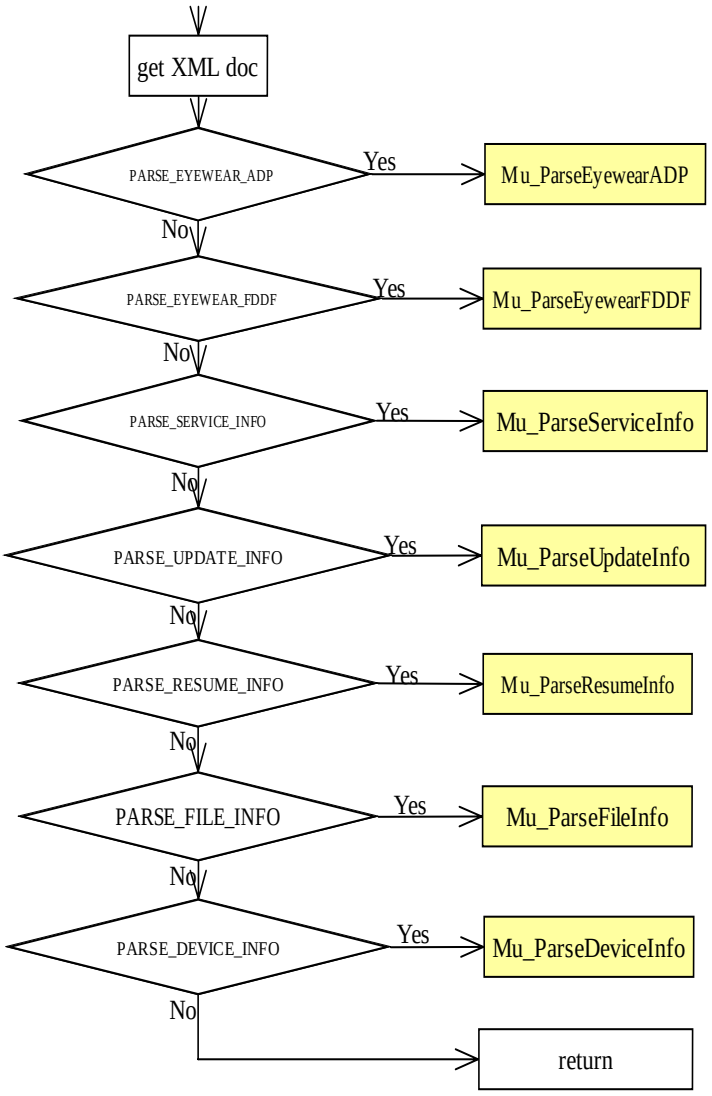
同 5.1.6. 中说明

5.8.7. Arithmetic 算法

略

5.8.8. Process Flow 处理流程

[注意]:
1、黄色部分由其他函数完成;



[注意]:
1、黄色部分由其他函数完成;

5.8.9. Pseudocode 伪代码

```
int Mu_ParseXML(int type,    xmlDocPtr doc,    void ** back);
{
    int bcakvalue = NO_MUERROR;

    if(NULL == doc){
            Mu_ErrorPrint()
            backvalue = …;
            return backvalue;
    }

    switch(type){
    case PARSE_EYEWEAR_ADP:
        backvalue = Mu_ParseEyewearADP(doc, *back);
        break;
    case PARSE_EYEWEAR_FDDF:
        backvalue = Mu_ParseEyewearFDDF(doc, *back);
        break;
    case PARSE_SERVICE_INFO:
        backvalue = Mu_ParseServiceInfo(doc, *back);
        break;
    case PARSE_UPDATE_INFO:
        backvalue = Mu_ParseUpdateInfo(doc, *back);
        break;
    case PARSE_RESUME_INFO:
        backvalue = Mu_ParseResumeInfo(doc, *back);
        break;
    case PARSE_FILE_INFO:
        backvalue = Mu_ParseFileInfo(doc, *back);
        break;
    case PARSE_DEVICE_INFO:
        backvalue = Mu_ParseDeviceInfo（doc, *back）;
        break;
    default:
        fprintf(stdout, "Error format!\n");
        backvalue = MUEEOR;
        break;
    }

    return backvalue;
```

}

5.8.10. Interface 接口

本函数是在解析包含重复、不可变字段 XML 文件时，主函数调用的接口；

同时，本函数还利用其他函数来完成相应的功能：
Mu_ParseXXX()：该系列函数为具体的解析、存储函数，如后面章节所述；

5.8.11. Malloc 存储分配

无

5.8.12. Restrict 限制

略

5.8.13. Test 测试

在测试上面章节函数时，对本节函数的功能已经测试完毕。

5.8.14. Unsolve 未解决问题

略

5.9. Mu_Parse

5.9.1. Name 函数名称

int Mu_Parse (int type, const int fd, xmlChar *xpath, void ** back);

5.9.2. Description 函数描述

用户需要解析 XML 文件中的内容时，需要调用本函数，并且按具体需求填充函数参数；

函数中的各参数参考本章 5.7.5；

函数的返回值是 int 型，以标识函数的运行状态；

5.9.3. Function 功能

该函数对外提供了总的、唯一的 XML 解析接口，用户在解析 XML 时，只需要在该函数参数中填充正确的参数值即可；

该函数不对所需要的信息进行存储和组织，只是在打开的文件（fd）中查找适合规则（xpath）的字段，然后通过类型（type）决定调用具体的存储函数对所解析的信息进行组织；

解析得到的信息通过无类型二级指针 back 返回，back 的具体类型由调用者决定；

5.9.4. Capability 性能

略

5.9.5. Input 输入

fd：打开的文件句柄，该句柄指向存储有待解析 XML 文件；

type：类型字段，该值在全局类型（4.1.节）中描述，在解析时，必须明确指明所需要解析的类型；

xpath：为了使用 xpath 对 XML 文件中的字段进行定位，必须指明待解析字段的位置；

[注意]:

1、该字串的定义可以参考《xpath 手册》;

back：二级指针，用于返回解析的信息，根据不同的类型，信息的组织方式也不一样；

5.9.6. Output 输出

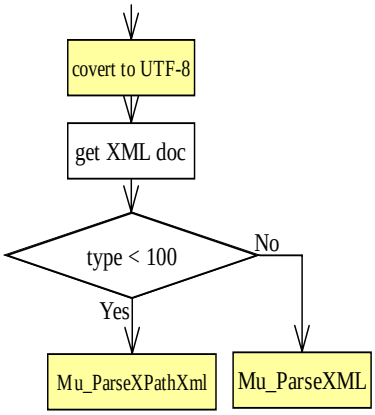
同 5.1.6. 中说明

5.9.7. Arithmetic 算法

略

5.9.8. Process Flow 处理流程

[注意]:
1、黄色部分由其他函数完成;



[注意]:
2、黄色部分由其他函数完成;

Covert to UTF-8：对待解析文档进行格式转码，在本程序中，该函数被留空 Mu_Covert;

5.9.9. Pseudocode 伪代码

```
int Mu_Parse (int type,    const int fd ,    xmlChar *xpath,    void ** back);
{
    int bcakvalue = MUOK;
    xmlDocPtr doc = NULL;

    Mu_Convert(fd);//reserved
```

```
    doc = xmlReadFd(fd, NULL, "UTF-8", XML_PARSE_RECOVER);
    //check libxml manual to got the parameters!
    if(doc == NULL){
        Mu_ErrorPrint()!
        return ERROR;
        }

    if(CHECK_XPATH(type))
        backvalue = Mu_ParseXML(type, doc, back);
    else
        backvalue = Mu_ParseXpathXML(type, doc, xpath, back);

    xmlFreeDoc(doc);

    return backvalue;
}
```

5.9.10. Interface 接口

本函数是在用户解析 XML 文件时调用的主函数接口；

同时，本函数还利用其他函数来完成相应的功能：
Mu_Covert()：该函数主要将非 UTF-8 码的文件转换成 UTF-8 码，由于本项目所使用 XML 均为 UTF-8，所以对其实现功能暂预留为空；
Mu_ParseXXXXML()：该系列函数依 XML 文件是否重复、可变分类，其为具体的需要调用函数解析，参见前面章节所述；

5.9.11. Malloc 存储分配

无

5.9.12. Restrict 限制

略

5.9.13. Test 测试

在测试上面章节函数时，对本节函数的功能已经测试完毕。

[注意]:
1、在测试需要删除字段的函数时，应该检查测试 XML 在程序运行后的结构;

5.9.14. Unsolve 未解决问题

如果 XML 文件不是 UTF-8 格式编码的，那么在解析时，必须首先进行编码转换，由于在本项目中，XML 均定义为 UTF-8 的，故转码函数留空。

5.10. Mu_ParseEyewearADP

5.10.1. Name 函数名称

int Mu_ParseEyewearADP(xmlDocPtr Doc, EyewearADPPtr back);

5.10.2. Description 函数描述

函数接收从 Mu_ParseXML 传递过来的参数，解析 CMD XML 文件。该 XML 文件用于 Device 与 Server 间传递信息;

函数接收的各参数如 5.9.5.中描述;

函数的返回值是 int 型，以标识函数的运行状态;

5.10.3. Function 功能

该函数在运行时被 Mu_ParseXML 调用，从该函数处获得相关的参数值;

函数从 XML 文档指针所指向的内容中，依次序解析出 CMD XML 中的不可变字段信息。

解析得到的信息添加到 back 为头结点的链表中;

5.10.4. Capability 性能

略

5.10.5. Input 输入

doc：待解析 XML 文件的文档指针；

back：一级指针，指明存储的头结点；

5.10.6. Output 输出

同 5.1.6.中说明

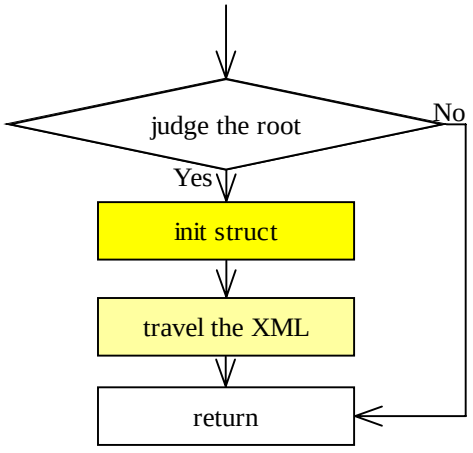
5.10.7. Arithmetic 算法

略

5.10.8. Process Flow 处理流程

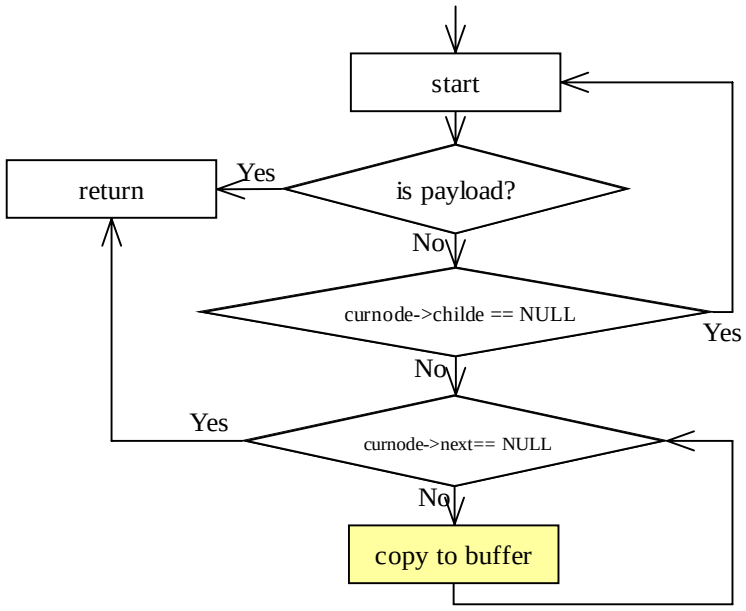
函数的的处理分为二个部分，一部分是运行主体，另一部分则是该类型文件的遍历函数；

针对 XML 文件的解析，使用了递归的方式，函数的流程图如下：



[注意]:
1、图中黄色部分由其他函数完成;

对于遍历函数，结构如下：



[注意]:

1、黄色部分由其他函数完成;

5.10.9. Pseudocode 伪代码

[注意]:

1、调用该函数前，调用者应该已经初始化 EyewearADP 结构;

```
int Mu_EyewearADP(xmlDocPtr doc, EyewearADPPtr back)
{
    xmlNodePtr curNode = NULL;
    xmlChar *muvalue = NULL;

    ;
    if(NULL == (curNode = xmlDocGetRootElement(doc))
        ||!xmlStrcmp(curNode->name, (xmlChar *)"Eyewear_ADP")){
        return MUEXADP;
    }

    //init
    Mu_EyewearADPClear(back);

    curNode = curNode->childe;
    Mu_EyewearADPTravel(doc,curNode, back);
```

```
    return backvalue;
}

/*parse the data and copy except payload segment*/
void Mu_EyewearADPTravel(xmlDocPtr doc, xmlNodePtr Node, EyewearADPPtr
back)
{
    while(Node)
    {
        if((xmlStrcmp(Node->name,(xmlChar *)"payload")))
        {
            Mu_EyewearADPTravel2(doc,Node,back);
        }
        Node = Node->next;
    }

    return;
}

int Mu_EyewearADPTravel2(xmlDocPtr doc,xmlNodePtr Node,EyewearADPPtr
back)
{
    xmlChar *muvalue = NULL;
    xmlNodePtr tempNode = NULL;

    while(Node)
    {
        if(Node->children)
            Mu_EyewearADPTravel2(doc,Node->children, back);
        if(NULL == ( muvalue = xmlNodeListGetString(doc, Node->children,
1)))
        {
            Node = Node->next;
            continue;
        }
        else
        {
            Mu_EyewearADPStrncpy(back ,muvalue, Node->name);
        }
        Mu_xmlFree(muvalue);
```

```
                tempNode = Node->next;
                xmlUnlinkNode(Node);
                xmlFreeNode(Node);
                Node = tempNode;
    // Node = Node->next;
    }
    return MUOK;
}
```

5.10.10. Interface 接口

本函数在解析时,利用递归函数在 XML 文档内移动,并且利用相应函数,解析字段信息;

本函数用到的函数有:

```
Mu_EyeWearADPInit();
Mu_EyewearADPTravel();
Mu_EyewearADPStrncpy();
```

5.10.11. Malloc 存储分配

在函数运行过程中，需要分配一段内存区域，用于存储已解析的信息。分配的长度由所解析的字串长度本身决定;

[注意]:

1、EyewearADP 结构，由调用者创建并初始化;

对于超过 MUXML_LENGTH_MAX 的字段，解析时，直接返回错。

5.10.12. Restrict 限制

略

5.10.13. Test 测试

对本函数的测试，必须借助于主函数，并且使用正确的调用参数;

测试时，需要存在完整的 XML 文件;

对于解析的结果存储于内存中，因此为了直观地了解解析结果，应该写一个

小的测试程序，输出内存中的内容；

需要测试程序对内存的影响；

5.10.14. Unsolve 未解决问题

略

5.11. Mu_ParseServiceInfo

5.11.1. Name 函数名称

int Mu_ParseServiceInfo(xmlDocPtr Doc, ServiceInfoPtr back);

5.11.2. Description 函数描述

函数接收从 Mu_ParseXML 传递过来的参数，解析 Server Information Description XML 文件。该 XML 文件用于在 Device 端存储服务器的相关信息；

函数接收的各参数如 5.9.5.中描述；

函数的返回值是 int 型，以标识函数的运行状态；

5.11.3. Function 功能

该函数在运行时被 Mu_ParseXML 调用，从该函数处获得相关的参数值；

函数从 XML 文档指针所指向的内容中，依次序解析出 Server Information Description XML 中的不可变字段信息。

解析得到的信息存储到 back 所指的结构中；

5.11.4. Capability 性能

略

5.11.5. Input 输入

同 5.9.5.中说明

5.11.6. Output 输出

同 5.1.6.中说明

5.11.7. Arithmetic 算法

ServiceInfo 结构由二部分组成：

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

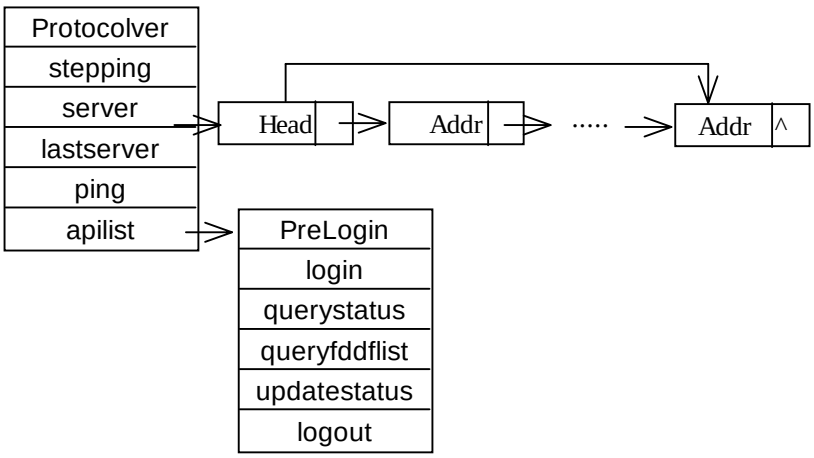
该结构体是一个服务器地址的链表表头，在协议中，服务器的地址个数是不确定的；

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、上述二个结构均由调用者创建并初始化;

2、具体可以参考《MuFTAD 数据结构详细设计》;



5.11.8. Process Flow 处理流程

参考 5.10.8.中说明

5.11.9. Pseudocode 伪代码

[注意]:
1、调用此函数前，调用者应该已创建并初始化 ServiceInfo 结构;

```
int Mu_ParseServiceInfo(xmlDocPtr doc, ServiceInfoPtr back)
{
    xmlNodePtr curNode = NULL;
    int backvalue = NO_MUERROR;
    xmlChar *muvalue = NULL;

    if(NULL == (curNode = xmlDocGetRootElement(doc))
        ||! xmlStrcmp(curNode->name, (xmlChar *)"Service_Info")){
        return MUEXSVI;
    }

    curNode = curNode->child;

    Mu_ServiceInfoClear(back);

    Mu_ServiceInfoTravel(doc,curNode, back);

    return backvalue;
}

/*parse serviceinfo.xml*/
void Mu_ServiceInfoTravel(xmlDocPtr doc,xmlNodePtr Node, ServiceInfoPtr
back)
{
    xmlChar *muvalue = NULL;
    ServerNodePtr servernode = NULL;
    while(Node)
    {
        if(Node->children)
            Mu_ServiceInfoTravel(doc,Node->children, back);
        if(NULL == ( muvalue = xmlNodeListGetString(doc, Node->children,
```

```
1)))
        {
            Node = Node->next;
            continue;
        }

        else if(xmlStrcmp(Node->name,(xmlChar*) "Addr"))
        {
            Mu_ServiceInfoStrncpy(back ,muvalue, Node->name);
        }
        else
        {
            Mu_ServerNodeCreate(&servernode);
            Mu_ServerNodeStrncpy(servernode, muvalue, Node->name);
            Mu_ServerNodeInsert(back->server, servernode);
        }
        Mu_xmlFree(muvalue);
        Node = Node->next;
    }

    return;
}
```

5.11.10. Interface 接口

本函数在解析时,利用递归函数在 XML 文档内移动,并且利用相应函数,解析字段信息;

本函数用到的函数有:

```
Mu_ServiceInfoInit();
Mu_ServiceInfoCtrncpy();
Mu_ServerNodeCreate();
Mu_ServerNodeStrncpy();
Mu_ServerNodeInsert();
```

5.11.11. Malloc 存储分配

同 5.9.11.中说明

5.11.12. Restrict 限制

略

5.11.13. Test 测试

同 5.9.13.中说明

5.11.14. Unsolve 未解决问题

略

5.12. Mu_ParseEyewearFDDF

参考 5.11. Mu_ParseSeviceInfo

需要提供二个类似的函数:

```
int Mu_ParseEyewearFDDF(xmlDocPtr Doc, FddfFilePtr back);
int Mu_EyewearFDDFTravle(xmlDocPtr Doc,xmlNodePtr Node,FddfFilePtr back);
```

[注意]:

1、对 Description 字段的解析应该考虑对字段的截断，而不是简单的放弃;

5.13. Mu_ParseUpdateInfo

参考 5.10. Mu_ParseEyeweareADP

需要提供二个类似的函数:

```
int Mu_ParseUpdateInfo(xmlDocPtr Doc, UpdateInfoPtr back);
int Mu_UpdateInfoTravel(xmlDocPtr Doc,xmlNodePtr Node,UpdateInfoPtr Ptr)
```

5.14. Mu_ParseDeviceInfo

参考 5.10. Mu_ParseEyeweareADP

需要提供二个类似的函数:

```
int Mu_ParseDeviceInfo (xmlDocPtr Doc, DeviceInfoPtr back);
```

int Mu_DeviceInfoTravel(xmlDocPtr Doc,xmlNodePtr Node,XmlDeviceInfoPtr Ptr);

5.15. Mu_ParseResumeInfo

参考 5.10. Mu_ParseEyeweareADP

需要提供二个类似的函数:

int Mu_ParseResumeInfo(xmlDocPtr Doc, ResumeInfoPtr back);
int Mu_ResumeInfoTravel(xmlDocPtr Doc,xmlNodePtr Node,ResumeInfoPtr Ptr);

5.16. Mu_ParseFileInfo

参考 5.10. Mu_ParseEyeweareADP

需要提供二个类似的函数:

int Mu_ParseFileInfo(xmlDocPtr Doc, FileInfoPtr back);
int Mu_FileInfoTravel(xmlDocPtr Doc,xmlNodePtr Node,FileInfoPtr Ptr);

5.17 Mu_ParsePayload

需要提供的两个类似的函数:

int Mu_ParsePayload(xmlDocPtr Doc,SRV_SvrInfPtr srv);
int Mu_PayloadTravel(xmlDocPtr Doc,xmlNodePtr Node,SRV_SvrInfPtr Ptr);

5. 18 Mu_ParseFDDFsegment

说明:

解析 XML 文件中节点中的信息。

需要提供的函数:

int Mu_ParseFDDFsegment(xmlXPathObjectPtr Ptr,SegmentHeadPtr back);

6. Appendix 附录

6.1. Assistant Routines 辅助函数

6.1.1. Mu_TruncateInfo

```
int Mu_TruncateInfo(xmlChar *value, int len)
{
	//according to MUXML_LENGTH_MAX,truncate the string
	//all the characters in string are UTF-8
	//value: The string you want to cut.
	//len: The display width of a string you want.
	int i = 0;
	int j = 0;
	char temp[61];
	char * XML_Buff = (char*)malloc(xmlStrlen(value) + 1);
	memset(XML_Buff, 0, (xmlStrlen(value) + 1));
	strncpy(XML_Buff, (char*)value, (xmlStrlen(value)));
	if(len > 60)
		len = 60;
	memset(temp, 0 ,61);
	do{
		//If the character is a Chinese
		if((unsigned char)XML_Buff[i] > 224)
		{
			strncat(temp, &XML_Buff[i], 1);
			strncat(temp, &XML_Buff[i+1], 1);
			strncat(temp, &XML_Buff[i+2], 1);
			i = i + 3;
			j = j + 2;
		}
		//If the character is a symble
		else if((unsigned char)XML_Buff[i] > 192)
		{
			strncat(temp, &XML_Buff[i], 1);
```

```
            strncat(temp, &XML_Buff[i+1], 1);
            i = i + 2;
            j = j + 2;
        }
        //If the character is a alpha
        else
        {

            strncat(temp, &XML_Buff[i], 1);
            i++;
            j++;
        }
    }while(j < len);

    return j;
}
```

6.2. XML files XML 文件

6.2.1. Server Information Description

00_ServerInfo.XML
<serverList><addr>字段在该 XML 文件中一般会包含多个;
用 Mu_ParseSericeInfo 函数进行解析该 XML 文件中的
<Protocol_Ver>
<Stepping>
<APIPathList>

用 Mu_ParseMulti 函数解析 XML 文件中的
<ServerList><Addr>

用 Mu_ParseSingle 函数解析 XML 文件中的
<ServerList><Ping>
<ServerList><Last>

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。

用 Mu_ParseEyewearADP 函数解析返回的信息；

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；

调用 Mu_ParseEyewearADP 函数解析返回的信息；

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；

调用 Mu_ParseEyewearADP 函数解析返回信息中的大部分信息；

但是对于返回信息中的<payload>字段，分别调用 Mu_ParseMulti2xx 和 Mu_ParseMutli4xx 解析；

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；

调用 Mu_ParseEyewearADP 函数解析返回信息中的大部分信息；

但是对于返回信息中的<payload>字段，调用 Mu_ParseSingleFDDF 解析；

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；

调用 Mu_ParseEyewearADP 函数解析返回信息中的大部分信息；

6.2.7. FDDF

FDDF_1.xml
该 XML 描述待下载节目的分段；

用 Mu_ParseEyewearFDDF 函数解析返回信息中的大部分内容；

对于其中的字段，用 Mu_ParseMutliFDDF 解析；

6.2.8. FileInfo

file_info.xml
该 XML 用于描述文件信息；

在检测文件的完整性时用到，用 Mu_ParseFileInfo 解析；

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；

用 Mu_ParseResuming 函数解析；

6.2.10. DeviceInfo

deviceinfo.xml
存储设备相关信息；

用 Mu_ParsedeviceInfo 函数解析；

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；

用 Mu_ParseUpdateInfo 函数解析；