YASDI API-Dokumentation

Technische Beschreibung

YASDI-TDE084020 | Version 2.0

DE

Inhaltsverzeichnis

1 Erklärungen zu den verwendeten Symbolen

Um Ihnen einen optimalen Gebrauch dieses Handbuchs und einen sicheren Baugruppeneinsatz in den Phasen der Inbetriebnahme, des Betriebs und der Wartung zu gewährleisten, beachten Sie bitte die folgenden Erklärungen zu den verwendeten Symbolen.

Dieses Symbol kennzeichnet einen Sachverhalt, der wichtig für den optimalen Betrieb Ihres Produktes ist. Lesen Sie diese Abschnitte daher aufmerksam.

2 Einleitung

Dieses Dokument beschreibt den Aufbau und die Benutzung der Software "YASDI" zur Kommunikation mit SMA-Geräten. Der Name "YASDI" steht für "**Y**et **A**nother **S**MA **D**ata **I**mplementation".

Die Software implementiert die Kommunikation per "SMA-Data(1)-Protokoll" über die Transportprotokolle "Sunny-Net" und "SMA-Net" mit SMA-Geräten (u.a. Sunny Boy-Wechselrichter) als eine Softwarebibliothek ohne eine eigene grafische Oberfläche.

YASDI unterstützt die Abfrage von aktuellen Messwerten (Momentanwerten) sowie das Lesen und ggf. Setzen von Geräteparametern. Eine Abfrage von archivierten Messwerten von angeschlossenen Datenloggern (z. B. Sunny Boy Control oder Sunny Beam) sind nicht möglich. Derzeit werden folgende Kommunikationsmedias unterstützt:

- RS232/RS485/Powerline
- UDP/IP (Sunny Boy Control mit NET Piggy-Back)

Die Software wurde so ausgelegt, dass sie einfach an andere Umgebungen (Betriebssysteme) angepasst werden kann. Zur Zeit gibt es primär Portierungen für Windows, Linux und Apple Macintosh (Mac OS X). Alle systemabhängigen Funktionen werden über Softwareschichten vom Betriebssystem abstrahiert und sind dadurch einfach portierbar.

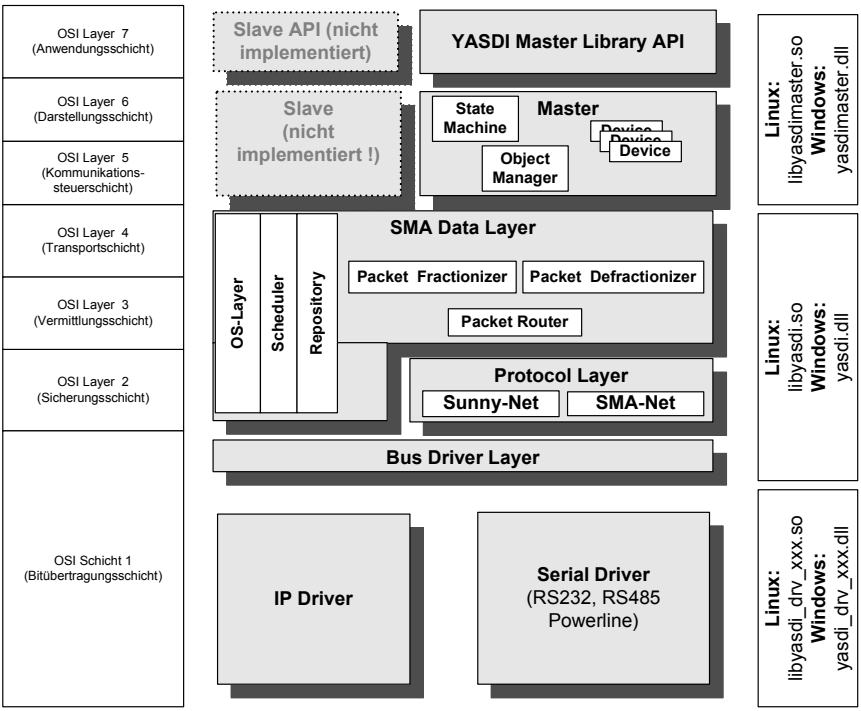
Die Software ist in der Programmiersprache "C" geschrieben und ermöglicht eine größtmögliche Portabilität auf mögliche andere Zielplattformen.

Die Implementierungen für Windows und Linux sind als dynamische Bibliotheken ausgeführt. Die Verwendung als statische Bibliothek im Zusammenhang eines monolithischen Programms ist ebenfalls möglich.

Aktuell ist YASDI als Quellcode erhältlich (Lizenz „LGPL“) sowie kompiliert (nur für Windows).

2.1 Software–Übersicht

Bei der Implementierung von YASDI wurde diese an das OSI-Schichtenmodell für Netzwerkprotokolle angelehnt. Die einzelnen Schichten sind in bestimmten Bibliotheken zusammengefasst, die aus der folgenden Übersicht entnommen werden können:



3 Verwendete Datentypen

In den Schnittstellenbeschreibungen werden Datentypen verwendet, die im folgenden aufgeführt sind (definiert in der Datei "smadef.h"):

Datentyp	Datentypbeschreibung
DWORD	32-bit unsigned
WORD	16-bit unsigned
BYTE	8-bit unsigned
BOOL	8-bit unsigned
DOUBLE	64-bit double Wert (8 Bytes)

4 Funktionsreferenz

Die Bibliothek „yasdimaster“ beinhaltet Funktionen zum Steuern des sogenannten Masters. Dieser ermöglicht das aktive Abfragen von Informationen oder Messwerten von SMA Geräten.

4.1 Funktionen zur Initialisierung

4.1.1 YasdiMasterInitialize

Initialisierung der YASDI-Master-Bibliothek. Die Funktion muss vor allen anderen Funktionen der Bibliothek genau einmal aufgerufen werden.

```
void yasdiMasterInitialize(char * cIniFileName,
                           DWORD * pDriverCount);
```

Parameter:

cIniFileName:

- Dateiname der Konfigurationsdatei (yasdi.ini) inklusive Pfad. Weiteres zur Konfigurationsdatei in Kapitel 4

pDriverCount:

- Verweist auf eine Variable, in der nach dem Aufruf die Anzahl der in YASDI zurzeit geladenen Schnittstellentreiber zurückgegeben werden.

4.1.2 YasdiMasterShutdown

Mit diesem Funktionsaufruf werden benutzter Speicher und Ressourcen der YASDI-Master-Library wieder freigegeben. Die Funktion sollte immer nach Benutzung von YASDI aufgerufen werden. Es sind keine Parameter zur Übergabe notwendig.

```
void yasdiMasterShutdown( void );
```

Parameter: - keine –

Return-Wert: -keiner-

4.1.3 YasdiReset

Die Funktion führt einen vollständigen Reset der Software durch. Alle aktuell evtl. erfassten Geräte werden entfernt. Die Software befindet sich dann in einem Zustand, wie nach dem Aufruf der Funktion "yasiMasterInitialize(...)".

```
void yasdiReset( void )
```

Parameter: - keine –

Return-Wert: -keiner-

4.1.4 YasdiMasterAddEventListener

Die Funktion fügt eine Callbackfunktion ein, mit der auf asynchrone Ereignisse von YASDI gewartet werden kann.

```
void yasdiMasterAddEventListener(
                                    void * vpEventCallback,
                                    BYTE bEventType);
```

Parameter:

vpEventCallback:

- Funktionszeiger zum Einhängen. Für jede Art Ereignisklasse kann ein eigener Funktionszeiger eingehängt werden.

BEventType:

- Der Eventtyp, der beobachtet werden soll. Folgende Typen sind derzeit möglich: YASDI_EVENT_DEVICE_DETECTION:

Übewachung von Geräteerfassungen. Es wird die Funktion mit folgendem Typ angesprungen:

```
typedef void (*TYasdiEventDeviceDetection)(
                                        BYTE subEvent,
                                        DWORD devHandle );
```

"subEvent" ist dabei einer der folgenden Events:

- YASDI_EVENT_DEVICE_ADDED:
  - Neues Gerät gefunden: Gerätehandle als „devHandle"
- YASDI_EVENT_DEVICE_REMOVED:
  - Gerät entfernt: Gerätehandle als „devHandle"
- YASDI_EVENT_DEVICE_SEARCH_END:
  - Erfassung beendet: (keine Parameter)

YASDI_EVENT_CHANNEL_NEW_VALUE:

- Überwachen von Kanalwertabfragen, die per GetChannelValueAsync gestartet wurden. Der Prototyp der Callback-Funktion ist hierbei:

```
typedef void (*TYasdiEventNewChannelValue)(
                        DWORD dChannelHandle,
                        DWORD dDeviceHandle,
                        double dValue,
                        char * cpTextValue,
                        int iErorrCode);
```

Parameter:

dChannelHandle: das Kanalhandle

dDeviceHandle: das Gerätehandle

dValue: der numerische Wert des Kanals

cpTextValue: Verweis auf den textuellen Wert des Messwertes

iErorrCode: Ergebnis der Kanalabfrage:

- YE_TIMEOUT: Timeout, Geräte nicht erreicht
- YE_OK: Kanalwert gültig

4.1.5 YasdiMasterRemEventListener

Die Funktion entfernt eine vorher mit yasdiMasterAddEventListener eingefügte Call-back-Funktion.

```
void yasdiMasterRemEventListener(
                              void * eventCallback,
                              BYTE bEventType)
```

Parameter:

EventCallback:

- Die Funktion die mit yasdiMasterAddEventListener eingefügt wurde.

BEventType:

- Der Eventtyp der Callback-Funktion. Möglich ist hier:
- YASDI_EVENT_DEVICE_DETECTION
- YASDI_EVENT_CHANNEL_NEW_VALUE

4.2 Steuern der Schnittstellentreiber

Die Funktion erfragt alle verfügbaren YASDI-Schnittstellen (RS232, RS485, Powerline, IP,...). YASDI kann zur Laufzeit mehrere Schnittstellen gleichzeitig verwenden. Die Konfiguration der Schnittstellen erfolgt über die YASDI-Konfigurationsdatei (INI-Datei).

```
DWORD YasdiMasterGetDriver(DWORD * dwpDriverIDArray,
                           int iMaxDriverIDs);
```

Parameter:

dwpDriverIDArray:

- Zeiger auf DWORD Array zur Aufnahme der IDs der Schnittstellentreiber

iMaxDriverIDs:

- Maximalanzahl der zu liefernden Schnittstellen IDs

Return Wert:

Anzahl der zurückgelieferten Schnittstellen-IDs

4.2.1 YasdiMasterGetDriverName

Liefert zu einer Schnittstellen-ID den zugehörigen Schnittstellennamen (z.B. „COM1“) zurück.

```
DWORD yasdiMasterGetDriverName(DWORD  dDriverID,
                               char * cpDestBuffer,
                               DWORD  dMaxBufferSize);
```

Er liefert zum Beispiel den Text "COM1" bei der ersten seriellen Schnittstelle.

Parameter:

dDriverID :

- Die ID des Treibers

cpDestBuffer:

- Zeiger auf den Puffer zur Aufnahme des Namens

dMaxBufferSize:

- maximale Größe des Puffers zur Aufnahme des Namens

Return Wert:

Die Funktion liefert die Anzahl der benutzten Zeichen im Zielpuffer zurück.

4.2.2 YasdiMasterSetDriverOnline

Eine Schnittstelle von YASDI lässt sich mit dieser Funktion auf online schalten. Das bedeutet, dass diese Schnittstelle von YASDI sofort verwendet werden kann.

```
BOOL yasdiMasterSetDriverOnline( DWORD dDriverID );
```

Parameter:

dDriverID:

- Gibt die Schnittstelle an, die angeschaltet werden soll.

Return Wert:

Die Funktion liefert bei Erfolg "TRUE" (<>0) zurück. Das bedeutet, die Schnittstelle ließ sich anschalten. Bei Fehler liefert diese "FALSE" (0) zurück. Möglicherweise wird in diesem Fall die Schnittstelle bereits von einem anderen Programm verwendet.

4.2.3 YasdiMasterSetDriverOffline

Dieser Aufruf aktiviert eine Schnittstelle. Die Schnittstelle kann anschließend nicht mehr von YASDI verwendet werden.

```
void yasdiMasterSetDriverOffline(DWORD dDriverID);
```

Parameter:

dDriverID:

- Gibt die Schnittstelle an, die deaktiviert werden soll.

Return Wert:

Keiner

4.2.4 YasdiMasterDoDriverIoCtrl

Ausführen von Treiberspezifischen Kommandos.

```
int yasdiMasterDoDriverIoCtrl(DWORD DriverID,
                                                    int cmd,
                                                    BYTE * params)
```

Parameter:

DriverID: die Treiber-ID

- cmd: Der Kommando-String zum Ausführen
- Params: ggf. Parameters des Kommandos

Return-Wert :

Der Rückgabewert ist kommandospezifisch.

4.3 Gerätemanagement

4.3.1 DoStartDeviceDetection

Die folgende Funktion ermöglicht das Suchen von SMA Geräten.

```
Int DoStartDeviceDetection(
              /*[IN]*/ int iCountDevsToBePresent,
              /*[IN]*/ BOOL bWaitForDone)
```

Parameter:

iCountDevsToBePresent:

- Anzahl der Geräte, die in der aktuellen Anlage enthalten sind

bWaitForDone:

- Warte auf die Beendigung der Erfassung (true: Funktion blockiert bis zum Erfassungsende, false: Funktion kehrt sofort zurück (synchron). Im Falle des Wartens, werden die Ereignisse über die Funktion YASDI_EVENT_DEVICE_DETECTION gesendet.

Return Wert:

YE_OK :

- Anzahl der angegebenen Geräte wurde gefunden.

YE_NOT_ALL_DEVS_FOUND:

- Die Anzahl der Geräte konnte nicht gefunden werden.

YE_DEV_DETECT_IN_PROGRESS:

- Es läuft bereits eine Erfassung

4.3.2 DoStopDeviceDetection

Die Funktion bricht eine laufende asynchrone Erfassung ab. Der Abbruch wird ggf. verzögert durchgeführt.

```
int DoStopDeviceDetection( void );
```

Parameter:

- keine –

Retrun Wert:

YE_OK: Erfassung beendet.

4.3.3 RemoveDevice

Die Funktion entfernt ein Gerät aus den internen Listen von YASDI.

```
int RemoveDevice( DWORD DeviceHandle );
```

Parameter:

"DeviceHandle": Das Gerätehandle

Return-Wert:

YE_OK: Gerät entfernt

YE_UNNOWN_HANDLE: Unbekanntes Gerätehandle

4.3.4 GetDeviceHandles

Die Funktion ermöglicht die Abfrage aller derzeit erfassten Geräte. Pro Gerät wird genau ein Handle zurückgeliefert.

```
DWORD GetDeviceHandles(
                        DWORD * pdHandles,
                        DWORD iHandleCount)
```

Parameter:

pdHandles:

- Verweist auf ein Array, das die Handles der Geräte aufnehmen soll. Jedes Handle entspricht einem Datentyp DWORD. Falls ein NULL-Zeiger hier übergeben wird, liefert die Funktion nur die genaue Anzahl der Geräte zurück.

iHandleCount:

- Maximalgröße des Array, die aufgenommen werden darf.

Return – Wert:

Der Rückgabewert der Funktion entspricht der Anzahl der tatsächlich im Array abgelegten Handles.

4.3.5 GetDeviceName

Die Funktion liefert zu einem Gerätehandle den Gerätenamen. Jeder Gerätename setzt sich derzeit aus dem Gerätetyp, dem String "SN:" sowie dessen Seriennummer zusammen.

```
int GetDeviceName( DWORD dDevHandle, char * cpDestBuffer, int
iLen)
```

Parameter:

dDevHandle: das Gerätehandle

cpDestBuffer: Puffer zur Aufnahme des Namens

iLen: Größe des Namenspuffers

Return-Wert:

Der Rückgabewert der Funktion entspricht der tatsächlich verwendeten Platzbedarfs des Strings (String-Länge).

4.3.6 GetDeviceSN

Mit dieser Funktion wird die genaue Seriennummer eines Geräts erfragt. Die Seriennummer entspricht einem 32-bit Wert.

```
int GetDeviceSN( DWORD pdDevHandle, DWORD * pdSNBuffer);
```

Parameter:

pdDevHandle: Das Gerätehandle

pdSNBuffer: Zeiger auf ein DWORD zur Aufnahme der Seriennummer.

Return-Wert:

YE_OK: ok, Seriennummer geliefert

YE_UNNOWN_HANDLE: Unbekanntes Handle, Gerät unbekannt.

4.3.7 GetDeviceType

Hiermit kann der Gerätetyp ermittelt werden. Der Gerätetyp entspricht einem String mit zurzeit 8-Zeichen-Länge (siehe SMA-Data Spezifikation "SMA DAT-12:ZD").

```
int GetDeviceType(DWORD DevHandle, char * DestBuffer, int len);
```

Parameter:

DevHandle: Das Gerätehandle

DestBuffer: Puffer zur Aufnahme des Typs

len: Größe des Puffers zur Aufnahme des Typs

Return-Wert:

YE_OK: Gerätetyp kopiert

YE_UNNOWN_HANDLE: Unbekanntes Gerätehandle

4.4 Funktionen zur Abfrage von Messkanälen

4.4.1 GetChannelHandlesEx

Hiermit können alle aktuellen sichtbaren Kanäle eines Gerätes ermittelt werden. Die entspricht dem aktuellen Zugriffslevel.

```
DWORD GetChannelHandlesEx(DWORD pdDevHandle,
                          DWORD * pdChanHandles,
                          DWORD dMaxHandleCount,
                          TChanType chanType );
```

Parameter:

pdDevHandle:

- Das Gerätehandle

pdChanHandles:

- Verweis auf ein DWORD-Array zur Aufnahme der Kleinhandels

dMaxHandleCount:

- Die Größe des Array in Anzahl der DWORDs

chanType:

Kanaltypen, die kopiert werden sollen. Folgende sind möglich:

- SPOTCHANNELS: Alle Momentanwertkanäle
- PARAMCHANNELS: Alle Parameterkanäle
- TESTCHANNELS: Alle Testkanäle
- ALLCHANNELS: Alle obigen Kanäle.

Return-Wert:

Liefert die Anzahl der Kanalhandles zurück, die in das Array kopiert wurden.

Der aktuelle Zugriffslevel kann ggf. mit der Funtion „yasdiMasterSetAccessLevel“ verändert werden, um auch ggf. geschützte Kanäle zu erhalten.

4.4.2 FindChannelName

Mit Hilfe dieser Funktion kann ein Kanal anhand seines Namens gesucht werden.

```
DWORD FindChannelName(DWORD pdDevHandle, char * cpChanName);
```

Parameter:

pdDevHandle: das Gerätehandle

cpChanName: Zeiger auf Puffer zur Aufnahme des Namens

Return-Wert:

INVALID_HANDLE: (<0): Handle ungültig

>0: Das Handle des Kanals, der dem Namen entspricht

4.4.3 GetChannelName

Funktion liefert den Namen des Kanals.

```
int GetChannelName( DWORD dChanHandle,
                    char * cpChanName,
                    DWORD dChanNameMaxBuf);
```

Parameter:

dChanHandle: Das Kanalhandle

cpChanName: Der Puffer zur Aufnahme des Kanalnamens

dChanNamemaxBuf: Puffergröße

Return-Wert:

YE_OK: alles ok

YE_UNNOWN_HANDLE: Unbekanntes Handle

4.4.4 GetChannelValue

Diese Funktion liefert den Wert eines Kanals. Wie "alt" der Kanalwert sein darf, kann der Funktion vorgegeben werden. Die Funktion arbeitet synchron, d.h. sie blockiert solange, bis der Kanalwert ermittelt wurde oder ein Fehler aufgetreten ist. Das Blo-ckieren kann ggf. mehrere Sekunden dauern.

```
int GetChannelValue(DWORD dChannelHandle,
                    DWORD dDeviceHandle,
                    double * dblValue,
                    char * ValText,
                    DWORD dMaxValTextSize,
                    DWORD dMaxChanValAge)
```

Parameter:

dChannelHandle: Das Kanalhandle

dDeviceHandle: Das Gerätehandle

dblValue: Verweis auf den numerischen Kanalwert.

ValText: Verweis auf den optionalen textuellen Wert

dMaxValtextSize: Größe des textuellen Puffers

dMaxChanValAge: Maximales Alter des Kanalwertes in Sekunden

Ein „dMaxChanValAge“-Wert von "0" erzwingt eine Abholung des aktuellen Wertes. Kanalwerte werden ggf. zwischengespeichert. Ein Wert von 10 bedeutet, dass der Kanalwert höchstens 10 Sekunden alt sein darf. Wenn er älter ist, wird er automatisch vom Gerät neu angefordert.

Return-Wert:

YE_OK: Alles Ok: Kanalwert ist gültig.

YE_UNNOWN_HANDLE: Unbekannte Handle

YE_TIMEOUT: Wert konnte vom Gerät nicht ermittelt werden (keine Antwort)

YE_VALUE_NOT_VALID: Wert ist nicht gültig

4.4.5 GetChannelValueAsync

Diese Funktion liefert den Wert eines Kanals asynchron. Wie "alt" der Kanalwert sein darf, kann der Funktion vorgegeben werden. Die Funktion blockiert beim Aufruf nicht. Es wird über die Callback-Funktion ein Event gesendet, wenn der Wert verfügbar ist.

```
int GetChannelValueAsync(DWORD dChannelHandle,
                         DWORD dDeviceHandle,
                         DWORD dMaxChanValAge );
```

Parameter:

dChannelHandle: Das Kanalhandle

dDeviceHandle: das Gerätehandle

dMaxChanValAge: maximales Alter des Kanalwertes in Sekunden

Return-Wert:

YE_OK: alles OK, Abfrage läuft

YE_UNNOWN_HANDLE: Ein Kanalhandle ist unbekannt

YE_NO_ACCESS_RIGHTS: Kein aktueller Zugriff auf den Kanal

Der Kanalwert wird über die Callbackfunktion übergeben, die per Funktion „yasdi-MasterAddEventListener()“ eingehängt werden kann.

4.4.6 GetChannelValueTimeStamp

Der Funktionsaufruf liefert den Zeitstempel eines Kanalwerts. Die Zeit wird in der Zeitzone Greenwich Mean Time (GMT+0) als UNIX-Zeit übergeben.

```
DWORD GetChannelValueTimeStamp( DWORD dChannelHandle, DWORD
dDeviceHandle)
```

Parameter:

dChannelHandle: Das Kanalhandle

dDeviceHandle: Das Gerätehandle

Return-Wert:

Unix-Zeit (GMT+0)

4.4.7 GetChannelUnit

Liefert die Kanaleinheit eines Kanal als Zeichenkette zurück.

```
int GetChannelUnit( DWORD dChannelHandle,
                    char * cChanUnit,
                    DWORD cChanUnitMaxSize)
```

Parameter:

dChannelHandle: Kanalhandle

cChanUnit: Verweis auf Puffer zur Aufnahme der Einheit

cHcnaUnitMaxSize: Größe des Puffers

Return-Wert:

YE_OK: kein Fehler

YE_UNNOWN_HANDLE: Unbekanntes Handle

4.4.8 SetChannelValue

Setzt den numerischen Wert eines Kanals. Die Funktion blockiert bis der Wert gesetzt werden konnte, oder ein Timeout aufgetreten ist.

```
int SetChannelValue( DWORD  dChannelHandle,
                     DWORD  dDevHandle,
                     double dblValue)
```

Parameter:

dChannelHandle: Das Kanalhandle

dDevHandle: das Gerätehandle

dblValue: Der numerische Wert des Kanals zum Setzen

Return-Wert:

YE_OK: kein Fehler

YE_UNKNOWN_HANDLE: unbekanntes Handle übergeben

YE_NO_ACCESS_RIGHTS: Keine Berechtigung zum Setzen des Kanals

YE_VALUE_NOT_VALID: Der Wert des Kanals liegt außerhalb der erlaubten Grenzen

YE_TIMEOUT: Gerät antwortet nicht, Wert nicht gesetzt

4.4.9 SetChannelValueString

Die Funktion setzt einen Kanalwert, den der String übergeben wird. Der String muss ein gültiger Statustextwert sein. Numerische Werte werden mit der Funktion „Set-ChannelValue()“ gesetzt. Die Funktion blockiert, bis der Wert gesetzt oder ein Timeout aufgetreten ist.

```
int SetChannelValueString(DWORD dChannelHandle,
                          DWORD dDevHandle,
                          const char * cpChanvalstr )
```

Parameter:

„dChannelHandle”: das Kanalhandle

“dDevHandle”: das Gerätehandle

“cpChanvalstr”: der Statustext, der gesetzt worden soll

Return-Wert:

YE_OK: alles ok

YE_UNKNOWN_HANDLE: Unbekanntes Handle übergeben

YE_TIMEOUT: Gerät hat nicht geantwortet

YE_CHAN_TYPE_MISMATCH: Der Kanal ist kein Textkanal

YE_INVAL_ARGUMENT: Der Textwert ist nicht gültig

4.4.10 GetChannelStatTextCnt

Falls es zu einem Kanal Statustexte gibt, so liefert diese Funktion die Anzahl der Kanaltexte dieses Kanals zurück. Die Texte können dann anschließend einfach von der Funktion "GetChannelStatText(...)" abgefragt werden.

```
int GetChannelStatTextCnt(DWORD dChannelHandle)
```

Parameter:

dChannelHandle: das Kanalhandle.

Return Wert:

Anzahl der Kanaltexte dieses Kanals.

4.4.11 GetChannelStatText

Diese Funktion liefert einen bestimmten Statustext des Kanals zurück. Die Anzahl der Texte sollte mit der Funktion GetChannelStatTextCnt(...) vorher abgefragt werden.

```
int GetChannelStatText(DWORD dChannelHandle,
                       int iStatTextIndex,
                       char * TextBuffer,
                       int BufferSize);
```

Parameter:

"dChanneHandle": Das Kanalhandle

"iStatTextIndex": der Index des abzufragenden Textes.

"TextBuffer": Zeiger auf den Speicherbereich, in den der Text kopiert wird.

"BufferSize": Puffergröße

Return-Wert:

YE_OK: alles ok (Ergebnis gültig)

YE_INVALID_HANDLE: ungültiges Kanalhandle wurde übergeben.

4.4.12 YasdiMasterGetChannelPropertyDouble

Mit dieser Funktion können SMAData-Protokoll-spezifische Informationen abgefragt werden.

```
int yasdiMasterGetChannelPropertyDouble(DWORD chanHandle,
                                        char * propertyStr,
                                        double * result
```

Parameter:

ChanHandle: Das Kanalhandle

PropertyStr: Der Name der Eigenschaft zur Abfrage

Result: Der abgefragte Wert (double Wert).

Return-Wert:

YE_OK: ok, Wert kopiert

YE_INVAL_ARGUMENT: Unbekannte Eigenschaft

Als Eigenschaften stehen derzeit zur Verfügung:

"smadata1.cindex" (“Kanalindex”)

"smadata1.ctype" (Kanaltyp)

"smadata1.ntype" (numerische Kanaltyp)

"smadata1.level" (Zugriffslevelwert)

"smadata1.gain" (Gain-Wert)

"smadata1.offset": Kanalwertoffset

Weitergehende Informationen zu den Eigenschaften sind in der SMAData(1) Spezifikation nachzuschlagen.

4.4.13 GetChannelAccessRights

Die Funktion liefert nähere Zugriffsrechte zu einem Kanal.

```
int GetChannelAccessRights(DWORD dchannelHandle,
                           BYTE * accessrights);
```

Parameter:

"dchannelHandle": Das Kanalhandle

"accessrights": Liefert die Zugriffsrechte des Kanals binär kodiert. Dabei gilt:

- CAR_READ: Kanal ist im aktuellen Zugriffslevel lesbar
- CAR_WRITE: Kanal ist im aktuellen Zugriffslevel schreibbar

Return-Wert:

YE_OK: ok

YE_UNNOWN_HANDLE: Das Handle war ungültig.

4.4.14 GetChannelArraySize

Liefert die Größe eines Kanalwertes zurück, d.h. aus wie vielen Werten ein einzelner Kanalwert besteht (Array-Tiefe).

```
int GetChannelArraySize(DWORD chanhandle, WORD * wpArrayDeep);
```

Parameter:

channelHandle: Das Kanalhandle

wpArrayDeep: Verweis auf Ergebniswert (WORD)

Return-Wert:

YE_OK: alles ok

YE_UNNOWN_HANDLE: unbekanntes Handle

4.4.15 GetChannelValRange

Die Funktion liefert den möglichen Kanalwertbereich zurück, der für diesen Kanal möglich ist.

```
int GetChannelValRange(DWORD dChannelHandle,
                       double * min,
                       double * max );
```

Parameter:

dChannelHandle: Das Kanalhandle

min: Verweis auf Double Wert zur Aufnahme des Minimalwertes

max: Verweis auf Double Wert zur Aufnahme des Maximalwertes

Return-Wert:

YE_OK: ok, Werte gültig

YE_UNKNOWN_HANDLE: Unbekannte Handle übergeben

-3: Es gibt keinen dezidierten Wertebereich für den Kanal

4.5 API-Benutzung an einem Beispiel

Eine typische Funktionsaufruf-Reihenfolge der YASDI-API zur Abfrage von SMA Data-Geräten könnte folgendermaßen aussehen:

```
/* YASDI initialisieren */
yasdiMasterInitialize(...)

/* Liefere alle Schnittstellentreiber, die YASDI kennt
   und schalte die benötigten an... */
yasdiGetDriver(...)
yasdiSetDriverOnline(...)
/* Suche alle angeschlossenen Geräte (hier genau eines) */
DoMasterCmdEx("detect",1,NULL,NULL);

/* Gib mir alle Gerätehandles */
GetDeviceHandles(...)

/* Gib mir zu diesem Gerät alle Kanalhandles */
GetChannelHandles(...)

/* Kanalwerte abfragen oder setzen */
While( youWant )
{
SetChannelValue(...) oder  GetChannelValue(...)
}

/* verwendete Schnittstellen wieder ausschalten */
yasdiSetDriverOffline(...)

/* YASDI beenden */
yasdiMasterShutdown()
```

Es ist darauf zu achten, dass zu Begin der Benutzung die Funktion "yasdiMasterInitialize()" und am Ende "yasdiMasterShutdown()" jeweils einmal aufgerufen werden. Alle anderen Funktionen der Master API können beliebig oft in beliebiger Reihenfolge benutzt werden.

5 Konfigurationsdatei

Die von YASDI verwendete Konfigurationsdatei verwendet das von Windows bekannte INI-Format. Dieses wird auch in der Linux-Variante verwendet.

Der Pfad zur Konfigurationsdatei wird der Master-Funktion "yasdiMasterInitialize(...)" übergeben (intern wird der Pfad automatisch zur "yasdiInitialize(...)"-Funktion durchgereicht). Die Datei besteht aus verschiedenen Sektionen:

Sektion: **"DriverModules"**	
Eintrag	**Beschreibung**
"Driver0" ... "Driver9	"Die zu verwendenden Schnittstellentreiber (z. B. **yasdi_drv_serial.dll** unter Windows oder **libyasdi_drv_serial.so** unter Linux), der zur Laufzeit geladen werden soll.

Serieller Treiber Sektionen: "COMx" (x=COM-Portnummer)	
Eintrag	**Beschreibung**
Device	Der Dateiname, der der seriellen Schnittstelle des jeweiligen Betriebssystems entspricht. (Für die erste serielle Schnittstelle Windows: "COM1", Linux: "/dev/ttyS0"
Media	Das Medium, das der serielle Treiber verwenden soll. Zurzeit werden **Powerline, RS232** und **RS485** unterstützt.
Baudrate	Die Geschwindigkeit der seriellen Schnittstelle in Bits pro Sekunde. Üblich für Sunny Boys sind "1200", sowie für ein Sunny Boy Control "19200".
Protocol	Das zu verwendende Transportprotokoll. Zur Auswahl stehen: "**SunnyNet**" oder "**SMANet**". Ältere Sunny Boy Control verwenden nur SunnyNet. Die neueren Geräte können beide Protokolle sprechen. Es können auch gleichzeitig beide verwendet werden: **"SMANet,SunnyNet"**

IP-Treiber Sektion: "**IP1**"	
Device0, Device1, ...DeviceX	Angabe der IP-Adresse des Gerätes (Sunny Boy Control mit Ethernet-Piggy-Back), z. B. Device0=192.168.18.1. Es können beliebig viele Adressen angegeben werden, jeder mit einem eigenen "DeviceX" Eintrag.
Protocol	Wie beim seriellen Modul (siehe dort)
MasterMode	(Optional) Default ist "MasterMode"==0: "Master Mode": <>0: "Slave Mode"

Sektion: "**Misc**"	
Eintrag	**Beschreibung**
StatisticOutput	Dateiangabe (Pfad + Datei) für Ausgabe von statistischen Werten. Es wird Speicherverbrauch von YASDI sowie die Anzahl der gesendeten und empfangenen Pakete zu den Geräten in einer Datei im XML-Format festgehalten. Der Eintrag ist optional. Der Eintrag dient lediglich zur Fehlersuche.
DebugOutput	Angabe einer Datei, in der Debug-Informationen gesichert werden sollen: Es kann eine Datei sowie die Angaben "stdout" oder "stderr" angegeben werden.

Sektion: "**Master**"	
Eintrag	**Beschreibung**
NetAddress	Optionaler Parameter zum Einstellen der Netzadresse des SMA-Data-Masters. Der Wertebereich liegt zwischen "0" (Default) und "65535".
ReadParamChanTimeout	Timeoutzeit in Sekunden beim Lesen von Parameterwerten der Geräte. Der Eintrag ist optional und per Default "6".
ReadParamChanRetry	Anzahl der erneuten Versuche beim Lesen von Parameterkanälen (nach Timeout). Der Eintrag ist optional. Defaultwert: "4" (Wiederholungen).
WriteParamChanTimeout	Optionaler Parameter zur Angabe der Timeoutzeit beim Schreiben von Parameterkanälen. Defaultwert: "6".

Sektion: "**Master**"	
Eintrag	**Beschreibung**
WriteParamChanRetry	Optionaler Parameter zur Angabe der Wiederholungen beim Parametersetzen (Default "4").
ReadSpotChanTimeout	Optionaler Parameter zur Angabe der Timeoutzeit beim Lesen von Spotwertkanälen. Defaultwert:"6"
ReadSpotChanRetry	Anzahl der Wiederholungen beim Lesen von Spotwertkanälen (nach Timeout). Eintrag ist optional (Default "4").
DeviceAddrRangeLow	Die untere Grenze des erlaubten Netzadressenbereichs (Geräteadressenanteil) eines erfassten Gerätes. Der Eintrag ist optional. Der Wertebereich erstreckt sich von "0" (default) bis "255" einschließlich.
DeviceAddrRangeHigh	Die obere Grenze des erlaubten Netzadressenbereichs (Geräteadressenanteil) eines erfassten Gerätes. Der Eintrag ist optional. Der Wertebereich erstreckt sich von "0" bis "255" (Default) einschließlich.
DeviceAddrBusRangeLow	Die untere Grenze des erlaubten Netzadressenbereichs (Busadressenanteil) eines erfassten Gerätes. Der Eintrag ist optional. Der Wertebereich erstreckt sich von "0" bis "255" (Default) einschließlich.
DeviceAddrBusRangeHigh	Die obere Grenze des erlaubten Netzadressenbereichs (Busadressenanteil) eines erfassten Gerätes. Der Eintrag ist optional. Der Wertebereich erstreckt sich von "0" bis "255" (Default) einschließlich.
DeviceAddrStringRangeLow	Die untere Grenze des erlaubten Netzadressenbereichs (Strangadressenanteil) eines erfassten Gerätes. Der Eintrag ist optional. Der Wertebereich erstreckt sich von "0" bis "255" (Default) einschließlich.
DeviceAddrStringRangeHigh	Die obere Grenze des erlaubten Netzadressenbereichs (Strangadressenanteil) eines erfassten Gerätes. Der Eintrag ist optional. Der Wertebereich erstreckt sich von "0" bis "255" (Default) einschließlich.

Eine Konfigurationsdatei unter Windows könnte wie folgt aussehen:

Beispiel:

```
[DriverModules]
Driver0=yasdi_drv_serial.dll

#### Abschnitt der ersten seriellen Schnittstelle
[COM1]
Device=COM1
Media=RS232
Baudrate=1200
Protocol=SMANet

[Misc]

[Master]
```

6 Source–Verzeichnisstrukturen

YASDI verwendet die folgende Verzeichnisstruktur, die im folgenden kurz beschrie-ben ist:

Unterverzeichnis	Beschreibung
core	Der Kern von YASDI.
driver	Hier sind alle Treiber abgelegt, die YASDI unterstützt. Zurzeit sind dies die serielle Treiber und ein Treiber für Kommunikation über IP
include	Diverse Include-Dateien
libs	Die C-API-Schnittstellen
master	Implementierung des SMA-Data-Masters
os	Betriebssystemabstraktionsschichten: Windows, Linux, MacOSX....
protocol	Implementierungen der Transportprotokolle "SMA-Net" und "SunnyNet".
smalib	Diverse externe Support-Module (z.B. INI-Datei-lesen)
projects	Spezielle Builds, mit denen YASDI auf verschiedenen System zusammengebaut werden kann. Derzeit: Windows-lib: Makefiles erzeugt für Borland C++ Builder 5"Generic-cmake": Makefiles zum Erstellen von YASDI mittels CMAKE (siehe www.cmake.org)
shell	Ein kleines Beispielprogramm (Kommandozeilentool), das YASDI verwendet.

7 Kompilieren von YASDI

YASDI wird mit dem Build-System „CMAKE“ (siehe www.cmake.org) zusammengebaut. CMAKE erstellt Makefiles für verschiedene Compiler und Betriebssysteme. Eine unvollständige Liste ist z. B. folgende:

- GNU-Makefiles (für UNIX, und MinGW unter Windows)
- Microsoft Visual Studio
- Xcode (für Mac OS X)

Im Folgenden wird das Kompilieren von YASDI unter Linux für GNU-Makefiles in einer Bash beschrieben:

Beispiel:

```
bash>  cd <yasdi-source>

bash>  cd builds\generic-cmake

bash>  cmake .

bash>  make

bash>  # Starten der YasdiShell

bash>  export LD_LIBRARY_PATH=.

Bash>  ./yasdishell
```

Die in diesen Unterlagen enthaltenen Informationen sind Eigentum der SMA Solar Technology AG. Die Veröffentlichung, ganz oder in Teilen, bedarf der schriftlichen Zustimmung der SMA Solar Technology AG. Eine innerbetriebliche Vervielfältigung, die zur Evaluierung des Produktes oder zum sachgemäßen Einsatz bestimmt ist, ist erlaubt und nicht genehmigungspflichtig.

Haftungsausschluss

Es gelten als Grundsatz die Allgemeinen Lieferbedingungen der SMA Solar Technology AG.

Der Inhalt dieser Unterlagen wird fortlaufend überprüft und gegebenenfalls angepasst. Trotzdem können Abweichungen nicht ausgeschlossen werden. Es wird keine Gewähr für Vollständigkeit gegeben. Die jeweils aktuelle Version ist im Internet unter www.SMA.de abrufbar oder über die üblichen Vertriebswege zu beziehen.

Gewährleistungs- und Haftungsansprüche bei Schäden jeglicher Art sind ausgeschlossen, wenn sie auf eine oder mehrere der folgenden Ursachen zurückzuführen sind:

- Transportschäden
- Unsachgemäße oder nicht bestimmungsgemäße Verwendung des Produkts
- Betreiben des Produkts in einer nicht vorgesehenen Umgebung
- Betreiben des Produkts unter Nichtberücksichtigung der am Einsatzort relevanten gesetzlichen Sicherheitsvorschriften
- Nichtbeachten der Warn- und Sicherheitshinweise in allen für das Produkt relevanten Unterlagen
- Betreiben des Produkts unter fehlerhaften Sicherheits- und Schutzbedingungen
- Eigenmächtiges Verändern oder Reparieren des Produkts oder der mitgelieferten Software
- Fehlverhalten des Produkts durch Einwirkung angeschlossener oder benachbarter Geräte außerhalb der gesetzlich zulässigen Grenzwerte
- Katastrophenfälle und höhere Gewalt

Die Nutzung der mitgelieferten von der SMA Solar Technology AG hergestellten Software unterliegt zusätzlich den folgenden Bedingungen:

- Die SMA Solar Technology AG lehnt jegliche Haftung für direkte oder indirekte Folgeschäden, die sich aus der Verwendung der von SMA Solar Technology AG erstellten Software ergeben, ab. Dies gilt auch für die Leistung beziehungsweise Nicht-Leistung von Support-Tätigkeiten.
- Mitgelieferte Software, die nicht von der SMA Solar Technology AG erstellt wurde, unterliegt den jeweiligen Lizenz- und Haftungsvereinbarungen des Herstellers.

SMA-Werksgarantie

Die aktuellen Garantiebedingungen liegen Ihrem Gerät bei. Bei Bedarf können Sie diese auch im Internet unter www.SMA.de herunterladen oder über die üblichen Vertriebswege in Papierform beziehen.

Warenzeichen

Alle Warenzeichen werden anerkannt, auch wenn diese nicht gesondert gekennzeichnet sind. Fehlende Kennzeichnung bedeutet nicht, eine Ware oder ein Zeichen seien frei.

SMA Solar Technology AG

Sonnenallee 1

34266 Niestetal

Deutschland

Tel. +49 561 9522-0

Fax +49 561 9522-100

www.SMA.de

E-Mail: info@SMA.de

SMA Solar Technology AG

www.SMA.de

Sonnenallee 1
34266 Niestetal, Germany
Tel.: +49 561 9522 4000
Fax: +49 561 9522 4040
E-Mail: Vertrieb@SMA.de
Freecall: 0800 SUNNYBOY
Freecall: 0800 78669269