MELTEC

Embedded DLL (Win32/X64) für Zugriff auf UFT/UT-AT Sensoren

Systemvoraussetzungen:

Die DLL wurde nur für die Verwendung in allen aktuellen Windows Betriebssystemen (2011) jeweils als 32-Bit und als 64-Bit DLL implementiert und kann daher mit Windows XP, Windows Vista und Windows 7 jeweils mit der 32-Bit und der 64-Bit Version und für Win32 und AM64 Applikationen verwendet werden. Alle älteren Betriebssysteme, sowie zukünftige Windows Betriebssysteme die keine Win32 oder AM64 DLLs ausführen können, sind ungeeignet.

Es wird kein spezieller Gerätetreiber mehr benötigt, da die USB Geräte (UFT75, UT75 usw.) als virtueller COM Port über den in Windows enthaltenen Communication Class Treiber (CDC) betrieben werden.

Implementierung der DLL Funktionen:

ACHTUNG: Die DLL-Funktionen sind allesamt multi-threaded implementiert und müssen daher mit der aufrufenden Anwendung synchronisiert werden. Nach dem Laden der DLL steht die Liste der verfügbaren Sensorgeräte erst nach ca. 150 ... 500 Millisekunden zur Verfügung!

Aufgrund der multi-threaded Implementierung und der Verwendung von virtuellen COM Ports ist die Anzahl der gleichzeitig anschließbaren Sensoren auf 50 begrenzt. Die virtuellen COM Ports müssen eine Portnummer zwischen 1 und 999 besitzen. Dafür kann aber der in allen aktuellen Windows Versionen mitgelieferte CDC class device driver verwendet werden und versionsbedingte Treiberprobleme entfallen. Für die Installation ist lediglich noch eine .inf Datei nötig, die dem System mitteilt welcher Treiber für die USB ID anzuwenden ist. So werden nun sowohl 32-Bit als auch 64-Bit Betriebssysteme unterstützt. Die DLL selbst ist "auf dem kleinsten gemeinsamen Nenner" als 32-Bit-Code implementiert.

Die DLL exportiert alle Funktionen der Vorgänger DLL, unterstützt jedoch auch explizit ASCII und Unicode Aufrufe. Beim Kompilieren einer Anwendung definiert "UNICODE", welche Version einer Funktion aufgerufen wird. Dieser sollt deshalb ggf. in den Projekteinstellungen hinzugefügt werden, falls automatisch die Unicode Version der Funktionen verwendet werden soll. Alternativ kann die jeweilige Version durch den Suffix ...A() oder ...W() auch direkt aufgerufen werden. Bei der Verwendung der Funktionsnamen ohne Suffix wird immer die ASCII Code Version angewendet, solange "UNICODE" nicht definiert ist.

Die DLL exportiert die folgenden Funktionen:

Variante	Funktion	Ordinalnummer (**veraltet**)
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
Neutral		
	DllGetVersion	@99

Fehlercodes:

Funktionen, die einen detaillierten Fehlercode liefern, verwenden einen der Folgenden Werte:

Bezeichnung	Wert	Beschreibung
SENS_HEATING_ENABLED	1	Die Funktion wurde erfolgreich ausgeführt, es trat kein Fehler auf. Jedoch ist bei dem Sensor das integrierte Heizelement aktiviert, was zu einer signifikanten Abweichung der Messwerte führen kann.
SENS_SUCCESS	0	Die Funktion wurde erfolgreich ausgeführt, es trat kein Fehler auf.
SENS_FAILED	-1	Es ist ein allgemeines, nicht näher beschreibbares Problem aufgetreten. Der Fehler wird z.B. geliefert, wenn ein Sensorzugriff während der Gerätesuche erfolgt, auf diese warten muss, und dabei ein Timeout auftritt. Da nicht festgestellt werden kann, warum die Suche so lange dauert, wird ein allgemeiner Fehler geliefert.
SENS_NOT_FOUND	-2	Es wurde ein Sensor angegeben, der im lokalen USB nicht registriert ist. Z.B. wurde eine ungültige Seriennummer verwendet, oder das Gerät wurde in der Zwischenzeit entfernt, oder die Gerätesuche fand kein weiteres Gerät.
SENS_UNABLE_TO_OPEN	-3	Die DLL kann den Zugriff über den Gerätetreiber nicht öffnen. Der Fehler kann auftreten, wenn das Gerät zur Zeit von einer anderen Anwendung aus verwendet wird.
SENS_IO_ERROR	-4	Ein Kommunikationsfehler zwischen Sensor und PC ist aufgetreten. Der Fehler sollte nicht auftreten und deutet meist auf ein technisches Problem hin.
SENS_TYPE_NOT_SUPPORTET	-5	Das gewählte Gerät scheint zwar ein MELTEC Sensor zu sein, wird jedoch von dieser DLL nicht unterstützt.
SENS_DEVICE_NOT_READY	-6	Das angesprochene Gerät ist (noch) nicht bereit. Der Fehler kann nach dem Einschalten eines Sensors auftreten, wenn Messwerte abgefragt werden, bevor diese vorliegen. Die einzelnen Sensoren benötigen meist einige Sekunden, bis die erste Messung verfügbar ist. Es kann auch sein, dass kein Sensorkopf auf die Sensorelektronik aufgesteckt wurde.
SENS_RH_NOT_MEASURED	-7	Der Feuchtemesswert wurde (noch) nicht ermittelt oder steht aus einem anderen Grund nicht zur Verfügung.
SENS_TEMP_NOT_MEASURED	-8	Der Temperaturmesswert wurde (noch) nicht ermittelt oder steht aus einem anderen Grund nicht zur Verfügung.
SENS_INVALID_MEASUREMENT	-9	Die Messwerte sind zur Zeit ungültig, z.B. liefert der Sensorkopf keine Messwerte.
SENS_INVALID_FUNCTION	-10	Die Funktion ist unzulässig. Der Fehler entsteht z.B. dann, wenn versucht wird, bei einem älteren Sensor ein nicht vorhandenes Heizelement zu aktivieren.

Funktion „SensFindDevice()“:

Prototyp:

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

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

Beschreibung:

Funktion durchsucht die aktuelle Geräteliste nach dem n-ten Gerät und füllt den Parameterblock mit dessen Parametern. Wenn das Gerät gefunden wurde, dann wird SENS_SUCCESS geliefert, sonst SENS_NOT_FOUND. Für die Suche kann ein Filterstring angegeben werden, der eine Auswahl nach bestimmten Gerätetypen erlaubt. Nicht erforderliche Parameter müssen als NULL übergeben werden. Um alle an den lokalen PC angeschlossenen Geräte aufzulisten, kann die Funktion in einer Schleife mit steigendem n solange aufgerufen werden, wie SENS_SUCCESS geliefert wird.

Parameter:	Typ:	Beschreibung:
n	LONG_PTR	Gibt den Index (n) des gewünschten Gerätes vor. Es wird bei 0 begonnen. Die Funktion liefert die Daten des n-ten Gerätes im USB zurück, welches den geforderten Parametern entspricht. Dabei werden alle mit dem PC verbundenen Sensoren berücksichtigt und der Reihe nach durchsucht.
pszMask **pwszMask**	LPSTR (ASCII) LPWSTR (Unicode)	Adresse eines Filterstrings oder NULL. Wenn eine Stringadresse angegeben wird, dann werden nur Geräte berücksichtigt, in deren Typ-Bezeichnung dieser String an beliebiger Stelle enthalten ist.
pDevice	PSENSDEVICE PSENSDEVICEA PSENSDEVICEW	Zeiger auf einen Puffer mit der Struktur „SENSDEVICE“ oder NULL. Wird eine Pufferadresse angegeben, dann wird der Puffer bei erfolgreicher Suche mit den Daten des gesuchten Gerätes gefüllt.

Return Wert:

Typ LRESULT, die Funktion liefert SENS_SUCCESS, wenn das gewünschte Gerät gefunden wurde, sonst SENS_NOT_FOUND.

Der Puffer für die Gerätedaten ist wie folgt definiert:

```
typedef struct _SENSDEVICEA                    Bus-Verzeichnis Eintrag, ASCII Code Version
{
    TCHAR       szTypeName[32];                Gerätetype-Bezeichnung
    TCHAR       szSerialNo[32];                Seriennummer des Gerätes
    LONG_PTR    nIndex;                        Geräteindex

} SENSDEVICEA, * PSENSDEVICEA;

typedef struct _SENSDEVICEW                    Bus-Verzeichnis Eintrag, Unicode Version
{
    WCHAR       szTypeName[32];                Gerätetype-Bezeichnung
    WCHAR       szSerialNo[32];                Seriennummer des Gerätes
    LONG_PTR    nIndex;                        Geräteindex

} SENSDEVICEW, * PSENSDEVICEW;
```

Hinweise: Diese Funktion liefert alle wichtigen Informationen über die derzeit angeschlossenen Geräte. Durch multiplen Aufruf kann z.B. eine Listbox als Geräteliste aufgebaut werden. Die Funktion greift dabei auf die DLL interne Liste der Geräte zu, wobei alle Geräte an allen USB Schnittstellen der Reihe nach indiziert werden. Es werden maximal 50 Geräte gleichzeitig unterstützt. Die COM Port Nummer der virtuellen COM Ports muss zwischen 1 und 999 liegen.

Funktion „ SensReadValues()“:

Prototyp:

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);

Beschreibung:

Funktion zur Abfrage aktueller Messwerte eines Gerätes. Es werden abhängig vom abgefragten Gerät bis zu 3 Messwerte geliefert, z.B. beim UFT75-AT Sensor relative Feuchte, Temperatur und Taupunkttemperatur. Messwerte für relative Feuchte liegen immer zwischen 0 und 100%. Messwerte für Temperaturen zwischen –40 und +120 °C.

Parameter:	Typ:	Beschreibung:
Parameter	PVOID	Parameter ist abhängig von „bMode“. Wird als Modus SENS_READ_BY_SERIAL_NUMBER angegeben, so ist „Parameter“ ein Zeiger auf einen nullterminierten String (ASCII oder Unicode), der die vollständige Seriennummer des Sensors enthält. Ist „bMode“ SENS_READ_BY_INDEX, so wird als Parameter der Geräteindex n angegeben, der bei der Suchfunktion SensFindDevice(n, ...) benutzt wurde, um das Gerät zu identifizieren.
bMode	BOOL	Adressierungsmodus für Sensorauswahl. Wie zuvor beschrieben, kann der Sensor entweder über die Seriennummer der den Geräteindex identifiziert werden.
pfValRH	float *	Zeiger auf eine Variable vom Typ float, die den Messwert der relativen Feuchte aufnehmen soll (4-Byte Fließkomma). Der Feuchtemesswert kann nur zwischen 0.0 und 100.0 % liegen. Wird der Pointer nicht angegeben (NULL), so entfällt die Zuweisung dieses Wertes. Wird auf einen Sensor zugegriffen, der keine Feuchte misst, dann wird 0.0% geliefert.
pfValTemp	float *	Zeiger auf eine Variable vom Typ float, die den aktuellen Temperaturmesswert aufnehmen soll (4-Byte Fließkomma). Der Temperaturmesswert kann nur zwischen -40.0 und +120.0 °C liegen. Wird der Pointer nicht angegeben (NULL), so entfällt die Zuweisung dieses Wertes. Wird auf einen Sensor zugegriffen, der keine Temperatur misst, dann wird -40.0°C zugewiesen.
pfValDew	float *	Zeiger auf eine Variable vom Typ float, welche die berechnete Taupunkttemperatur aufnehmen soll (4-Byte Fließkomma). Der Taupunkt wird im PC aus den Messwerten der relativen Feuchte und der Temperatur berechnet. Deshalb kann dieser Wert nur dann verfügbar sein, wenn beide anderen Werte ebenfalls verfügbar sind. Der Wert kann nur zwischen -40.0 und +120.0 °C liegen. Wird der Pointer nicht angegeben (NULL), so entfällt die Zuweisung dieses Wertes. Kann der Wert nicht berechnet werden, so wird -40.0°C geliefert.

Return Wert:

Typ **LRESULT**, die Funktion liefert einen Fehlercode vom Typ „SENS_xxx“ als Funktionsergebnis zurück.

Adressierungsmodus:	Wert:	Funktion:
SENS_READ_BY_-SERIAL_NUMBER	0	Der Funktionsparameter „Parameter” enthält einen Zeiger auf einen nullterminierten String, der die Seriennummer des Sensors enthält.
SENS_READ_BY_INDEX	1	Der Funktionsparameter “Parameter” enthält den Geräteindex, der auch verwendet wurde, um mit der Funktion „SensFindDevice()“ die Geräteparameter zu ermitteln. ACHTUNG: Der Index eines Gerätes kann sich verändern, wenn die USB Konfiguration sich ändert. Es wird daher immer empfehlen, den Sensor über seine Seriennummer auszuwählen.

Hinweise: Feuchtemesswerte, die nicht verfügbar sind, werden als 0.0% geliefert. Temperaturmesswerte, die nicht verfügbar sind werden als –40.0 °Celsius geliefert. Die Berechnung der Taupunkttemperatur kann nur dann erfolgen, wenn sowohl Feuchte als auch Temperatur vom Sensor gemessen wurden. Die Sensoren benötigen meist einige Sekunden nach dem Einschalten oder dem Wechseln des Sensorkopfes, bis die entsprechenden Werte verfügbar sind.

Funktion „SensSetHeating()“:

Prototyp:

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

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

Beschreibung:

Funktion zur Aktivierung oder Deaktivierung des Heizelementes eines UFT75 Sensorgerätes, welches diese Funktion unterstützt.

Parameter:	Typ:	Beschreibung:
Parameter	PVOID	Parameter ist abhängig von „bMode“. Wird als Modus SENS_READ_BY_SERIAL_NUMBER angegeben, so ist „Parameter“ ein Zeiger auf einen nullterminierten String (ASCII oder Unicode), der die vollständige Seriennummer des Sensors enthält. Ist „bMode“ SENS_READ_BY_INDEX, so wird als Parameter der Geräteindex n angegeben, der bei der Suchfunktion SensFindDevice(n, ...) benutzt wurde, um das Gerät zu identifizieren.
bMode	BOOL	Adressierungsmodus für Sensorauswahl. Wie zuvor beschrieben, kann der Sensor entweder über die Seriennummer der den Geräteindex identifiziert werden.
bEnable	BOOL	Flag gibt an, ob das Heizelement aktiviert (TRUE) oder deaktiviert (FALSE) werden soll.

Return Wert:

Typ **LRESULT**, die Funktion liefert einen Fehlercode vom Typ „SENS_xxx“ als Funktionsergebnis zurück.

Mogliche Adressierungsmodi:

Adressierungsmodus:	Wert:	Funktion:
SENS_READ_BY_SERIAL_NUMBER	0	Der Funktionsparameter „Parameter” enthält einen Zeiger auf einen nullterminierten String, der die Seriennummer des Sensors enthält.
SENS_READ_BY_INDEX	1	Der Funktionsparameter “Parameter” enthält den Geräteindex, der auch verwendet wurde, um mit der Funktion „SensFindDevice()“ die Geräteparameter zu ermitteln. ACHTUNG: Der Index eines Gerätes kann sich verändern, wenn die USB Konfiguration sich ändert. Es wird daher immer empfehlen, den Sensor über seine Seriennummer auszuwählen.

Hinweise: Nur UFT75 Sensorgeräte ab Firmware Version 2.0.00 verfügen über die Fähigkeit, das Heizelement zu aktivieren. Wird die Funktion auf ein Gerät angewendet, welches nicht über ein Heizelement verfügt, dann wird der Status „SENS_INVALID_FUNCTION“ geliefert. Wurde die Heizung erfolgreich eingeschaltet, dann liefert die Messwertabfragefunktion „SensReadValues()“ statt „SENS_SUCCESS“ den Status „SENS_HEATING_ENABLED“. Die Messwerte sind dann zwar korrekt erfasst worden, jedoch beeinflusst die Heizung die Messwerte erheblich. Bereits eine Temperaturerhöhung um ein Grad Celsius kann eine Abweichung von mehrere Prozent bei der relativen Feuchte verursachen.

Funktion „ SensGetChangeFlag()“:

Prototyp:

BOOL CALLBACK SensGetChangeFlagA(VOID);

BOOL CALLBACK SensGetChangeFlagW(VOID);

Beschreibung:

Funktion liefert TRUE, wenn sich die Sensorkonfiguration seit dem letzten Aufruf verändert hat (neue Geräte oder Geräte entfernt), bzw. FLASE wenn keine Veränderung stattfand.

Parameter:	Typ:	Beschreibung:
---	VOID	Die Funktion hat keine Parameter.

Return Wert:

Typ **BOOL**, die Funktion liefert TRUE, wenn sich die Gerätekonfiguration seit dem letzten Aufruf verändert hat, und FALSE wenn nicht. Die ASCII Code und die Unicode Versionen sind identisch, jedoch aus kompatibilitätsgründen sind beide Varianten implementiert.

Hinweise: Falls TRUE geliefert wird, dann sollte die neue Gerätekonfiguration unbedingt mit der Funktion SensFindDevice() neu ermittelt werden, da Geräte entfernt worden sein könnten bzw. neue Sensoren hinzugefügt wurden. Wenn das Betriebssystem eine Änderung der USB Konfiguration meldet, dann veranlasst die DLL automatisch die Suche nach neuen Sensorgeräten. Diese wird multithreaded ausgeführt und muss mit der Applikation synchronisiert werden um eine aktuelle Sensorliste zu erhalten. Die Sensorsuche ist normalerweise nach 150 bis 500 Millisekunden beendet. Die DLL unterstützt maximal 999 Sensorgeräte an den COM Ports 1 bis 999. Sensoren, auf die während der Gerätesuche von anderer Seite aus zugergriffen wird, werden möglicherweise nicht erkannt.

Funktion „SensWaitReady ()“:

Prototyp:

BOOL CALLBACK SensWaitReadyA(LONG nTimeout);

BOOL CALLBACK SensWaitReadyW(LONG nTimeout);

Beschreibung:

Die Funktion wartet auf den Abschluss der Gerätesuche maximal für die angegebene Länge in Millisekunden und liefert TRUE, wenn die Suche nach Sensorgeräten innerhalb der angegebenen Zeit beendet wurde. Die Applikation kann damit mit den Threads der DLL synchronisiert werden.

Parameter:	Typ:	Beschreibung:
nTimeout	BOOL	Timout in Millisekunden.

Return Wert:

Typ **BOOL**, die Funktion liefert TRUE, wenn die Suche nach Sensorgeräten innerhalb der angegebenen Zeit beendet wurde oder nicht mehr läuft. Es wird FALSE geliefert, wenn die Gerätesuche im angegebenen Zeitraum noch nicht abgeschlossen werden konnte oder ein anderer Fehler auftrat.

Hinweise: Aufgrund der vollständig als multi-threaded ausgelegten Implementierung der DLL Funktionen kann die Anwendung bereits die Geräteliste abfragen, während die Gerätesuche im Hintergrund noch läuft. Dies kann Nach dem Start der Applikation problematisch sein, da die DLL die Ausführung der Applikation nicht blockiert. Somit sind meist noch keine Sensorgeräte erkannt worden, wenn die Applikation direkt nach dem Start bereits die Sensorliste abfragt. Die Funktion "SensWaitReady()" sollte dazu verwendet werden, mindestens die erste Gerätesuche mit der Applikation zu synchronisieren. Die multi-threaded Implementierung hat jedoch große Vorteile, da in keinem Fall mehr ein Anwendungs-Thread durch eine DLL Funktion blockiert werden kann. Außerdem dauert die Gerätesuche unabhängig von der Anzahl der angeschlossenen Sensoren fast immer gleich lange, da für jede Schnittstelle ein eigener Kommunikations-Thread verwendet wird. Üblicherweise sollten alle Sensoren nach ca. 150 Millisekunden erkannt worden sein, spätestens jedoch nach 500 Millisekunden.

Funktion „ DllGetVersion ()“:

Prototyp:

HRESULT WINAPI DllGetVersion(DLLVERSIONINFO * pDllVerInfo);

Beschreibung:

Funktion füllt eine DLLVERSIONINFO Struktur mit den Versionsdaten der DLL, entsprechend den Empfehlungen von Microsoft für Windows DLL's.

Parameter:	Typ:	Beschreibung:
pDllVerInfo	DLLVERSIONINFO *	Zeiger auf einen Datenpuffer mit der Struktur DLLVERSIONINFO (siehe "Shlwapi.h".

Return Wert:

Typ **HRESULT**, die Funktion liefert S_OK oder einen Fehlercode (E_FAIL, E_...), siehe "WinError.h".

Der Puffer für die Versionsdaten ist im Headerfile "Shlwapi.h" wie folgt definiert:

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

} DLLVERSIONINFO;
```

Beispiel-Code:

Der folgende Code demonstriert in Standard „C“, wie verfügbare UFT75-AT Sensorgeräte gesucht und in eine Listbox eingetragen werden. Mit einem Timer werden die Messdaten des vom Anwender ausgewählten Sensors in regelmäßigen Intervallen gelesen und angezeigt. Der Code entspricht im Wesentlichen der Funktion des Beispielprogramms „UFTAccessTest.exe“ (vereinfacht und Ausschnittsweise dargestellt). Datentypen und DLL-Aufrufe sind rot markiert. Natürlich wird eine Dialog-Ressource mit den verwendeten Feldern benötigt, die hier nicht extra ausgeführt wird.

1. Beispiel zum erstellen einer Listbox Auswahl, welche die verfügbaren UFT75 Sensorgeräte auflistet (vereinfacht):

```
#include ... Windows Includes einfügen
#include "UFTAccess.h"


VOID      SetupMyListBox(HWND hWnd)
{
 SENSDEVICE      DeviceInfo;              /* Gerätedatenpuffer      */
 HWND            hItem;                   /* ListBox Handle         */
 LONG            i;                       /* Zähler                 */
 LONG            Index;                   /* ListBox Auswahl        */

if(!SensWaitReady(500)) return;           /* auf Bereistschaft warten   */

 hItem = GetDlgItem(hWnd,                 /* ermittle ListBox Handle    */
                    IDC_SENSORLIST);

 SendMessage(hItem,                       /* init Liste             */
             LB_RESETCONTENT, 0, 0L);

 ZeroMemory(&DeviceInfo,                  /* init Puffer            */
            sizeof(SENSDEVICE));

 i = 0;

 while(SensFindDevice(i, 0L, &DeviceInfo) == SENS_SUCCESS)
 {
  sprintf(szBuffer, "%s  -  %s",          /* Eintrag formatieren    */
          DeviceInfo.szSerialNo, DeviceInfo.szDeviceID);

  Index = SendMessage(hItem,              /* Element hinzufügen     */
                      LB_ADDSTRING, 0, (LPARAM)szBuffer);

  if(Index != LB_ERR)                     /* wenn kein Fehler auftrat   */
  {
   SendMessage(hItem,                     /* Element hinzufügen     */
               LB_SETITEMDATA, (WPARAM)Index, (LPARAM)i);
  }

  i++; if(i > 999) break;                 /* nächstes Gerät, max. 1000 */
 }

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

 SetDlgItemText(hWnd, IDC_STATUS, "Noch keine Daten gelesen!");
}
```

2. Beispiel (Ausschnitt) aus einer Dialog-Callback Funktion. Die Messwerte des in der Listbox gewählten Sensors werden über Timer (1) ausgelesen und im Dialog angezeigt:

```
switch(Message)                                 /* je nach Message Code      */
{
 case WM_TIMER:                                 /* Timer-Message             */
  {
   switch(wParam)                               /* je nach Timerkennung      */
   {
    case 0x0001:                                /* neue Messung lesen        */
     {
      LRESULT           Status;                 /* Statuscode                */
      BYTE              szBuffer[256];          /* Arbeitspuffer             */
      HWND              hItem;                  /* Item Handle               */
      LONG              Index;                  /* Listenauswahl             */
      LPSTR             p;                      /* Hilfspointer              */
      float             fValRH;                 /* Feuchtemessung            */
      float             fValTemp;               /* Temperaturmessung         */
      float             fValDew;                /* Taupunktberechnung        */

      hItem = GetDlgItem(hWnd,                  /* ermittle ListBox Handle   */
                         IDC_SENSORLIST);

      Index = SendMessage(hItem,                /* ermittle Auswahl          */
                          LB_GETCURSEL, 0, 0L);

      if(Index == LB_ERR) return(FALSE);        /* wenn kein Sensor gewählt  */

      ZeroMemory(szBuffer, 256);                /* init Puffer               */

      SendMessage(hItem,                        /* lese Eintrag aus Liste    */
                  LB_GETTEXT, (WPARAM)Index, (LPARAM)szBuffer);

      p = strstr(szBuffer, "  -  ");            /* Seriennummer abtrennen    */
      if(p) *p = 0;

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

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

       sprintf(szBuffer, "%5.1f %%RH", fValRH);
       SetDlgItemText(hWnd, IDC_VAL_RH, szBuffer);

       sprintf(szBuffer, "%+5.1f °C",  fValDew);
       SetDlgItemText(hWnd, IDC_VAL_DEW, szBuffer);

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

   default:
    break;
  }
 }
 return(FALSE);

... andere Messages auswerten
```