## System requirements:

The dll file was designed for using with all current Windows operating system versions (2011). It is implemented as a 32-bit and as a 64-bit dll, what makes it possible to use it in Windows XP, Vista and Windows 7, including the 64-bit systems and also for Win32 and AM64 applications. All older operating systems, also future systems that no longer will support Win32 or AM64 dll's, are not usable.

There is now not longer a need for installing a special device driver for the sensor devices because the dll is now using the USB communication class device driver (CDC), which is included in all current Windows versions.

## Implementation of the dll functions:

**Note:** The dll functions are all implemented as multi-threaded and they must be synchronized with the calling threads. After loading the dll, the list of the sensor devices is available after 150 ... 500 milliseconds in average.

Because of the multi-threaded implementation and the usage of the virtual COM ports, the count of sensor devices, usable at same time, is limited to 50. The virtual COM ports must have a port number between 1 and 999. But because of this the CDC class driver, included in any Windows version, can be used and there should no further driver or compatibility problems occur. There is only a need for a installation .inf file which tells the Windows operating system what driver to use for the given USB vendor and device ID's. Also, 32-bit and 64-bit systems are supported. The dll is implemented on the smallest common level of this all, so at a 32-bit dll.

The dll is exporting all functions of the previous, to support backwards compatibility. It also supports explicit ASCII- and Unicode-calls now. The pre-processor definition "UNICODE" defines what type of call is used. You always should ensure, that this is defined if you wish to use Unicode-calls. Alternatively, you also can use the suffixes 'A' or 'W' at the end of the function names to specify what call is used. For using the function names without the suffixes, automatically the ASCII code types are used if 'UNICODE' is not defined. This is implemented for backwards compatibility also.

The dll is exporting the following functions:

| Variant | Function name | Ordinal number (**obsolete**) |
|---|---|---|
| ASCII (alt) | | |
| | SensFindDevice | @1 |
| | SensReadValues | @2 |
| | SensSetHeating | @3 |
| | SensGetChangeFlag | @4 |
| | SensWaitReady | @5 |
| ASCII | | |
| | SensFindDeviceA | @11 |
| | SensReadValuesA | @12 |
| | SensSetHeatingA | @13 |
| | SensGetChangeFlagA | @14 |
| | SensWaitReadyA | @15 |
| Unicode | | |
| | SensFindDeviceW | @21 |
| | SensReadValuesW | @22 |
| | SensSetHeatingW | @23 |
| | SensGetChangeFlagW | @24 |
| | SensWaitReadyW | @25 |
| Default | | |
| | DllGetVersion | @99 |

## Error codes:

Any function, which is returning a detailed error code, will use one of the following values:

| Error code | Value | Description |
|---|---|---|
| SENS_HEATING_ENABLED | 1 | The function call was successful. No error has occurred, but the integrated heating of the sensor device is activated. This leads to an significant measurement value error, because the measured temperature value is not valid. |
| SENS_SUCCESS | 0 | The function call was successful. No error has occurred. |
| SENS_FAILED | -1 | The call was not successful. A common failure has occurred. This e.g. may happen, if an sensor access is performed while the sensor device is queried by the device searching procedure and a timeout occurs. |
| SENS_NOT_FOUND | -2 | Attempt to access an not available sensor device, maybe using an invalid serial number string or the device was already dismounted or there is not additional device in device list found. |
| SENS_UNABLE_TO_OPEN | -3 | The dll is not available to open the access using the device driver. This may e.g. happen, if the selected port is used by another application in between. |
| SENS_IO_ERROR | -4 | There has a communication error occurred between sensor device and PC. If this error occurs, mostly a technical problem is the cause. |
| SENS_TYPE_NOT_SUPPORTET | -5 | The addressed device seems to be a MELTEC sensor device, but the type is currently not supported by this dll. |
| SENS_DEVICE_NOT_READY | -6 | The addressed device is not ready now. This may happen after powering up a sensor device on an attempt to query values before they are available. Normally, the sensor devices will need some seconds until the first values are available. It is also possible that no sensor head is connected to the device. |
| SENS_RH_NOT_MEASURED | -7 | The humidity measure is currently not available, maybe it was not measured yet. |
| SENS_TEMP_NOT_MEASURED | -8 | The temperature measure is currently not available, maybe it was not measured yet. |
| SENS_INVALID_MEASUREMENT | -9 | The measure values are currently invalid. Maybe the sensor head is currently not working correctly. |
| SENS_INVALID_FUNCTION | -10 | The function is not accessible. Maybe you've attempt to access a heating element within an old sensor which don't support this. |

## Function "SensFindDevice()":

**Prototypes:**

LRESULT CALLBACK SensFindDeviceA(LONG_PTR n, LPSTR pszMask, PSENSDEVICE pDevice);

LRESULT CALLBACK SensFindDeviceW(LONG_PTR n, LPWSTR pwszMask, PSENSDEVICE pDevice);

**Description:**

The function searches the current device list for the device with the given index and fills a parameters block with the device parameters of the found device. If a sensor defined with the given index is found, the function returns SENS_SUCCESS, otherwise it returns SENS_NOT_FOUND. You can use a filter string which to reduce the selection to special sensor devices. Not used parameters must be set to NULL. To find all connected sensor devices, you can call this function inside a loop with an incrementing index number, as LONG_PTR as the SENS_SUCCESS result code is returned.

| Parameter: | Type: | Description: |
|---|---|---|
| **n** | LONG_PTR | Defines the index (n) of the searched device. The index starts with n=0. The function returns the parameters of the first found device matching the given filter parameters. It is searching all currently connected devices. |
| **pszMask**<br>**pwszMask** | LPSTR (ASCII)<br>LPWSTR (Unicode) | Pointer to a filter string or NULL. If the filter string pointer is valid, only devices are found where the filter string is matching. The filter is matching, if the type-string of the device contains the filter string somewhere. |
| **pDevice** | PSENSDEVICE<br>PSENSDEVICEA<br>PSENSDEVICEW | Pointer to a buffer with the "SENSDEVICE" structure. If a valid pointer is given and the query succeeded, the buffer is filled with the sensor device parameters. |

**Return value:**

Type LRESULT, the function returns SENS_SUCCESS it the given device is found, otherwise it returns SENS_NOT_FOUND.

**The buffer for the device parameters consists of the following structure:**

```
typedef struct _SENSDEVICEA                 Device entry, ASCII code version
{
    TCHAR       szTypeName[32];             Device type name string
    TCHAR       szSerialNo[32];             Device serial number string
    LONG_PTR    nIndex;                     Device index number

} SENSDEVICEA, * PSENSDEVICEA;

typedef struct _SENSDEVICEW                 Device entry, Unicode Version
{
    WCHAR       szTypeName[32];             Device type name string
    WCHAR       szSerialNo[32];             Device serial number string
    LONG_PTR    nIndex;                     Device index number

} SENSDEVICEW, * PSENSDEVICEW;
```

**Notes:** This function returns all relevant information from the currently connected devices. You can use multiple calls with a incrementing index number to create a list of all connected devices. A maximum of 50 devices is supported for one PC. The COM port numbers for the installed ports must be between 1 and 999.

## Function "SensReadValues()":

**Prototypes:**

LRESULT CALLBACK SensReadValuesA(PVOID Parameter, BOOL bMode, float * pfValRH, float * pfValTemp, float * pfValDew);

LRESULT CALLBACK SensReadValuesW(PVOID Parameter, BOOL bMode, float * pfValRH, float * pfValTemp, float * pfValDew);

**Description:**

This function is used for querying the current measure values from a given device. Depending on the type of the addressed device, up to three values are returned, e.g. for an UFT75-AT sensor the relative humidity, the temperature and the dew point temperature. Values for the relative humidity are typically between 0 and 100 %, the temperature and the dew point temperature between -40 and +120°C (-40 and +248°F or between 233.15 and 393.14 K).

| Parameter: | Type: | Description: |
|---|---|---|
| Parameter | PVOID | The content of the parameter is depending on the "bMode" variable. If "bMode" is SENS_READ_BY_SERIAL_NUMBER, the parameter contains a pointer to a null-terminated serial number string (ASCII or Unicode), which must contain the exact serial number of the addressed sensor device. If "bMode" is SENS_READ_BY_INDEX, the parameter is the index number of the device. It's the same index number as used while searching for the device. |
| bMode | BOOL | Given addressing mode for selecting the device (see "Parameter"). |
| pfValRH | float * | Pointer to a float variable (IEEE 754, 32-bit) which receives the measurement value for the relative humidity. The relative humidity value is possible between 0.0 and 100.0 and its unit is percent. If this pointer is NULL, this parameter is ignored. If you attempt to access a sensor device which is not supporting humidity measurements, the result is always 0.0. |
| pfValTemp | float * | Pointer to a float variable (IEEE 754, 32-bit) which receives the measurement value for the temperature. The temperature value is possible between -40.0 and +120.0°C and its unit is in °Celsius by default. If this pointer is NULL, this parameter is ignored. If you attempt to access a sensor device which is not supporting temperature measurements, the result is always -40.0. |
| pfValDew | float * | Pointer to a float variable (IEEE 754, 32-bit) which receives the measurement value for the dew point temperature. The dew point temperature value is possible between -40.0 and +120.0°C and its unit is in °Celsius by default. If this pointer is NULL, this parameter is ignored. The dew point temperature is normally not calculated inside the sensor device but in the PC. It is based on the humidity and the temperature measurement so that these both values must be available. If it is not possible to calculate the dew point temperature yet, the return value is always -40°C. |

**Return value:**

Type **LRESULT**, the function returns an error code of the type "SENS_xxx“ as result.

| Addressing modes: | Value: | Function: |
|---|---|---|
| SENS_READ_BY_SERIAL_NUMBER | 0 | The function parameter named "Parameter" contains a pointer to a null terminated string which is the device serial number of the addressed sensor device. |
| SENS_READ_BY_INDEX | 1 | The function parameter named "Parameter" contains the index number of the addressed device as used for searching the device using the function "SensFindDevice()". Please note that the index number of a specific sensor device may change for other USB configurations. Therefore it is recommended to use the serial number string for addressing a device, because this is unique, independently from the current USB configuration such as COM port numbers |

**Notes:** Not available humidity measure values are always returned as 0.0%, not available temperature measure value are returned as -40.0°C. The dew point temperature is normally calculated by the PC, what makes it necessary that the humidity value is available as well as the temperature value too. Most of the sensor devices will need some seconds after powering up, until the values are available.

## Function "SensSetHeating()":

**Prototypes:**

LRESULT CALLBACK SensSetHeatingA(PVOID Parameter, BOOL bMode, BOOL bEnable);

LRESULT CALLBACK SensSetHeatingW(PVOID Parameter, BOOL bMode, BOOL bEnable);

**Description:**

Function for activating or deactivating the heating of a UFT75-AT sensor device, if the addressed device is supporting this.

| Parameter: | Type: | Description: |
|---|---|---|
| Parameter | PVOID | The content of the parameter is depending on the "bMode" variable. If "bMode" is SENS_READ_BY_SERIAL_NUMBER, the parameter contains a pointer to a null-terminated serial number string (ASCII or Unicode), which must contain the exact serial number of the addressed sensor device. If "bMode" is SENS_READ_BY_INDEX, the parameter is the index number of the device. It's the same index number as used while searching for the device. |
| bMode | BOOL | Given addressing mode for selecting the device (see "Parameter"). |
| **bEnable** | BOOL | Flag gibt an, ob das Heizelement aktiviert (TRUE) oder deaktiviert (FALSE) werden soll. |

**Return value:**

Type **LRESULT**, the function returns an error code of the type "SENS_xxx“ as result.

**Possible addressing modes:**

| Addressing modes: | Value: | Function: |
|---|---|---|
| SENS_READ_BY_SERIAL_NUMBER | 0 | The function parameter named "Parameter" contains a pointer to a null terminated string which is the device serial number of the addressed sensor device. |
| SENS_READ_BY_INDEX | 1 | The function parameter named "Parameter" contains the index number of the addressed device as used for searching the device using the function "SensFindDevice()". Please note that the index number of a specific sensor device may change for other USB configurations. Therefore it is recommended to use the serial number string for addressing a device, because this is unique, independently from the current USB configuration such as COM port numbers |

**Notes:** Nur die UFT75-AT-Sensor-Geräten ab Firmware-Version 2.0.00 und höher unterstützen die Heizung Funktionalität. Wenn diese Funktion für den Versuch, die Heizung eines Gerätes, die nicht unterstützt wird die Heizung zu aktivieren verwendet wird, gibt es den Fehlercode "SENS_INVALID_FUNCTION". Beim Aktivieren der Heizung gelungen, die Abfrage aufruft, mit der "SensReadValues ()" Funktion, dann sind wieder die Status-Code "SENS_HEATING_ENABLED" anstelle des "SENS_SUCCESS" code. In diesem Fall auch die Messwerte werden nicht richtig, weil sie durch die Erwärmung beeinflusst wird.

## Function „ SensGetChangeFlag()“:

**Prototypes:**

BOOL CALLBACK SensGetChangeFlagA(VOID);

BOOL CALLBACK SensGetChangeFlagW(VOID);

**Description:**

This function returns TRUE, if something with the sensor configuration has changed since last time calling this (new devices added or devices removed). If nothing has changed, it returns FALSE.

| Parameter: | Type: | Description: |
|---|---|---|
| --- | VOID | No parameters. |

**Return Value:**

Type **BOOL**. The function returns TRUE, if something with the sensor configuration has changed, e.g. if some devices are connected or some are disconnected since the last call of this function. The ASCII and the Unicode variants are identically but available both, for compatibility.

**Notes:** In the case, that the call of this function returns TRUE, it is strongly recommended that a new device search is performed, using the "SensFindDevice()" function, in order to support added or removed devices. The dll is automatically detecting new sensor devices if the operating system sends an USB device changing event. The device detecting normally needs in between 150 to 500 milliseconds. the dll is supporting a maximum of 999 devices at same time, located at COM port 1 to 999. Sensor devices which are used by another thread while performing a device detection cycle, are possibly not detected.

## Function „SensWaitReady()“:

**Prototypes:**

BOOL CALLBACK SensWaitReadyA(LONG nTimeout);

BOOL CALLBACK SensWaitReadyW(LONG nTimeout);

**Description:**

This function waits for finishing of a device detection cycle for a maximum time of the given timeout value in milliseconds. It returns TRUE, if the device searching was finished within the given timeout periode, or FALSE if a timeout occurs. The function may be used by application to synchronize with the dll functions.

| Parameter: | Type: | Description: |
|---|---|---|
| **nTimeout** | BOOL | Timeout value in milliseconds. |

**Return value:**

Type **BOOL.** The function returns TRUE, if the device searching has finished within the given timeout periode in milliseconds. It returns FALSE, if the device searching is still running if the given timeout elapsed.

Notes: Because of the complete multi-threaded implementation of the dll functions, it is possible for a application to query found devices while the device searching is still performed in the background. This may cause problems if an application queries the found devices directly after staring it up, because the device searching cycle has not finished at this time, and maybe none or not all really connected devices are returned to the application. At this point the function "SensWaitRead()" can be used to synchronize the device searching with the application. This multi-threaded implementation of the device searching has a lot of advantages, e.g. is the application never blocked by any calling of a dll function, or the device searching will always need nearly the same time, because all available COM ports are queried parallel and all sensor devices should be found after 150 ... 500 milliseconds.

## Function „ DllGetVersion()“:

**Prototype:**

HRESULT WINAPI DllGetVersion(DLLVERSIONINFO * pDllVerInfo);

**Description:**

Function fills a Windows DLLVERSIONINFO structure with the version data of this dll.

| Parameter: | Type: | Description: |
|---|---|---|
| **pDllVerInfo** | DLLVERSIONINFO * | Pointer to a buffer with the structure of DLLVERSIONINFO (see "Shlwapi.h". |

**Return value:**

Type **HRESULT**, the function returns S_OK if it succeeded, or an error code (E_FAIL, E_...), see "WinError.h".

The structure DLLVERSIONINFO is defined as follows (**"Shlwapi.h"**):

```
typedef struct _DLLVERSIONINFO
{
   DWORD cbSize;
   DWORD dwMajorVersion;                   // Major version
   DWORD dwMinorVersion;                   // Minor version
   DWORD dwBuildNumber;                    // Build number
   DWORD dwPlatformID;                     // DLLVER_PLATFORM_*

} DLLVERSIONINFO;
```

## Sample code:

The following sample code demonstrates in standard "C" language, how a Windows list box control is filled with the currently connected UFT75-AT sensor devices. The second part shows how to read the values using a timer event. The code is very similar to the function of the sample application "UFTAccessTest.exe", but only particularly shown. Handling the user interface is not contained in the sample code.

1st sample code, creating a list box which is viewing the connected sensor devices (simplified):

```
#include ... insert header files here
#include "UFTAccess.h"


VOID      SetupMyListBox(HWND hWnd)
{
 SENSDEVICE      DeviceInfo;              /* sensor device info buffer */
 HWND            hItem;                   /* list box window handle    */
 LONG            i;                       /* sensor device index       */
 LONG            Index;                   /* list box item index       */

if(!SensWaitReady(500)) return;           /* wait for dll ready        */

 hItem = GetDlgItem(hWnd,                 /* query list box handle     */
                    IDC_SENSORLIST);

 SendMessage(hItem,                       /* init list box item        */
             LB_RESETCONTENT, 0, 0L);

 ZeroMemory(&DeviceInfo,                  /* clear buffer              */
            sizeof(SENSDEVICE));

 i = 0;                                   /* begin with index 0        */

 while(SensFindDevice(i, 0L, &DeviceInfo) == SENS_SUCCESS)
 {                                        /* if succeeded query        */
  sprintf(szBuffer, "%s  -  %s",          /* format list box entry     */
          DeviceInfo.szSerialNo, DeviceInfo.szDeviceID);

  Index = SendMessage(hItem,              /* add entry to list box item */
                      LB_ADDSTRING, 0, (LPARAM)szBuffer);

  if(Index != LB_ERR)                     /* if succeeded              */
  {
   SendMessage(hItem,                     /* set device index as param. */
               LB_SETITEMDATA, (WPARAM)Index, (LPARAM)i);
  }

  i++; if(i >= 999) break;                /* next device, max. 999     */
 }

 SetDlgItemText(hWnd, IDC_VAL_RH,   "---");
 SetDlgItemText(hWnd, IDC_VAL_TEMP, "---");
 SetDlgItemText(hWnd, IDC_VAL_DEW,  "---");

 SetDlgItemText(hWnd, IDC_STATUS, "Preparing...");
}
```

2nd sample code. This shows a timer event handler in the window callback function for reading measure values of a selected sensor device or updating the device list box.

```
switch(Message)                                 /* depending on message      */
 {
  case WM_TIMER:                                /* for timer messages        */
   {
    switch(wParam)                              /* depending on timer ID     */
    {
     case 0x0001:                               /* on reading new values     */
      {
       LRESULT              Status;             /* status code               */
       BYTE                 szBuffer[256];      /* auxiliary buffer          */
       HWND                 hItem;              /* item handle               */
       LONG                 Index;              /* list box selection index  */
       LPSTR                p;                  /* auxiliary string pointer  */
       float                fValRH;             /* humidity value            */
       float                fValTemp;           /* temperatur value          */
       float                fValDew;            /* dew point value           */

       hItem = GetDlgItem(hWnd,                 /* get list box window handle */
                          IDC_SENSORLIST);

       Index = SendMessage(hItem,               /* query current selection   */
                           LB_GETCURSEL, 0, 0L);

       if(Index == LB_ERR) return(FALSE);       /* break, if no sensor selected*/

       ZeroMemory(szBuffer, 256);               /* clear buffer              */

       SendMessage(hItem,                       /* read list box string      */
                   LB_GETTEXT, (WPARAM)Index, (LPARAM)szBuffer);

       p = strstr(szBuffer, "  -  ");           /* get serial number         */
       if(p) *p = 0;

       Status = SensReadValues((PVOID)szBuffer,
                               SENS_READ_BY_SERIAL_NUMBER,
                               &fValRH, &fValTemp, &fValDew);

       if(Status >= SENS_SUCCESS)               /* if succeeded              */
       {
         CheckDlgButton(hWnd, IDC_HEATING,
                        (Status == SENS_HEATING_ENABLED) ? TRUE : FALSE);

         sprintf(szBuffer, "%5.1f %%RH", fValRH);    /* format the humidity value  */
         SetDlgItemText(hWnd, IDC_VAL_RH, szBuffer);

         sprintf(szBuffer, "%+5.1f °C",  fValDew);   /* format the dew point value */
         SetDlgItemText(hWnd, IDC_VAL_DEW, szBuffer);

         sprintf(szBuffer, "%+5.1f °C",  fValTemp);  /* format temperature value   */
         SetDlgItemText(hWnd, IDC_VAL_TEMP, szBuffer);
       }
      }
      break;

    default:
     break;
   }
  }
 return(FALSE);
```

... continue