Tira API v. 0.2

Jul 2003, (C) Home Electronics

This document describes functions exported by Tira2.dll that provide simple and lightweight interface to Tira-1 and 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 support@home-electro.com.

All functions will return 0 upon success. Non-zero results indicates error. However, error codes are not defined in this version of API.

```
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_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.

In this version of API Tira must be installed on COM1 through COM8.

```
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. 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.

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

This fucntion 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. (NB. Tira-1 will get timing data, but the frequency will have to be specified. Tira-2 captures frequency as well.) After that user is expected to send an IR code to Tira.

```
extern "C" __stdcall int tira_get_captured_data (
                  const unsigned char** data,
                  int* size );
```

After switching to capture mode periodically call `tira_get_captured_data`. 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 representaion 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 succesful 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_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_cancel_capture ();
```

This function cancels capture mode.

```
extern "C" __stdcall int tira_transmit (
                  int Repeat,
                  int Frequency,
                  const unsigned char* Data,
                  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`.

`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 time. `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` has to be specified if you use Tira-1. This parameter can take values from 0 to 7. The following table shows what will be the resulting frequency.

Parameter	Frequency
-1	Default
0	30.3KHz
1	32.2KHz
2	33.3KHz
3	35.7KHz
4	37.0KHz
5	38.4KHz
6	40.0KHz
7	55.5KHz

The data produced by Tira-2 already contain the frequency information. In order to make use of the embedded frequency value use `-1`. Values `0-7` will override the value embedded in data.

On next page you will find a complete listing of a sample 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";
   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_init          p_tira_init = 0;
t_tira_set_handler   p_tira_set_handler = 0;
t_tira_start         p_tira_start = 0;
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;
t_tira_transmit      p_tira_transmit = 0;
t_tira_delete        p_tira_delete = 0;

int __stdcall OurCalback(const char * eventstring) {
   cout << "IR Data " << eventstring << '\n';
   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 == 0 ) return Error();
   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 == 0 ) return Error();

cout << "Calling tira_init()\n\n";
p_tira_init();


Help();
bool CaptureActive = false;
const unsigned char* Data = 0;
int         DataSize = 0;

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";
         CaptureActive = false;
      };
   };
   char c = 0;
   if ( kbhit() )
      c = getche();
   else
      continue;

   cout << '\n';

   switch (c) {
      case  '1':    case '2':   case '3':    case '4':
      case  '5':    case '6':   case  '7':   case '8':

            {
               int p = c - '1';
               res = p_tira_start(p);
               if ( res == 0 ) cout << "Tira activated\n";
               break;
            };
      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";
            break;
      case 'C':
      case 'c':
            if ( Data != 0 ) {
               cout << "Disposing captured data...\n";
               res = p_tira_delete(Data);
               if ( res == 0 ) cout << "Memory freed\n";
               Data = 0;
            };
            res = p_tira_start_capture();
            if ( res == 0 ) cout << "Capture activated\n";
            CaptureActive = (res == 0 );
            break;
```

```
         case 'A':
         case 'a':
               res = p_tira_cancel_capture();
               if ( res == 0 ) cout << "Capture disactivated\n";
               CaptureActive = false;
               break;

         case 'T':
         case 't':
               if ( Data == 0 ) {
                  cout << "There is nothing to transmit\n";
                  break;
               };

               res = p_tira_transmit(2, /* repeat 3 times*/
                                     -1, /* Use embedded frequency value*/
                                     Data,
                                     DataSize);

               if ( res == 0 ) cout << "IR code transmitted\n";

               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";
               return 0;
         default:
               Help();
      }
      if ( res != 0 && res != -1 )
         cout << "Last function call failed!\n";

      cout << "\n>";

   };
   return 0;
}

int Help() {
   cout <<  "1-8\t Open Tira on corresponding COM port\n"
            "\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";
};
```