TERMEX

ТЕРМОМЕТРЫ ЛАБОРАТОРНЫЕ ЭЛЕКТРОННЫЕ LTA

Библиотека lta.dll

СОДЕРЖАНИЕ

Настоящее описание распространяется на библиотеку `lta.dll` и содержит сведения, необходимые для использования библиотеки в составе прикладного программного обеспечения (ПО), предназначенного для управления работой термометра.

Библиотека распространяется бесплатно.

1 ОБЩИЕ СВЕДЕНИЯ

1.1 Назначение

1.1.1 Динамически подключаемая библиотека `lta.dll` предназначена для осуществления обмена данными между прикладной программой и термометрами LTA, подключенными к USB порту компьютера.

1.1.2 Библиотека совместима с операционной системой `MS Windows XP ServicePack2` и старше.

1.2 Состав

1.2.1 Библиотека распространяется в виде трех файлов: файла библиотеки `lta.dll`, файла статической библиотеки импорта `ltadll.lib` и заголовочного файла `ltadll.h`

1.3 Загрузка и выгрузка библиотеки

1.3.1 Динамическая загрузка и выгрузка библиотеки в память использующего ее потока осуществляется стандартным способом с использованием функций Windows `LoadLibrary()` и `FreeLibrary()`.

1.3.2 Загрузка/выгрузка библиотеки может быть выполнена неявно с использованием статической библиотеки импорта (`ltadll.lib`), которая должна быть подключена к проекту.

2 ТИПЫ ДАННЫХ

Функции библиотеки используют следующие типы данных.

```
typedef uint32_t Lta_handle;
```

Дескриптор устройства. Используется для идентификации подключенного термометра функциями библиотеки. Нулевое значение обозначает отсутствие связанного с ним термометра.

```
typedef uint32_t Lta_error;
```

Тип кода ошибки, возвращаемого функциями библиотеки.

Значения кодов ошибок определены как:

```
enum Lta_error_code
{
    lta_no_error = 0U,              // Нет ошибок.
    lta_invalid_handle = 1U,        // Указанный дескриптор неверный (например,
                                    // термометр уже отключен).
    lta_win32api_error = 2U,        // Вызов функции WIN32 API (ReadFile,
                                    // WriteFile, ...) завершился с ошибкой.
    lta_invalid_param = 3U,         // Параметр, переданный функции, имеет
                                    // недопустимое значение.
    lta_timeout_error = 4U,         // Истек таймаут.
    lta_recv_overflow_error = 5U, // Переполнение приемного буфера. Лишние
                                    // данные отброшены.
    lta_format_error = 6U           // Формат ответа термометра неверный
};
```

```
typedef void (*Lta_plug_event)(Lta_handle);
typedef void (*Lta_unplug_event)(Lta_handle);
typedef bool (*Lta_enumerate_event)(Lta_handle);
```

Типы указателей на функции обратного вызова по событиям подключения/отключения термометра и нумерации. События генерируются синхронно из `LtaEnumerate()`.

```
typedef void (*Lta_data_event)(Lta_handle h, const void* buf, uint16_t size);
```

Тип указателя на функцию обратного вызова по событию прихода данных от указанного термометра. Событие генерируется асинхронно из внутреннего вспомогательного потока чтения. Данный поток чтения запускается при установке обработчика события функцией `LtaSetDataHandler()`и останавливается при установке нулевого обработчика. Внутри обработчика может потребоваться синхронизация с основным потоком приложения. Внутри обработчика нельзя вызывать никакие другие функции библиотеки.

`h` — дескриптор термометра, от которого пришли данные;

`buf` — байтовый буфер с данными;

`size` — количество принятых байтов.

```
typedef void (*Lta_device_change_event)();
```

Тип указателя на функцию обратного вызова по событию изменения конфигурации подключенных термометров. Событие генерируется асинхронно из внутреннего вспомогательного потока при подключении или отключении термометров. В ответ на наступление события основной поток приложения может вызвать LtaEnumerate() и определить изменения в конфигурации термометров. Внутри обработчика может потребоваться синхронизация с основным потоком приложения. Внутри обработчика нельзя вызывать никакие другие функции библиотеки.

3 ЭКСПОРТИРУЕМЫЕ ФУНКЦИИ

Экспортируемые библиотекой функции используют соглашение о вызове `stdcall`.

Все функции библиотеки, за исключением `LtaInit()` и `LtaDeinit()`, являются потокобезопасными.

3.1 LtaInit()

```
bool LtaInit();
```

Выполняет инициализацию. Должна вызываться до вызова любой другой функции. Если инициализация выполнена с ошибками (результат `false`), то пользоваться остальными функциями библиотеки нельзя.

3.2 LtaDeinit()

```
void LtaDeinit();
```

Выполняет очистку перед завершением работы. Должна вызываться непосредственно перед выгрузкой библиотеки.

3.3 LtaIsHadleValid()

```
bool LtaIsHandleValid(Lta_handle h);
```

Возвращает `true`, если дескриптор `h` валидный, т.е. ссылается на подключенный термометр, `false` — в противном случае.

3.4 LtaSetXXXHandler()

```
void LtaSetPlugHandler(Lta_plug_event cb);
void LtaSetUnplugHandler(Lta_unplug_event cb);
void LtaSetEnumerateHandler(Lta_enumerate_event cb);
void LtaSetDeviceChangeHandler(Lta_device_change_event cb);
void LtaSetDataHandler(Lta_handle h, Lta_data_event cb);
```

Каждая из функций устанавливает обработчик на соответствующее событие. Для отключения обработчика, в функцию следует передать нулевой указатель (`cb = 0`).
Функция `LtaSetDataHandler()` устанавливает обработчик `cb` для указанного термометра `h`.

3.5 LtaEnumerate()

```
void LtaEnumerate();
```

Выполняет нумерацию термометров, подключенных к компьютеру. В процессе выполнения, функцией генерируются события `Lta_unplug_event` для всех термометров, отключенных с момента прошлой нумерации, `Lta_plug_event` для всех термометров, подключенных с момента прошлой нумерации и `Lta_enumerate_event` для всех термометров, подключенных на момент вызова `LtaEnumerate()`. Если обработчик события `Lta_enumerate_event` возвращает `false`, то событие `Lta_enumerate_event` для оставшихся термометров больше не генерируется.

3.6 LtaCheckoutBySerialNumber()

```
Lta_handle LtaCheckoutBySerialNumber(const wchar_t* serial);
```

Возвращает дескриптор термометра по его серийному номеру. Если такого термометра не обнаружено, или `serial` невалидная строка, то возвращает `0`.

`serial` — нуль терминированная строка символов `wchar_t`, содержащая серийный номер термометра.

3.7 LtaGetSerialNumber()

```
Lta_error LtaGetSerialNumber(Lta_handle h, wchar_t* buf, uint32_t size);
```

Копирует в указанный буфер строку с серийным номером термометра. Если строка серийного номера имеет размер больше чем размер предоставленного буфера, то строка с номером обрезается. Скопированная строка всегда завершается нулевым символом.

В зависимости от успешности выполненной операции может вернуть следующие коды ошибок: `lta_no_error`, `lta_invalid_handle`, `lta_invalid_param`.

`h` — дескриптор термометра, для которого выполняется запрос.

`buf` — буфер, куда будет скопирована строка с серийным номером;

`size` — размер предоставленного буфера.

3.8 LtaSend()

```
Lta_error LtaSend(Lta_handle h, const void* buf, uint32_t size);
```

Отправляет указанному термометру заданное количество байт из буфера.

В зависимости от успешности выполненной операции может вернуть следующие коды ошибок: `lta_no_error`, `lta_invalid_handle`, `lta_invalid_param`, `lta_win32api_error`.

`h` — дескриптор термометра, которому посылаются данные;

`buf` — указатель на буфер с данными, который внутри функции приводится к `const uint8_t*`.

`size` — количество отправляемых байт данных;

3.9 LtaReceive()

```
Lta_error LtaReceive(Lta_handle h, void* buf, uint32_t size,
                     uint32_t tm, uint32_t* recv);
```

Получает от указанного термометра и копирует в пользовательский буфер ответ термометра.

В зависимости от успешности выполненной операции может вернуть любой код ошибки.

`h` — дескриптор термометра, из которого читаются данные;

`buf` — указатель на приемный буфер, который внутри функции приводится к `uint8_t*`.

`size` — размер приемного буфера в байтах;

`tm` — величина таймаута в мс. Рекомендуемое значение 2000.

`recv` — если не ноль, то в `*recv` будет содержаться количество реально скопированных в приемный буфер байт;

Если функция вернула `lta_timeout_error` или `lta_recv_overflow_error`, то в приемном буфере могут содержаться валидные данные, полученные до возникновения ошибки. Количество таких данных сохраняется в `*recv`.

3.10 LtaFlushInputQueue()

```
Lta_error LtaFlushInputQueue(Lta_handle h);
```

Удаляет все непрочитанные данные из внутреннего буфера указанного термометра.

В зависимости от успешности выполненной операции может вернуть следующие коды ошибок: `lta_no_error`, `lta_invalid_handle`, `lta_win32api_error`.

`h` — дескриптор термометра, внутренний буфер которого очищается;

4 ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

4.1 Законченный пример утилиты

Пример консольной утилиты на C++, позволяющей пользователю ввести команду для термометра и вывести его ответ и любые другие данные, принятые от термометра.

```
//---------------------------------------------------------------------------
//  filename:   test_dll.cpp
//  Тест-программа для lta.dll
//  При запуске программа:
//  1. Выполняет нумерацию ЛТА термометров и открывает первый попавшийся.
//  2. В цикле получает команды от пользователя и выполняет их.
//---------------------------------------------------------------------------
#include <tchar.h>
#include <iostream>
#include <string>
#include <boost/thread/mutex.hpp>
#include <boost/algorithm/string.hpp>
#include "ltadll.h"
#pragma comment(lib,"ltadll.lib")

namespace
{
// Простейшее обеспечение потокобезопасности std::cout.
class sync_cout
{
public:
    sync_cout() { mx.lock(); }
    ~sync_cout() { mx.unlock(); }

    template<typename T>
    sync_cout& operator<<(const T& t)
    {
        std::cout << t;
        return *this;
    }
private:
    static boost::mutex mx;
};
```

```
// Прототипы обработчиков соответствующих событий.
void __stdcall on_data(Lta_handle h, const void* buf, uint16_t size);
bool __stdcall enumerate_handler(Lta_handle h);
// Печатает приглашение на ввод.
void prompt();
// Выполняет нумерацию термометров.
void do_enumeration();
// Читает ответ термометра и выводит его в консоль.
Lta_error read_ack(Lta_handle h);

// Дескритпор текущего устройства.
Lta_handle cur_dev = 0;
// Для потокобезопасности std::cout
boost::mutex sync_cout::mx;

// Объект класса на время своего существования отключает обработчик события
// Lta_data_event и включает его после своего уничтожения.
class Ondata_guard
{
public:
    Ondata_guard() { LtaSetDataHandler(cur_dev, 0); }
    ~Ondata_guard() { LtaSetDataHandler(cur_dev, on_data); }
};

} // ns

//============================================================================
// I M P L E M E N T A T I O N
//============================================================================

//----------------------------------------------------------------------------
int _tmain(int argc, _TCHAR* argv[])
{
    if ( !LtaInit() )
    {
        std::cout << "lta.dll initialization error\n";
        return 0;
    }
```

```
std::cout <<
    "\n\n-----------------------------------------------------------------\n"
    "                 TERMEX LTA console example                      \n"
    "Comands:\n"
    "  :enum - performs enumeration for devices;\n"
    "  :flush - flush input queue for the current device;\n"
    "  :quit - quit the program;\n"
    "\n"
    "All other commands are sent to the current device.\n"
    "-----------------------------------------------------------------\n";

// Обработчик для события нумерации, в котором выбирается устройство
// для работы.
LtaSetEnumerateHandler(enumerate_handler);
// Флаг завершения работы.
bool exit = false;

// Основной цикл.
Lta_error err = lta_no_error;
while ( !exit )
{
    // Выполняется нумерация устройств.
    do_enumeration();

    // Командный цикл.
    while (true)
    {
        prompt();
        std::string cmd;
        std::getline(std::cin, cmd);
        boost::trim(cmd);
        if (cmd.empty()) continue;

        if (cmd == ":enum") break;    // Переход к основному циклу,
                                      // начинающийся с нумерации.
        else if (cmd == ":quit") { exit = true; break; }
```

```
            else if (cmd == ":flush")
            {
                err = LtaFlushInputQueue(cur_dev);
                if (lta_no_error == err) sync_cout() << "OK\n";
                else if (lta_win32api_error == err)
                    sync_cout() << "winAPI error.\n";
                else if (lta_invalid_handle == err)
                {
                    sync_cout() << "device unplugged\n";
                    break;  // К нумерации.
                }
            }
            else
            {
                // Введена строка для отправки устройству.
                // Отправка команды и получение ответа выполняется синхронно.
                Ondata_guard dg;
                cmd.push_back('\n');
                err = LtaSend(cur_dev, cmd.c_str(), cmd.size());
                if (lta_win32api_error == err || lta_invalid_handle == err)
                {
                    sync_cout() << "device unplugged\n";
                    break;  // К нумерации.
                }
                else
                {
                    // Отправка успешна, ожидаем ответ.
                    err = read_ack(cur_dev);
                    if (lta_win32api_error == err || lta_invalid_handle == err)
                    {
                        sync_cout() << "device unplugged\n";
                        break;  // К нумерации.
                    }
                    else if (lta_no_error != err)
                        sync_cout() << "read ack with error: " << err;
                    sync_cout() << '\n';
                }
            }
        }
```

```
    }
    return 0;
}

namespace
{

//----------------------------------------------------------------------------
// Асинхронный обработчик прихода данных.
void __stdcall on_data(Lta_handle h, const void* buf, uint16_t size)
{
    // Размер данных от термометра в одном репорте не более 60 байт.
    // Буфера в 64 байта достаточно.
    char tbuf[64];
    const char* t = static_cast<const char*>(buf);
    std::copy(t, t + size, tbuf);
    if ( tbuf[size-1] == '\x04' )
        tbuf[size-1] = '\0';
    tbuf[size] = '\0';
    sync_cout() << tbuf;
}


//----------------------------------------------------------------------------
// Обработчик события нумерации.
bool __stdcall enumerate_handler(Lta_handle h)
{
    wchar_t ser[16] = { L'\0' };
    LtaGetSerialNumber(h, ser, 16);
    std::wcout << "  " << '\'' << ser << '\'' << '\n';
    if (0 == cur_dev)  // Если текущее устройство еще не выбрано,
    {                  // выбираем его.
        cur_dev = h;   //
        LtaSetDataHandler(h, on_data);
        return false;       // Останавливаем дальнейшую нумерацию.
    }
    return true;
}
```

```
//-----------------------------------------------------------------------------
// Выводит приглашение на ввод команды.
void prompt()
{
    sync_cout() << "\n";
    if (LtaIsHandleValid(cur_dev))
    {
        wchar_t ser[16] = { L'\0' };
        LtaGetSerialNumber(cur_dev, ser, 16);
        std::wcout << ser;
    }
    sync_cout() << ">";
}

//-----------------------------------------------------------------------------
// Выполняет нумерацию устройств с выводом результатов.
void do_enumeration()
{
    sync_cout() << "enumerating ... \n";
    cur_dev = 0;
    LtaEnumerate();
    if (0 == cur_dev) sync_cout() << "no devices found.\n";
}

//-----------------------------------------------------------------------------
// Синхронно читает ответ от термометра и выводит его в консоль.
Lta_error read_ack(Lta_handle h)
{
    char buf[1024];
    uint32_t received;
    Lta_error res = LtaReceive(h, buf, 1024, 2000, &received);
    if ( lta_no_error == res || res > lta_invalid_param )
    {
        buf[received] = '\0';
        sync_cout() << buf;
    }
    return res;
}
} // ns
```