Tira API v. 0.4

Revision History

v. 0.4 Sep 2005

Support for transmitting CCF is added Capture in CCF format is added

v. 0.3 Sep 2003

Introduced functions for Visual Basic support tira_cleanup is introduced

applies to DLL version 1.03

v. 0.2 Jul 2003

Calling convention of the callback is changed

Sep 2005, (C) Home Electronics

This document describes functions exported by Tira2.dll that provides simple and lightweight interface to Tira-2 USB IR transmitters. This DLL takes care of Tira's low level protocol and especially of handling IR data, such as preparing data for sending to Tira. For technical support regarding this API or Tira in general please email to tech-support@home-electro.com.

Because the API is supplied as a separate DLL, upgrades are easy, and compatibility with future devices is ensured. All you need is to replace the DLL with a newer version.

All functions will return 0 upon success. Non-zero results indicates error. However, error codes are not defined unless specified for individual functions.

```
extern "C" __stdcall int tira_init (void);
```

This function must be called before any of the other functions from this API. It creates and initialize Tira service thread.

```
extern "C" __stdcall int tira_cleanup (void);
```

This function releases all resources used by the DLL and terminates Tira service thread. After calling this function it is safe to unload the DLL, if necessary.

```
extern "C" __stdcall int tira_start ( int PortID );
```

Connects to Tira installed on the specified com port. (Although Tira is a USB device, a virtual COM port is created to make software development easier).

Note that port numbers start from zero. That means that "0" corresponds to COM1, "1" corresponds to COM2 and so on.

If you use Tira with D200X driver, you need to use portID 1024.

Tira must be installed on COM port from 2 to 256.

```
typedef int (__stdcall * tira_six_byte_cb) (const char * eventstring);
extern "C" __stdcall int tira_set_handler ( tira_six_byte_cb cb );
```

When Tira receives IR signal the specified callback function will be called. Different IR signals are represented by different event strings. Event strings are null-terminated, 13 bytes long strings. Here is the example of six byta data string "5B8700001212". Note that the memory holding the event-string will be reused upon returning from the callback. Therefore the data from the string must be copied elsewhere.

Note that the callback function is executed *in a separate thread*. Therefore the callback function must be thread-safe.

This function presents alternative way of receiving "6 bytes" IR codes, without having to use callback function. This function was introduced for use with Visual Basic applications, there callback approach does not work.

```
extern "C" __stdcall int tira_stop ();
```

This function disactivates Tira.

```
extern "C" __stdcall int tira_start_capture ();
```

Activates capture mode. In capture mode Tira yields complete information required to reproduce an IR signal. After that user is to put a remote control about two inches from Tira and send an IR code to Tira.

After switching to capture mode periodically call <code>tira_get_captured_data</code>. When returned size and the pointer to a data is non-zero that indicates that Tira has received IR signal. The data pointer returned by this function contains representation of the IR signal that can be used later for transmission.

The data pointer is a heap allocated memory and your application is responsible for proper freeing it once it is not needed. Basically, for every successful capture a new memory block is allocated. If you do not free it, memory leak will occur. In order to free the memory block use tira_delete.

```
extern "C" __stdcall int tira_get_captured_data_vb (
   unsigned char data[],
   int* size );
```

Alternative function for retrieving captured IR codes. Unlike <code>tira_get_captured_data</code>, <code>tira_get_captured_data_vb</code> does not allocate new memory block. Instead it copies IR data into user supplied memory block. <code>*size</code> must be set to the size of the memory block prior to calling this function. Upon successful return from the call <code>size</code> is set to the actual number of bytes required for the IR code. Use this functions for developing Visual Basic applications. See supplied sample application for usage example.

```
extern "C" __stdcall int tira_cancel_capture ();
```

This function cancels capture mode.

```
extern "C" __stdcall int tira_delete (const unsigned char* ptr);
```

This function will free the memory block. Only memory blocks previously acquired from tira get captured data can be freed with this function.

```
extern "C" __stdcall int tira_access_feature (
    unsigned int FeatureID,
    bool Write,
    unsigned int* Value,
    unsigned int Mask
);
```

This allows read and write various parameters of the device and the library.

| FeatureID | Description |
|------------|--|
| 0xF0000000 | Controls capture mode. When set to 0, tira_cature returns data in proprietary format. When set to 1, tira_capture returns data in CCF format. Values other than 0 or 1 are not-defined. Mask is not used and should be set to 0. |
| | By default the capture is performed in proprietary format. |

Return Value:

- 0 Returned on success.
- 1 Function failed, error accessing the feature
- 2 Function failed, feature is not implemented

Example (sets the library to capture in CCF mode):

```
extern "C" __stdcall int tira_transmit (
   int Repeat,
   int Frequency,
   const unsigned char* IRData,
   const DataSize );
```

This function transmits IR code.

Parameters:

Data points to a data previously received from get captured data.

DataSize is the size of the data. Again, the size of the data must be the same as received from get_captured_data.

Alternatively Data can point to a valid CCF strings. (CCF is format for IR codes widely used by various remote controls).

Repeat indicates a number of times the IR code to be repeated. (Some equipment will not react even to a valid IR code if it is not repeated several times. Repeat == 0 means the code to be sent once, Repeat == 1 means the code to be sent twice, and so on.

Note that in most cases you can not simply call tira_transmit several times to send repeat codes. Repeated codes often differs from the initial ones. Plus delays between repeats must be strictly observed. All this is handled by Tira.

Frequency refers to modulation frequency of the transmitted IR signal. Actual frequency of the IR signal is embedded in the IR data returned by $tira_capture$. You need to set this parameter to -1.

Return Value:

On success tira transmit returns 0.

Invalid Parameters: -1. The function will return -1 if the IRData is not recognized or CCF string is not complete or not supported.

Example:

```
tira_transmit(
    3, -1, "0000 0069 0001 0002 0015 0030 0030 0030 0045 0015", 59);
```

On the next page you will find a complete listing of a sample C/C++ applications that shows how every call of this API is used.

```
//-----
// Tira Sample application
#include <windows.h>
#include <conio.h>
#include <iostream>
using namespace std;
int Error() {
   cout << "Last Error returned : " << GetLastError() << "\n";</pre>
  return 0;
typedef int ( stdcall * tira six byte cb) (const char * eventstring);
typedef int (__stdcall * t_tira_init) ();
typedef int (__stdcall * t_tira_set_handler) (tira_six_byte_cb cb);
typedef int (__stdcall * t_tira_start) ( int PortID);
typedef int (__stdcall * t_tira_stop) ();
typedef int (__stdcall * t_tira_start_capture) ();
typedef int (__stdcall * t_tira_cancel_capture) ();
typedef int (__stdcall * t_tira_get_captured_data)
                   (const unsigned char** data, int* size );;
typedef int (__stdcall * t_tira_transmit)
      (int repeat, int frequency, const unsigned char* data, const dataSize );
typedef int ( stdcall * t tira delete) (const unsigned char* ptr);
t_tira_stop
                    p_tira_stop = 0;
t_tira_start_capture p_tira_start_capture = 0;
t_tira_cancel_capture p_tira_cancel_capture = 0;
t_tira_get_captured_data p_tira_get_captured_data = 0;
p_tira_delete = 0;
t tira delete
int stdcall OurCalback(const char * eventstring) {
   cout << "IR Data " << eventstring << '\n';</pre>
  return 0;
};
int Help();
int main() {
  // Load the DLL
  HMODULE handle = LoadLibrary("Tira2.dll");
  if ( handle == 0 ) return Error();
   void* last;
   last = p_tira_init = GetProcAddress(handle, "tira_init");
   if ( last == 0 ) return Error();
   last = p tira set handler = (t tira set handler) GetProcAddress(handle,
"tira_set_handler");
   if ( last == 0 ) return Error();
   last = p_tira_start = (t_tira_start) GetProcAddress(handle, "tira_start");
   if ( last == \overline{0} ) return \overline{E}rror\overline{()};
   last = p_tira_stop = (t_tira_stop) GetProcAddress(handle, "tira_stop");
   if ( last == 0 ) return Error();
   last = p tira start capture = (t tira start capture) GetProcAddress(handle,
"tira_start_capture");
   if ( last == 0 ) return Error();
   last = p_tira_cancel_capture = (t_tira_cancel_capture) GetProcAddress(handle,
"tira cancel capture");
   if ( last == 0 ) return Error();
   last = p tira get captured data = (t tira get captured data) GetProcAddress(handle,
"tira get captured data");
```

```
if ( last == 0 ) return Error();
last = p_tira_transmit = (t_tira_transmit) GetProcAddress(handle, "tira_transmit");
if ( last == 0 ) return Error();
last = p tira delete = (t tira delete) GetProcAddress(handle, "tira delete");
if ( last == \overline{0} ) return Error();
cout << "Calling tira_init()\n\n";</pre>
p tira init();
Help();
bool CaptureActive = false;
const unsigned char* Data = 0;
            DataSize = 0;
int
cout << "\n>";
while (1) {
   int res = -1;
   Sleep(100);
   if ( CaptureActive ) {
      res = p_tira_get_captured_data(&Data, &DataSize);
      if ( Data != 0 && DataSize != 0 ){
         cout << "IR Code captured!\n";</pre>
         CaptureActive = false;
      };
   };
   char c = 0;
   if ( kbhit() )
      c = getche();
   else
      continue;
   cout << '\n';
   switch (c) {
      case '1':
case '5':
                     case '2': case '3': case '4': case '6': case '7': case '8':
                int p = c - '1';
                res = p tira start(p);
                if ( res == \overline{0} ) cout << "Tira activated\n";
             };
      case 'S':
      case 's':
             res = p tira stop();
             if ( res == 0 ) cout << "Tira disactivated\n";
             CaptureActive = false;
             break;
      case 'B':
      case 'b':
             res = p tira set handler(OurCalback);
             if ( res == 0 ) cout << "Callback activated\n";</pre>
             break;
      case 'C':
      case 'c':
             if ( Data != 0 ) {
                cout << "Disposing captured data...\n";</pre>
                res = p tira delete(Data);
                if ( res == \overline{0} ) cout << "Memory freed\n";
                Data = 0;
             };
             res = p tira start capture();
             if ( res == \overline{0} ) cout << "Capture activated\n";
             CaptureActive = (res == 0 );
             break;
```

```
case 'A':
         case 'a':
               res = p_tira_cancel_capture();
               if ( res == \overline{0} ) cout << "Capture disactivated\n";
               CaptureActive = false;
               break;
         case 'T':
         case 't':
               if ( Data == 0 ) {
                  cout << "There is nothing to transmit\n";</pre>
                  break;
               res = p_tira_transmit(2, /* repeat 3 times*/
                                      -1, /* Use embedded frequency value*/
                                      Data,
                                      DataSize);
               if ( res == 0 ) cout << "IR code transmitted\n";</pre>
               CaptureActive = false;
               break;
         case 'D':
         case 'd':
               res = p tira delete(Data);
               if ( res == 0 ) cout << "Memory freed\n";
               Data = 0;
               break;
         case EOF:
         case '\n':
              break;
         case 'Q':
         case 'q':
               cout << "Exiting...\n";</pre>
               return 0;
         default:
              Help();
      if ( res != 0 && res != -1 )
         cout << "Last function call failed!\n";</pre>
      cout << "\n>";
   };
   return 0;
int Help() {
            "1-8\t Open Tira on corresponding COM port\n"
  cout <<
            "\t for example, '3' opens Tira on COM3\n"
            "S\t Stops Tira\n"
            "B\t attach Callback. Application will be able to receive IR signals \n"
            "C\t activates capture mode\n"
            "A\t cancels capture mode\n"
            "T\t transmit the most recently captured IR code\n"
            "D\t dispose data allocated for IR code\n";
            "Q\t Quit\n";
};
```