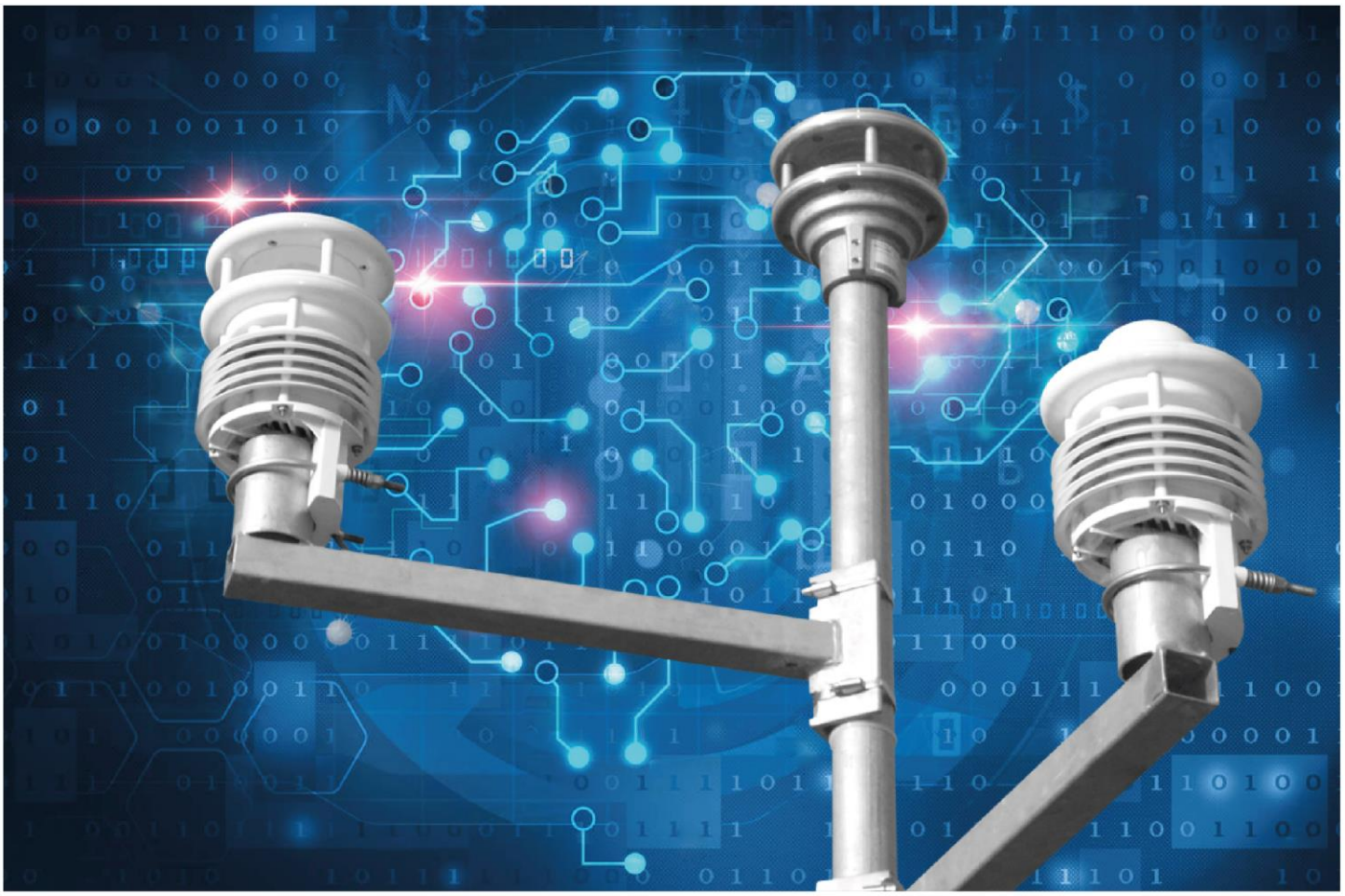
Lufft
an OTT HydroMet brand



# UMB Software Bibliothek

Referenzhandbuch

OTT HydroMet

# Inhalt



## Änderungshistorie

| Version | Datum | Änderungen |
|---|---|---|
| **V0.1** | 12.03.2021 | Erste Version |
| **V0.2** | 22.03.2021 | Screenshots angepasst<br>Erläuterungen zur UMB Spezifikation |
| **V0.3** | 19.04.2021 | 64-bit Versionen der Bibliothek |
| **V0.4** | 06.05.2021 | 64-bit Version für ARM |
| **V0.5** | 20.05.2021 | Tabelle mit unterstützten UMB Befehlen |

# 1 Das UMB Protokoll

Das UMB-Protokoll ist ein von der Firma Lufft spezifiziertes, offengelegtes Binärprotokoll zur Konfiguration und Datenabfrage von Messgeräten.

Die aktuelle Version der Spezifikation findet sich im Download-Bereich der Homepage www.Lufft.de. Das Dokument enthält alle Informationen zum Frame-Aufbau und zeitlichem Ablauf sowie eine detaillierte Beschreibung aller Befehle.

# 2 Die UMB Bibliothek

Die Bibliothek ist in der Sprache C geschrieben und für Windows sowie für Linux verfügbar.

Sie verwendet keine dynamische Speicherallokation.

In der Bibliothek sind die in Tabelle 1 aufgeführten Befehle des UMB Protokolls implementiert.

| <cmd> | Beschreibung | Bibliothek V0.4 |
|---|---|---|
| 20h | Hard- und Softwareversion | |
| 21h | EEPROM auslesen | ● |
| 22h | EEPROM programmieren | ● |
| 23h | Onlinedatenabfrage | |
| 24h | Offlinedatenabfrage | |
| 25h | Reset / Default | ● |
| 26h | Statusabfrage | ● |
| 27h | Uhrzeit / Datum setzen | |
| 28h | Uhrzeit / Datum auslesen | |
| 29h | Testbefehl | |
| 2Ah | Monitor | |
| 2Bh | Protokollwechsel | |
| 2Ch | letzte Fehlermeldung | |
| 2Dh | Geräteinformation | (●) |
| 2Eh | Reset mit Verzögerung | |

| <cmd> | Beschreibung | Bibliothek V0.4 |
|---|---|---|
| 2Fh | Onlinedatenabfrage mehrere Kanäle | ● |
| | | |
| 30h | neue Geräte-ID dauerhaft setzen (verc 1.0) | |
| 30h | neue Geräte-ID temporär setzen (verc 1.1) | |
| 36h | UMB-Tunnel | |
| 37h | Firmware übertragen | ● |
| 38h | Binärdaten übertragen | |
| | | |
| 40h – 7Fh | reserviert für gerätespezifische Kommandos | |
| 80h – 8Fh | reserviert für Entwicklung | |
| | | |
| F0h | EEPROM programmieren mit PIN | |

Tabelle 1 Befehle des UMB Protokolls, die von der Bibliothek unterstützt werden

Mit dem Befehl ‚Geräteinformation' (2Dh) kann eine Vielzahl von Geräte-Eigenschaften abgefragt werden. Davon werden bislang die in Tabelle 2 angegebenen Sub-Befehle unterstützt.

| <info> | Beschreibung | Bibliothek V0.4 |
|---|---|---|
| 10h | Gerätebezeichnung | ● |
| 11h | Gerätebeschreibung | |
| 12h | Hard- und Softwareversion | |
| 13h | erweiterte Versions-Info | |
| 14h | Größe des EEPROM | |
| 15h | Anzahl verfügbare Kanäle | ● |
| 16h | Nummern der Kanäle | ● |
| 17h | Anzahl der gerätespezifischen Versionsinformation-Slots auslesen | |
| 18h | Gerätespezifische Versionsinformationen auslesen | |

| <info> | Beschreibung | Bibliothek V0.4 |
|---|---|---|
| 20h | Messgröße des Kanals | |
| 21h | Messbereich des Kanals | |
| 22h | Messeinheit des Kanals | |
| 23h | Datentyp des Kanals | |
| 24h | Messwerttyp | |
| | | |
| 30h | komplette Kanalinfo | ● |
| 40h | Anzahl der IP-Interfaces | |
| 41h | IP-Information | |

Tabelle 2 Sub-Befehle des Befehls ‚Geräteinformation', die von der Bibliothek unterstützt werden

# 3 Lieferumfang

Der Ordner „**lufft**" enthält alle Dateien, die zur Verwendung der UMB Bibliothek benötigt werden:

- Die Software-Bibliotheken für Windows bzw. Linux / Linux auf ARM:

| | windows | linux | Linux / ARM |
|---|---|---|---|
| **64 bit** | UmbControllerLib.lib | libUmbController.a | libUmbControllerArm_64.a |
| **32 bit** | UmbControllerLib_32.lib | libUmbController_32.a | libUmbControllerArm_32.a |

- Die Header-Dateien zur Verwendung der Bibliothek:
  **UmbControllerLib.h**: Schnittstelle der Bibliothek
  **Umb_Types.h**: Allgemeine Typdefinitionen

Im Ordner „**src**" findet man Dateien mit Beispielen für die Anbindung der Bibliothek an das eigene System:

- **UmbCtrlTest.cpp**: Testprogramm zur Veranschaulichung der Funktionsweise
- **ComWin.c/.h**: Beispiel-Implementierung zur Anbindung unter Windows
- **ComLinux.c/.h**: Beispiel-Implementierung zur Anbindung unter Linux

Der Ordner „**win**" enthält nicht-Lufft-eigene Dateien, die im Testprogramm bzw. in den Beispiel-Implementierungen unter Windows verwendet werden. Hier sind die in den jeweiligen Quelldateien festgelegten Nutzungsbedingungen zu beachten.

Im Ordner „**examples**" befindet sich ein Beispiel für die Installation der Bibliothek auf einem RaspberryPi. Weitere Beispiele sind geplant.

# 4 Inbetriebnahme

Für die Verwendung der UMB Bibliothek müssen die beiden Header-Dateien Umb_Types.h und UmbControllerLib.h in das eigene Projekt kopiert werden.

Abhängig vom verwendeten System (Windows, Linux, Linux auf ARM) wird die zugehörige Bibliothek benötigt, siehe auch Kapitel 3.

Die Installationsanleitung für einen RaspberryPi ist separat in der Datei README.txt im Verzeichnis /examples/RaspberryPi nachzulesen.

# 5 Verwendung

Der jeweils aktuelle Funktionsumfang der Bibliothek ist der Schnittstellendatei UmbControllerLib.h zu entnehmen.

## 5.1 System-Anbindung

Die Ansteuerung der seriellen Schnittstelle erfolgt über Funktionszeiger, die in der Struktur UMB_CTRL_COM_FUNCTION_T definiert sind, siehe Abbildung 1.

```
//! callback functions for communication
typedef struct
{
    void* pUserHandle;

    Std_ReturnType(*pfnInit)    (void* pUserHandle);
    Std_ReturnType(*pfnDeinit)  (void* pUserHandle);
    Std_ReturnType(*pfnUse)     (void* pUserHandle);
    Std_ReturnType(*pfnUnuse)   (void* pUserHandle);

    Std_ReturnType(*pfnTx)      (void* pUserHandle, const uint32 length, const uint8* const pBytes);

    Std_ReturnType(*pfnRxAvail) (void* pUserHandle, uint32* const pAvail);
    Std_ReturnType(*pfnRx)      (void* pUserHandle, const sint32 timeoutMs, const uint32 maxLen,
                                                    uint32* const pLength, uint8* const pBytes);
    Std_ReturnType(*pfnRxClearBuf)(void* pUserHandle);

} UMB_CTRL_COM_FUNCTION_T;
```

Abbildung 1 Struktur mit Funktionszeigern zur Ansteuerung der seriellen Schnittstelle

Die Funktionszeiger (*pfnInit) und (*pfnDeinit) sind optional und können z. B. verwendet werden, um die serielle Schnittstelle zu öffnen bzw. zu schließen. Wird dies aber bereits an anderer Stelle getan, können die beiden Funktionszeiger auch auf NULL gesetzt werden.

Alle anderen Funktionszeiger sind obligatorisch und müssen implementiert werden.

Die Funktionszeiger (*pfnUse) und (*pfnUnuse) sind für den Schutz von Variablen oder Programmsegmenten durch Semaphoren vorgesehen. In den Beispiel-Implementationen führen diese Funktionen derzeit keinen aktiven Code aus.

Das Handle *pUserHandle kann benutzt werden, um anwenderspezifische Daten an die Callback-Funktionen durchzureichen. So werden in den Beispiel-Implementierungen comWin.cpp und ComLinux.cpp alle Daten, die im laufenden Betrieb benötigt werden, in einer Struktur COM_HANDLE_T zusammengefasst. *pUserHandle zeigt auf die Adresse eines solchen Datensatzes, wodurch diese Daten dann in den Callback-Funktionen verfügbar sind. Abbildung 2 zeigt die Initialisierung eines *pUserHandle, Abbildung 3 die spätere Anwendung.

```
UMB_CTRL_COM_FUNCTION_T* pComFunction = (UMB_CTRL_COM_FUNCTION_T*)malloc(sizeof(UMB_CTRL_COM_FUNCTION_T));

if (pComFunction)
{
    pComFunction->pUserHandle = malloc(sizeof(COM_HANDLE_T));

    if (pComFunction->pUserHandle)
    {
        COM_HANDLE_T* pComHandle = (COM_HANDLE_T*)pComFunction->pUserHandle;

        pComHandle->config = *pConfig;
        memset(&pComHandle->port, 0, sizeof(pComHandle->port));
    }
```

Abbildung 2 Initialisierung eines *pUserHandle

```
static Std_ReturnType ComInit(void* pUserHandle)
{
    COM_HANDLE_T* pComHandle = (COM_HANDLE_T*)pUserHandle;

    try
    {
        int number = std::strtoul(pComHandle->config.serialIf, NULL, 10);
        pComHandle->port.Open(number, pComHandle->config.baudrate, CSerialPort::NoParity, 8,
                              CSerialPort::OneStopBit, CSerialPort::NoFlowControl);
    }
    catch (CSerialException& e)
    {
        printf("Unexpected CSerialPort exception, Error:%u\n", e.m_dwError);
        return E_NOT_OK;
    }

    return E_OK;
}
```

Abbildung 3 Verwendung eines *pUserHandle

Die Module ComLinux.cpp/.h sowie ComWin.cpp/.h zeigen beispielhaft, wie die Zuweisung dieser Funktionszeiger umgesetzt werden kann:

In ComLinux ist die Ansteuerung der seriellen Schnittstelle direkt implementiert, ComWin dagegen verwendet Fremdsoftware (SerialPort.h), für die nur noch die zur UMB Bibliothek kompatiblen Wrapper-Funktionen bereitgestellt werden, siehe auch Abbildung 4.

```
160  static Std_ReturnType ComTx(void* pUserHandle, const uint32 length, const uint8* const bytes)
161  {
162      COM_HANDLE_T* pComHandle = (COM_HANDLE_T*)pUserHandle;
163
164      if (write(pComHandle->m_fdTTY, bytes, length) > 0)
165      {
166          return E_OK;
167      }
168      return E_NOT_OK;
169  }
170
171
172  static Std_ReturnType ComRx(void* pUserHandle, const sint32 timeoutMs, const uint32 maxLen,
173      uint32* const pLength, uint8* const pBytes)
174  {
175      COM_HANDLE_T* pComHandle = (COM_HANDLE_T*)pUserHandle;
176      int retval;
177      fd_set set;
178      struct timeval timeout;
179
180      if((pComHandle->m_fdTTY < 0) || (pLength == nullptr) || (pBytes == nullptr))
181      {
182          return E_NOT_OK;
183      }
184      FD_ZERO(&set);
185      FD_SET(pComHandle->m_fdTTY, &set);
186      timeout.tv_sec = timeoutMs / 1000;
187      timeout.tv_usec = (timeoutMs % 1000) * 1000;
188      retval = select(pComHandle->m_fdTTY + 1, &set, NULL, NULL, &timeout);
189      if(retval > 0)
190      {
191          retval = read(pComHandle->m_fdTTY, pBytes, maxLen);
192          if(retval > 0)
193          {
194              *pLength = retval;
195              return E_OK;
196          }
197      }
198      return E_NOT_OK;
199  }
```

```
119  static Std_ReturnType ComTx(void* pUserHandle, const uint32 length, const uint8* const bytes)
120  {
121      COM_HANDLE_T* pComHandle = (COM_HANDLE_T*)pUserHandle;
122
123      pComHandle->port.Write(bytes, length);
124
125      return E_OK;
126  }
127
128
129
130
131  static Std_ReturnType ComRx(void* pUserHandle, const sint32 timeoutMs, const uint32 maxLen,
132      uint32* const pLength, uint8* const pBytes)
133  {
134      COM_HANDLE_T* pComHandle = (COM_HANDLE_T*)pUserHandle;
135      COMMTIMEOUTS timeouts;
136
137      pComHandle->port.GetTimeouts(timeouts);
138      timeouts.ReadIntervalTimeout = MAXDWORD;
139      timeouts.ReadTotalTimeoutMultiplier = MAXDWORD;
140      timeouts.ReadTotalTimeoutConstant = timeoutMs;
141      pComHandle->port.SetTimeouts(timeouts);
142
143      *pLength = pComHandle->port.Read(pBytes, maxLen);
144
145      return E_OK;
146  }
147
148
149
150
151
152
153
154
155
156
157
158
```

Abbildung 4 Implementierungsbeispiele zur Ansteuerung der seriellen Schnittstelle:
links: Beispiel für Linux, manuelle Implementierung
rechts: Beispiel für Windows, Verwendung bereits existierender Implementierung

## 5.2 Initialisierung

Die Initialisierung der UMB Bibliothek umfasst 3 Punkte:

- Zuweisung der Funktionszeiger zur Ansteuerung der seriellen Schnittstelle
  Der Übersicht halber erfolgt die Zuweisung der benötigten Funktionszeiger am besten in einer eigenen, vom Anwender definierten Funktion, siehe hierzu Abschnitt 5.1.

- Bereitstellung des Handles
  Die UMB Bibliothek verwendet keine dynamische Speicherallokation. Daher muss der Anwender den Speicher für die verwendeten Bibliotheks-Instanzen bereitstellen.
  Dieses Handle wird beim Aufruf aller weiteren Funktionen der UMB Bibliothek benötigt.

- Aufruf der Initialisierungsfunktion der Bibliothek
  Der Initialisierungsfunktion UmbCtrl_Init() müssen das Handle sowie die Variable, die die Funktionszeiger enthält, übergeben werden.

Abbildung 5 zeigt exemplarisch die Initialisierungssequenz, Abbildung 6 die Abfrage von Geräte-Namen und -Status.

```
int main(int argc, char* argv[])
{
    UMB_CTRL_STATUS_T status;
    UMB_CTRL_T *pUmbCtrl;

    // UMB lib version
    UMB_CTRL_VERSION_T version = UmbCtrl_GetVersion();
    printf("UMB Lib Version: major=%d, minor=%d\n", version.major, version.minor);

    // Initialization
    // TODO: Adjust to used serial interface
    char serialIf[] = { "1" };
    COM_CONFIG_T comConfig;
    UMB_CTRL_COM_FUNCTION_T * pUmbCtrlComFunction;

    // TODO: Adjust to used baudrate
    comConfig.baudrate = 19200;
    comConfig.serialIf = serialIf;
    pUmbCtrlComFunction = ComFunctionInit(&comConfig);

    pUmbCtrl = malloc(UmbCtrl_GetHandleSize());
    status = UmbCtrl_Init(pUmbCtrl, pUmbCtrlComFunction, 0);
```

Abbildung 5 Initialisierung der UMB-Bibliothek

```
// Further processing
UMB_ADDRESS_T umbAddress;
// TODO: Adjust to used class id / device id
umbAddress.deviceId = 0x01; // device id: 1
umbAddress.classId = 0x70;  // class id: 7 = weather station

uint8 name[41] = { 0 };
status = UmbCtrl_GetDevName(pUmbCtrl, umbAddress, name);
if (status.global == UMB_CTRL_STATUS_OK)
{
    printf("Device name: %s\n", name);
}
else
{
    printf("ERROR [request device name]: lib=0x%0X dev=0x%0X\n",
        status.detail.library, status.detail.device);
}

ERROR_STATUS_T deviceStatus;
status = UmbCtrl_GetDevStatus(pUmbCtrl, umbAddress, &deviceStatus);
if (status.global == UMB_CTRL_STATUS_OK)
{
    printf("Device status: %d\n", deviceStatus);
}
else
{
    printf("ERROR [request device status]: lib=0x%0X dev=0x%0X\n",
        status.detail.library, status.detail.device);
}
```

Abbildung 6 Abfrage von Geräte-Name und Geräte-Status

## 5.3 Testprogramm

Das Testprogramm in UmbCtrlTest.cpp zeigt beispielhaft die Verwendung der UMB Controller Bibliothek. Vor Verwendung des Testprogramms müssen alle mit ‚TODO' gekennzeichneten Stellen im `main()`-Programm an das eigene Testsystem angepasst werden . Diese sind

- Präprozessor-Definition _USE_NCURSES, um die graphische Fortschritts-Anzeige bei der Update-Funktion unter Linux nutzen zu können (Näheres siehe unten):
  ```
  #define _USE_NCURSES
  ```

- Verwendete serielle Schnittstelle, z. B.
  ```
  char serialIf[] = { "3" };
  ```
  Hinweis:
  Unter Linux muss hier der gesamte Pfad der seriellen Schnittstelle angegeben werden, z. B.
  ```
  char serialIf[] = { "/dev/tty03" };
  ```

- Baudrate der seriellen Schnittstelle, z. B.
  ```
  comConfig.baudrate = 19200;
  ```

- UMB-Adresse des UMB-Gerätes, mit dem kommuniziert werden soll, z. B.
  ```
  umbAddress.deviceId = 0x01; // device id: 1
  umbAddress.classId = 0x70;  // class id: 7 = weather station
  ```

- Pfad und Name der Firmware Datei, z. B.
  ```
  char fileName[] = { "C:\\Projekte\\UmbController\\WS100_update.bin" };
  ```

Die auskommentierten Funktionen (siehe Abbildung 7) werden am besten einzeln und nach Bedarf in das Testprogramm übernommen, um mit der jeweiligen Funktionalität vertraut zu werden.

```
    //writeMemory(pUmbCtrl, umbAddress);
    //getChannelInfo(pUmbCtrl, umbAddress);
    //getChannelData(pUmbCtrl, umbAddress);
    //firmwareUpdate(pUmbCtrl, umbAddress);

    // De-Initialization
    UmbCtrl_Deinit(pUmbCtrl);
}
```

Abbildung 7 Beispielfunktionen zur Verwendung der UMB-Bibliothek

**Zur Präprozessor-Definition _USE_NCURSES**

Die Beispielimplementierung `firmwareUpdate()` verwendet eine grafische Darstellung des Update-Fortschritts, die unter Linux das Paket ncurses voraussetzt. Dieses muss z. B. auf einem RaspberryPi manuell installiert werden, da es nicht über raspbian-stretch-lite vorinstalliert ist.

Eine Anleitung hierfür findet sich in der Datei README.txt im Verzeichnis /examples/RaspberryPi.

Soll diese Fortschrittsanzeige genutzt werden, muss man bei installiertem Paket ncurses die Präprozessor-Definition _USE_NCURSES setzen. Ist diese Anweisung dagegen auskommentiert, wird statt der graphischen eine einfache Fortschrittsanzeige verwendet, die ohne weitere Pakete auskommt.

# 6 Hinweise zum Firmware-Update

Ältere UMB-Geräte wie WSx00, Ventus, Anacon etc. verwenden eine Update-Datei im .mot-Format. Diese können nicht über das UMB-Protokoll, sondern ausschließlich über Hexload auf ein Gerät übertragen werden.

Für die neue Generation der UMB-Geräte wie z. B. MARWIS, WS1000, WS100, SHM31 u. a. wurde daher das Datei-Format .bin definiert, das auch ein Firmware-Update über UMB ermöglicht.

➔ Firmware-Updates über das UMB-Protokoll sind nur für die UMB-Geräte möglich, deren Update-Datei im .bin-Format vorliegt.


Lufft
an OTT HydroMet brand
OTT
HydroMet