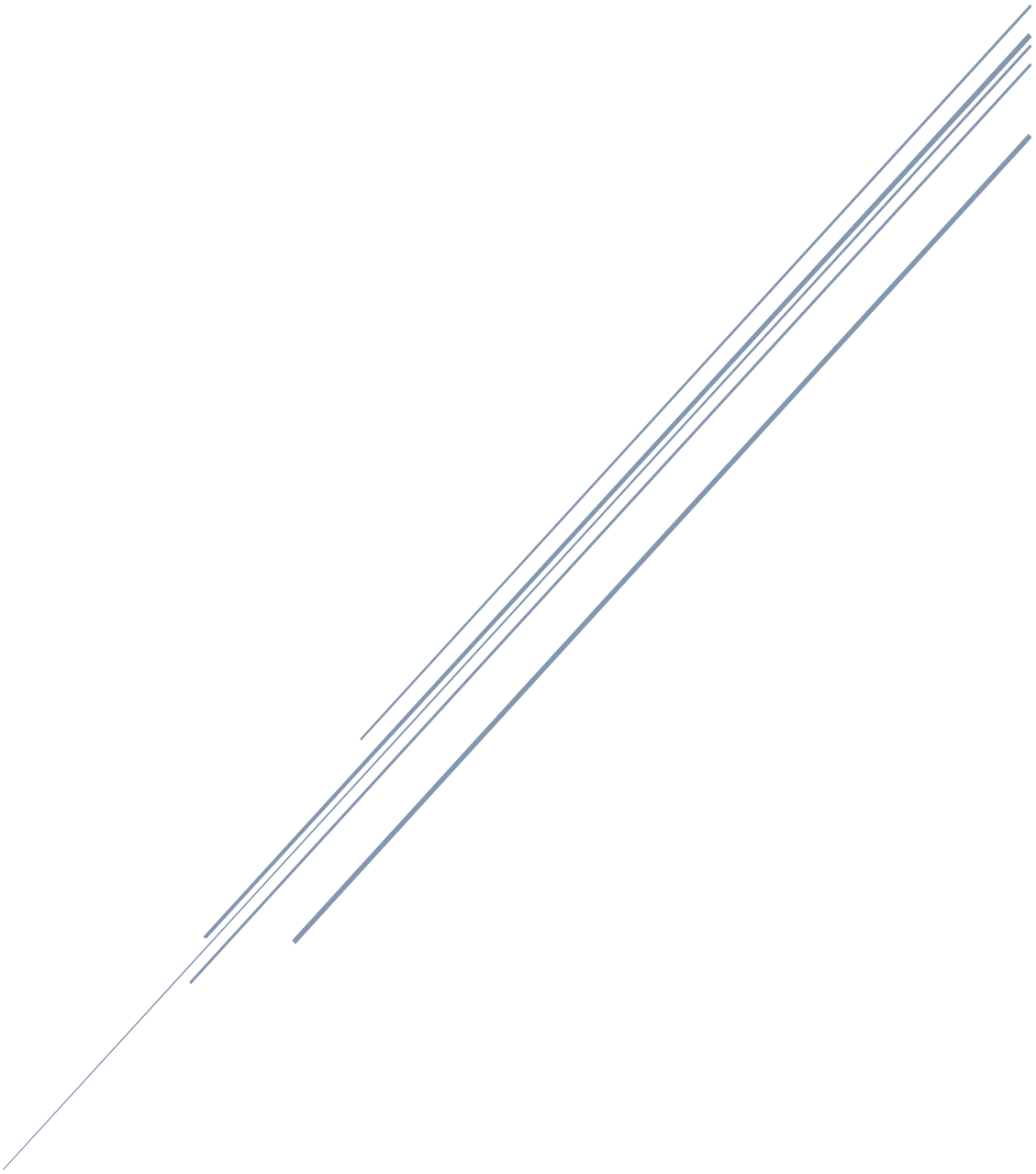
UCI API 文档



优利德科技（中国）股份有限公司

版本：

版本	修改项	日期
V1.0	正式版	2016 年 1 月
V1.1	提供查询、打开、读写数据和读写文件的参数未封装版本，所有简化版本后面都会有一个大写的 X 做区分。具体参看文档和 uci.h 文件的注释。	2016 年 10 月 24 日
V1.2	新增 *uci_SetAttribute* 和 *uci_SetNotify* 的接口说明。	2016 年 12 月 29 日
V1.2.1	增加新的错误码描述； 增加“如何获取设备访问地址？”和“如何检查设备是否在线？”描述； 其它细节修改	2018 年 8 月 6 日
V1.2.2	丰富 uci_QueryNodesX 的说明，对查询文本做进一步说明，并给出完整示例。	2018 年 8 月 10 日

S 简介：

UCI 接口实现多设备，多接口的统一通信。具体的某一款设备的通信协议，有单独的相匹配的文档提供，本文档只介绍 UCI 接口的使用。

在参看文档的同时，也可以参照 SDK 中的示例工程理解接口的使用。

UCI 接口 C 语言版本的声明请参照 uci.h。

请注意：

1、 本 SDK 提供 UNICODE 和 ASCII 两个字符编码的 UCI 库；请根据需要选择合适的版本。比如 LABVIEW 请使用 ASCII 版本。 VC 默认使用 UNICODE 版本；

2、 所有接口都有一个超时（Timeout）参数，该参数指的是：如果接口执行的操作在该设定的时间（ms）内没有完毕，则中断返回。

3、 所有接口的返回值都是一个 32 位有符号整数，<0 时代表错误，其值对应错误码，可以参照附录:UCI 错误码，或使用 uci_GetLastError 接口查询错误描述文本；≥0 代表成功，而很多接口，再>0 时表示更多的意义，比如返回的会话 ID 和数据量等等，具体请参照每个接口的详细说明。

相关引用文件：

1、ucidef.h

UCI 库的相关基础定义。

2、uci.h

UCI 库的接口定义文件。

3、uci.lib

UCI 动态库的链接文件。

4、uci.dll

UCI 动态库文件。

接口说明：

1、uci_QueryNodes

查询指定的通信节点，得到可被连接的所有设备。 可以使用简版的接口 uci_QueryNodesX.

Syntax:

```
u_status _UCIAPI uci_QueryNodes(_in const QParams* _params,
                                _out Node* _outBuf,
                                _in_out u_size* _nodeCnt,
                                _in u_size _timeOut);
```

Parameters ：

const QParams* _params	指定查询参数
Node* _outBuf	存放查询到的节点数据
u_size* _nodeCnt	入参为要查获取的节点数量，出参为实际读取到的节点数量。
u_size _timeOut	查询超时，单位： ms.

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

要查询多少个节点由调用者决定，通过传入相应的_outBuf 和_nodeCnt 即可！

eg:

```
Node nodes[10];
ZeroMemory(nodes, sizeof(nodes));
u_size notesCNT = sizeof(nodes) / sizeof(nodes[0]);
int r = uci_QueryNodes(&qp, nodes, &notesCNT, 2000);
if (r < 0) {
    TRACE(_T("\r\n Error to query nodes, err code = 0x%x"), r);
    return;
}
if (notesCNT == 0){
    AfxMessageBox(_T("没有查找到任何符合条件的设备！"));
    return ;
}
```

其中 qp 的创建见：QParams

2、uci_QueryNodesX

查询指定的通信节点，得到可被连接的所有设备。 是 uci_QueryNodes 的简化版本。

Syntax:

```
u_status _UCIAPI uci_QueryNodesX(u_cstring _msg, Node* _nodes,
                                 u_size _node_count, u_size _timeout);
```

Parameters ：

u_cstring ***_msg***	查询字符串，格式如： "LAN:4162,5000;USB:0x1234&0x5345,0x7777&0x5345;" **注意**：每种接口名称后面紧跟“:”，必须以“;”结尾。 USB 字段的意义是 PID&VID，不是 VID&PID，比如 0x7777&0x5345，代表 PID = 0x7777，VID =0x5345.
*Node** ***_nodes***	存放查询到的节点数据，具体见 Node.
u_size ***_node_count***	要查询的节点个数。
u_size ***_ timeout***	查询超时，单位： ms.

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

要查询多少个节点由调用者决定，通过传入相应的***_nodes*** 和***_node_count*** 即可！

要查询什么设备，就要添加对应的设备信息。 网络设备要添加查询端口，USB 设备要添加 PID&VID，否则无法查询到。

目前比较完整的查询文本如下：

"USB:0x1234&0x5345,0x7777&0x5345,0x0834&0x5656,0x5537&0x4348,0xE008&0x1A86,0xEA80&0x10C4;"

Example:

```
Node nodes[10];
ZeroMemory(nodes, sizeof(nodes));
u_size notesCNT = sizeof(nodes) / sizeof(nodes[0]);
int r = uci_QueryNodesX(_T("LAN:4162,5000;USB:0x1234&0x5345,0x7777&0x5345;"), nodes, notesCNT, 2000);
if (r < 0) {
    TRACE(_T("\r\n Error to query nodes, err code = 0x%x"), r);
    return;
}
if (notesCNT == 0){
    AfxMessageBox(_T("没有查找到任何符合条件的设备！"));
    return ;
}
```

详细示例请参看 SDK 中的 UCIDemoProj 工程源码。

3、uci_Open

与设备建立连接。也可以使用简化版本 uci_OpenX.

Syntax:

```
u_status _UCIAPI uci_Open(_in u_cstring _addr,
                          u_session* _session,
                          u_uint32 _timeOut = 6000);
```

Parameters ：

u_cstring _addr	设备连接字符串， 以'\0'结尾。
u_session* _session	返回已建立的会话 ID.在后续接口中会用到。
u_size _timeOut	连接超时时间，单位： ms.

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

确定 _addr 的最简单的方式是通过机型对应的文档中获取，或者通过调用 uci_QueryNodes 查询已经在网的所有设备，然后通过 Node.UCIAddr 得到连接字符串地址。

4、uci_OpenX

与设备建立连接。是 uci_Open 的简化版本。

Syntax:

```
u_status _UCIAPI uci_OpenX(_in u_cstring _addr, u_uint32 _timeOut);
```

Parameters ：

u_cstring _addr	设备连接字符串， 以'\0'结尾。
u_size _timeOut	连接超时时间，单位： ms.

Return Value:

<0 表示错误，错误码详见：UCI 错误码;
>0 表示会话 ID;

Remarks ：

确定 _addr 的最简单的方式是通过机型对应的文档中获取，或者通过调用 uci_QueryNodes 查询已经在网的所有设备，然后通过 Node.UCIAddr 得到连接字符串地址。

example:

```
static u_session g_session = INVALID_SESSION;

//....

u_status r =
       uci_OpenX("[C:DSO][D:UPO2000CS][T:USB][PID:0x1234][VID:0x5345][EI:0x81][EO:0x3][CFG:3]", 2000);
if (UCISUCCESS(r))
    g_session = (u_session)r;
```

5、uci_Close

关闭会话

Syntax:

*u_status _UCIAPI uci_Close(u_session **_session**)*

Parameters ： 无

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

uci_Close 和 uci_Open 应该成对出现，即建立连接后，再退出时应该关闭连接！

6、uci_Write

写数据。 也可以使用简化版本 uci_WriteX.

Syntax:

```
u_status _UCIAPI uci_Write(u_session _session,
                           PWParams _params,
                           const u_byte* _data = nullptr,
                           u_size _len = 0);
```

Parameters ：

*u_session **_session***	会话 ID，通过 uci_Open 接口得到。
*PWParams **_params***	写操作参数，见 *WParams*.
const u_byte **_data***	要写入的数据缓冲区地址， 可以为 NULL
*u_size **_len***	要写入的数据缓冲区的长度， 如果_buf == NULL,则_len 会被忽略可以为任何值

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

对应常见内置数据类型，可以将数据格式化到命令字符串，而不是使用*_data* 和*_len* 的组合写数据。可以封装为下面的接口：

```
int UCIWrite(const TCHAR* _cmd, UINT _timeOut) {
    WParams wp;
    ZeroMemory(&wp, sizeof(wp));
    return uci_Write(g_curSession, uci_CreateWParams(wp, _cmd, _timeOut));
}
```

其中 **g_curSession** 是由 uci_Open 得到。

如果数据已经无法格式化到命令字符串，那么可以通过*_data* 和*_len* 写数据。具体使用，见具体设备通信协议的具体写命令。

7、uci_WriteX

写数据。 是 uci_Write 的简化版本。

Syntax:

```
u_status _UCIAPI uci_WriteX(u_session _session,
                            u_cstring _msg,
                            u_uint32 _timeout,
                            const u_byte* _data,
                            u_size _len);
```

Parameters ：

u_session ***_session***	会话 ID，通过 uci_Open 接口得到。
u_cstring ***_msg***	写指令字符串。
u_uint32 ***_timeout***	写操作超时
*const u_byte** ***_data***	要写入的数据缓冲区地址， 可以为 NULL
u_size ***_len***	要写入的数据缓冲区的长度， 如果_buf == NULL,则_len 会被忽略可以为任何值

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

对应常见内置数据类型，可以将数据格式化到命令字符串，而不是使用*_data* 和*_len* 的组合写数据。
如果数据已经无法格式化到命令字符串，那么可以通过*_data* 和*_len* 写数据。具体使用，见具体设备通信协议的具体写命令。

8、uci_WriteSimple

写数据。 是 uci_Write 的简化版本。省去了二进制写数据接口。

Syntax:

```
u_status _UCIAPI uci_WriteSimple(u_session _session, u_cstring _msg, u_uint32 _timeout);
```

Parameters ：

u_session _session	会话 ID，通过 uci_Open 接口得到。
u_cstring _msg	写指令字符串。
u_uint32 _timeout	写操作超时

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

对应常见内置数据类型，可以将数据格式化到命令字符串，而不是使用_data 和_len 的组合写数据。
如果数据已经无法格式化到命令字符串，那么可以通过_data 和_len 写数据。具体使用，见具体设备通信协议的具体写命令。

9、uci_FormatWrite

以格式化字符串的方式写数据。

Syntax:

```
u_status _UCIAPI uci_FormatWrite( u_session _sesn,
                                  u_uint32 _timeOut,
                                  const u_char *format, ...);
```

Parameters ：

u_session _sesn	会话 ID，通过 uci_Open 接口得到。
u_size _timeOut	写超时，单位：ms.
const u_char *format, ...	格式化字符串。

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

本接口是 uci_Write 接口的再封装，目的是方便只以格式化字符串写数据的操作。

示例：

```
void SendKey(int _key) {
#if 0
    CString s;
    s.Format(_T("KEY:%d;"), _key);
    UCIWrite(s, 1000);
#else
    uci_FormatWrite(g_curSession, 1000, _T("KEY:%d;"), _key);
#endif
}
```

其中 UCIWrite 见 uci Write 说明

10、uci_Read

读数据。 也可以使用简化版：uci ReadX

Syntax:

```
u_status _UCIAPI uci_Read(u_session _session,
                          PRParams _params,
                          u_byte* _data,
                          u_size _dataLen);
```

Parameters ：

u_session ***_session***	会话 ID，通过 uci Open 接口得到。
PRParams ***_params***	读操作的命令参数，见 RParams.
*u_byte** ***_data***	接收数据的缓冲区。
u_size ***_dataLen***	接收数据的缓冲区大小，大小由协议决定。

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

11、uci_ReadX

读数据。是 uci_Read 的简化版。

Syntax:

```
u_status _UCIAPI uci_ReadX(u_session _session, u_cstring _msg, u_uint32 _timeout,
                           u_byte* _data, u_size _dataLen);
```

Parameters ：

u_session _session	会话 ID，通过 uci_Open 接口得到。
u_cstring _msg	读指令字符串。
u_uint32 _timeout	读指令超时。
u_byte* _data	接收数据的缓冲区。
u_size _dataLen	接收数据的缓冲区大小，大小由协议决定。

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

一次只能执行一个读指令。

12、uci_WriteFromFile

写文件。 也可以使用简化版本：uci_WriteFromFileX.

Syntax:

```
u_status _UCIAPI uci_WriteFromFile(u_session _session, WFileParams* _info);
```

Parameters ：

u_session _session	会话 ID，通过 uci_Open 接口得到。
WFileParams* _info	写文件相关参数（WFileParams）

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

13、uci_WriteFromFileX

写文件。是 uci_WriteFromFileX 的未封装版本。

Syntax:

```
u_status _UCIAPI uci_WriteFromFileX(u_session _session,
                                    u_cstring _msg,
                                    u_cstring _filePath,
                                    u_uint32 _timeout);
```

Parameters ：

u_session _session	会话 ID，通过 uci_Open 接口得到。
u_cstring _msg	命令字符串，以'\0'结尾. 具体格式见文档;
u_cstring _filePath	要写入的文件路径（包括文件名）
u_uint32 _timeout	写超时(单位：ms)

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

14、uci_ReadToFile

读数据到文件。 也可以使用参数未封装版本：uci_ReadToFileX.

Syntax:

```
u_status _UCIAPI uci_ReadToFile(u_session _session, RFileParams* _params);
```

Parameters ：

u_session _session	会话 ID，通过 uci_Open 接口得到。
RFileParams* _params	读文件数据的参数。见：RFileParams

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

15、uci_ReadToFileX

读数据到文件。 是 uci_ReadToFile 的未封装版本.

Syntax:

```
u_status _UCIAPI uci_ReadToFileX(u_session _session, u_cstring _msg, u_cstring _filePath,
                    u_uint32 _timeout, u_tchar *_filePathFinal, u_int32 _filePathFinalLength);
```

Parameters ：

u_session _session	会话 ID，通过 uci_Open 接口得到。
u_cstring _msg	命令字符串，以'\0'结尾. 具体格式见文档;
u_cstring _filePath	缓冲数据到本地磁盘的文件路径。(包括文件名和后缀）. 可以是绝对路径也可以是相对路径。如果是相对路径可以通过_filePathFinal 获取绝对路径。
u_uint32 _timeout	超时(单位：ms)
u_tchar *_filePathFinal	输出缓冲到本地的文件的绝对路径
u_int32 _filePathFinalLength	filePathFinal 的长度，按字符计数

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

16、uci_SendCommand

发送控制命令

Syntax:

```
u_status _UCIAPI uci_SendCommand(u_session _session, PCommandParams _params);
```

Parameters ：

u_session _session	会话 ID，通过 uci_Open 接口得到。
PCommandParams _params	命令参数，见 CommandParams.

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

17、uci_GetLastError

得到最后一次操作的错误信息

Syntax:

```
u_cstring _UCIAPI uci_GetLastError();
```

Parameters ：

Return Value:

返回错误信息的字符串

Remarks ：

可以使用 uci_SetAttribute 接口设置整个 uci 库的语言环境，其中返回的错误信息的文本的语言也可同步切换。

19、 uci_SetAttribute

设置属性

Syntax:

```
u_status _UCIAPI uci_SetAttribute(u_session _session, u_cstring _msg, const u_object* _obj, u_size _objSize);
```

Parameters ：

u_session _session	会话 ID，通过 uci_Open 接口得到。
u_cstring _msg	属性字符串，具体看 Remarks 部分。
const u_object* _obj	要设置的数据缓冲区地址
u_size _objSize	要设置的数据缓冲区大小

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

目前支持的属性有：

1、 设置 UCI 接口错误消息的语言版本 中文 ： “lang:zh-hans;” 英文 ：” lang:en-Us;”

2、 订阅设备接入和移除通知 ：“devchange:1”； 可以通过接受设备接入和移除事件，来处理设备掉线和在线重新连接等关于设备连接的逻辑，而不是通过发送心跳包的方式。

示例：

```
uci_SetAttribute(INVALID_SESSION, _T("lang:zh-Hans;devchange:1;"), nullptr, 0);
```

注意： *_session* 必须是 INVALID_SESSION.

20、 uci_SetNotify

添加 UCI 事件订阅。

Syntax:

```
u_status _UCIAPI uci_SetNotify( UCIMSGProc _pNotify )
typedef int(__stdcall *UCIMSGProc)(UCIMSG* _msg);
```

Parameters ：

UCIMSGProc _pNotify	响应订阅的回调函数

Return Value:

<0 表示错误，错误码详见：UCI 错误码

Remarks ：

如果要添加设备移除和接入事件的订阅可以使用该接口添加响应接口。

数据结构：

1、QParams（查询参数）

```
typedef struct _QParams {
        //@brief : 要查询的通信节点类型
        //@remarks : 从 enum NodeType 取值！ 通过标志位存放类型值
        //@eg ： NodeType::USB | NodeType::LAN
        u_int32   Type;
        //@brief : 端口集端口数量
        u_int32   PortCount;
        //@brief : 端口集
        u_int32*  Ports;
        //@brief : PVID 集的 PVID 的个数
        //@remarks :
        u_int32   PVIDCount;
        //@brief : PID 和 VID 集
        //@remarks : 使用 MakePVID 创建，GetPID 和 GetVID 获取
        u_int32*  PVID;
    }QParams, *PQParams;//@brief : 查询设备时的配置参数
```

引入文件： ucidef.h
可以使用接口 UCI_CreateQParam 来创建该结构数据：

eg：

```
QParams qp;
    int pvids[10] = { MakePVID(0x1234, 0x5345) };
```

只查询 USB 接口：

```
    UCI_CreateQParam(qp, NodeType::USB, NULL, 0, pvids, 1);
```

查询 USB 和 LAN 接口：

```
    int ports[10] = { 4162, 8000 };
    UCI_CreateQParam(qp, NodeType::USB | NodeType::LAN, ports, 2, pvids, 1);
```

2、WParams（写操作参数）

```
//@brief : 写数据接口参数包
//@remarks : 可通过接口 uci_CreateWParams 创建
typedef struct _WParams {
    //@brief : 命令字符串，以'\0'结尾. 具体格式见文档;
    u_cstring CMDString;
    //@brief : 返回的数据量数据，具体意义视具体命令而定;
    u_uint32  RetCount;
    //@brief : 读超时
    u_uint32  Timeout;
}WParams, *PWParams;
```

其中:

CMDString 的格式，应该视具体的设备通信协议而定。

3、RParams（读操作参数）

```
//@brief : 读数据接口的参数包
//@remarks : 普通读数据命令可通过 uci_CreateRParams 接口创建
typedef struct _RParams {
    //@brief : 命令字符串，以'\0'结尾. 具体格式见具体协议文档;
    u_cstring CMDString;
    //@brief : 要返回的数据量，具体意义视具体命令而定，通常和要读取\查询的数据是对应的;
    u_uint32  RetCount;
    //@brief : 读超时
    u_uint32  Timeout;
    //@remarks : 普通操作不使用，可空置;在查询设备时用到，
    //数据类型为 QParams，其它以文档为准!
    u_buf     ExtraData; //reserve
    //@brief : 附加数据的长度
    u_size    ExtraDataLen;
}RParams, *PRParams;
```

此结构在 uci_Query 、 uci_QueryNodes 和 uci_Read 等接口中使用。

其中，ExtraData 和 ExtraDataLen， 只在特殊情况下会遇到，普通读操作不会用到，可以为空。何时用到，视具体协议而定。

4、WFileParams（写文件参数）

```
//写文件操作的参数包
typedef struct _WFileParams {
    //@brief : 命令字符串，以'\0'结尾. 具体格式见文档;
    u_cstring  CMDString;
    //要写入的文件路径（包括文件名）
    u_cstring  FilePath;
    //写超时(单位: ms)
    u_uint32   TimeOut;
}WFileParams, *PWFileParams;
```

CMDString ：命令字符串见具体协议文档。
FilePath ： 是全路径，包含文件名和后缀名。

5、RFileParams（读文件参数）

```
//读文件接口的参数包
typedef struct _RFileParams {
    //@brief : 命令字符串，以'\0'结尾. 具体格式见文档;
    u_cstring   CMDString;
    //要读取的文件的保存路径（包括文件名）
    u_cstring   FilePath;
    //读超时(单位: ms)
    u_uint32    TimeOut;
}RFileParams, *PRFileParams;
```

CMDString ：命令字符串见具体协议文档。
FilePath ：包含文件名和后缀的全路径，将在该路径下创建文件并保存数据到文件。

6、PCommandParams （命令参数）

```
typedef struct _CommandParams {
    //@brief : 命令字符串，以'\0'结尾. 具体格式见文档;
    u_cstring CMDString;
    //@brief : 命令参数 1
    u_uint32  Param1;
    //@brief : 命令参数 1
    u_uint32  Param2;
    //@brief : 超时
    u_uint32  Timeout;
}CommandParams, *PCommandParams;
```

CMDString ：视具体协议而定。

7、Node （节点/设备接口描述）

```
//@brief : 查询到的设备节点的参数
//@remarks :
typedef struct _Node {
    //@brief : 通信接口方式
    NodeType Type;
    //@brief : 设备名(机型名)
    //@remarks : 在本UCI库导出时，该名称为UCI库协议一致的名称，
    //           在已出厂机型未统一设置内置名称时，UCI库内部自动匹配为UCI协议设置的名字!
    u_tchar   Name[50];
    //@brief : 设备类型
    //@remarks : 信号源 = SG, 示波器 = DSO;
    u_tchar   DevType[10];
    //@brief : LAN口参数
    LANDescriptor LAN;
    //@brief : USB接口参数
    USBDescriptor USB;
    //@brief : 连接字符串
    u_tchar   UCIAddr[256];
    //@brief : 序列号
    u_tchar   SN[50];
    //@brief : 设备状态
    u_status  Status;
    //@brief : 设备实际显示名称
    u_tchar   IDN[20];
}Node, *PNode;

//@brief : 通信节点类型
//@remarks : 在QParams中是按位与，在Node中取的是enum值。
typedef enum _NodeType{
    LAN = 0x0001,
    USB = 0x0010,
}NodeType;

//@brief : USB设备的描述符
//@remarks :
typedef struct _USBDescriptor {
    //@brief : PID
    u_ushort  PID;
    //@brief : VID
    u_ushort  VID;
    //@brief : 地址
```

```
    u_ushort  Addr;
}USBDescriptor;

//@brief : IP地址
//@remarks : 填充顺序为 f1(192).f2(168).f3(1).f4(253) - 小端模式
typedef union _IPAddr {
    struct { u_byte f1, f2, f3, f4; } Field;
    u_int32 Addr;
}u_IPAddr;

//@brief : LAN口通信的设备描述符
//@remarks :
typedef struct _LANDescriptor {
    //@brief : IP地址（字符串类型）
    u_tchar    IP[16];
    //@brief : IP地址
    u_IPAddr   Addr;
    //@brief : 网络端口
    //@remarks : TCPIP连接用的端口号
    u_ushort   Port;
}LANDescriptor;
```

通用说明：

1、UCI 错误码

所有 UCI 接口的返回值都的意义都是一致的，那就是<0 代表错误，错误码定义见 ucidef.h（始终以该文件中的定义为准）.

错误码对应的错误消息可以通过 uci_GetLastError 接口获取。

可以使用宏定义：UCISUCCESS 和 UCIERR 判断接口的返回值。

可以切换消息的语言版本： 中文 ： “lang:zh-hans;” 英文 ：” lang:en-Us;”

```
uci_SetAttribute(INVALID_SESSION, _T("lang:zh-Hans;devchange:1;"), nullptr, 0);
```

错误码	描述
0	成功！
-5	请求的资源正在使用
-116	请求超时
-1020	资源初始化错误
-1019	无效的会话 ID
-1018	操作超时
-1017	失败
-1016	不支持的操作
-1015	内存资源不足
-1014	系统忙
-1013	通信异常，不可逆！
-1012	通道未打开！
-1011	WinAPI 错误，此时，说明是 windows 平台，则可以使用 WIN API GetLastError 得到进一步的错误信息。
-1041	未知的错误
-1040	建立连接的字符串地址格式错误
-1039	连接还未建立
-1038	连接已断开
-1037	不支持的通信方式
-1060	未发现指定的设备
-1059	不支持的设备

-1058	需要先执行查询设备的操作
-1057	设备不匹配
-1056	查询网络设备失败
-1055	USB 设备地址只在执行查询设备操作后才有效
-1054	未找到 U 盘!
-1053	按键已经锁定!
-1080	命令字符串格式错误
-1079	只支持单条命令
-1078	一个命令只支持一个属性
-1077	不支持的命令
-1076	发送命令失败
-1075	协议数据格式错误
-1074	设备端写文件到 flash 失败!
-1073	未发现有效回复数据
-1072	命令错误，请检查相关协议!
-1071	表达式无效!
-1070	在当前调制模式下不可以加载任意波形文件
-1069	请使用（读/写）文件接口
-1068	请使用（读/写）参数接口
-1120	数字溢出!
-1119	超出范围!
-1118	数据未全部读取完!
-1117	数据校验失败!
-1116	无效的数据!
-1115	压缩错误!
-1114	解压缩错误!
-1113	数据传输错误!
-1112	数据传输中断!
-1111	无数据可读!
-1100	参数错误
-1099	参数提供的空间太小
-1098	所提供的文件名太长(最长 50 个字节)
-1097	参数所给的数据大小与协议不匹配(防止数据错误)
-1140	不能访问!
-1139	一个未指定的错误发生!
-1138	文件未找到!
-1137	无效的路径!
-1136	超出了文件可被打开的数量
-1135	无效的文件句柄

-1134	当前工作目录不能被移除
-1133	磁盘满
-1132	设置文件指针错误
-1131	出现硬件错误
-1130	SHARE.EXE 尚未加载，或共享区域锁定!
-1129	尝试锁定已锁定的区域
-1128	磁盘已满
-1127	已到达文件结尾
-1126	写文件到磁盘失败
-1125	文件长度超出范围

2、如何获取设备访问地址？

获取地址（主要指 USB 通信地址）的方式有两个：

1）、直接从机型对应的编程手册中获取；

由于大多数设备并没有通过 SN 来区分设备，所以，设备地址大多是一样的，那么可以直接从机型对应的编程手册中获取设备地址，然后直接使用 uci_OpenX 建立会话即可。但如果相同型号的设备同时连接，此时，就需要在线查找了。

2）、通过查询接口在线查询获取。

可通过 uci_QueryNodes 和 uci_QueryNodesX 两个接口查询在线的设备列表。请根据情况选择想使用的接口。

3、如何检查设备是否在线？

在程序刚开始时，可以使用查询接口（uci_QueryNodes 和 uci_QueryNodesX）查询设备，从而获知设备是否在线。在程序建立会话后，一般的方法是维护一个心跳包，即定时的访问设备，根据返回的状态码来判断设备是否在线，但可以采用更加简易的方式。那就是通过“订阅设备插拔通知”的方式实现，这样无论是 USB 通信还是网络通信都可以在委托接口中可以收到设备连接和移除的通知。具体实现方式，请参考口 *uci_SetAttribute* 和 *uci_SetNotify* 说明。

注意：这里是以 Windows 平台为例。