YASDI API documentation

technical description

Contents 1

1 Explanations of the symbols used

In order to guarantee optimum use of this manual and a safe modules used in the phases of commissioning, operation and maintenance, please note the following explanation of the symbols used.

This symbol indicates a fact which is important for the optimal operation of your product. Read these sections therefore carefully.

2 Introduction

This document describes the structure and use of the software "YASDI" to communicate with SMA devices. The name "YASDI" stands for " Y et A nother S MA D ata I mplementation ".

The software implements the communication by "SMA Data (1) protocol" on the transport protocols "Sunny-Net" and "SMA-Net" with SMA devices (eg Sunny Boy inverters) as a software library without its own graphical interface.

YASDI supports querying current measurement values (instantaneous values) as well as reading and possibly setting device parameters. A query of archived measured values of connected data loggers (eg., Sunny Boy Control or Sunny Beam) are not possible. Currently, the following communication Medias are supported:

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

The software has been designed so that it can be easily adapted to other environments (operating systems). Currently there are primary ports for Windows, Linux and Apple Macintosh (Mac OS X). All system-dependent functions are abstracted software layers of the operating system, making them easily portable.

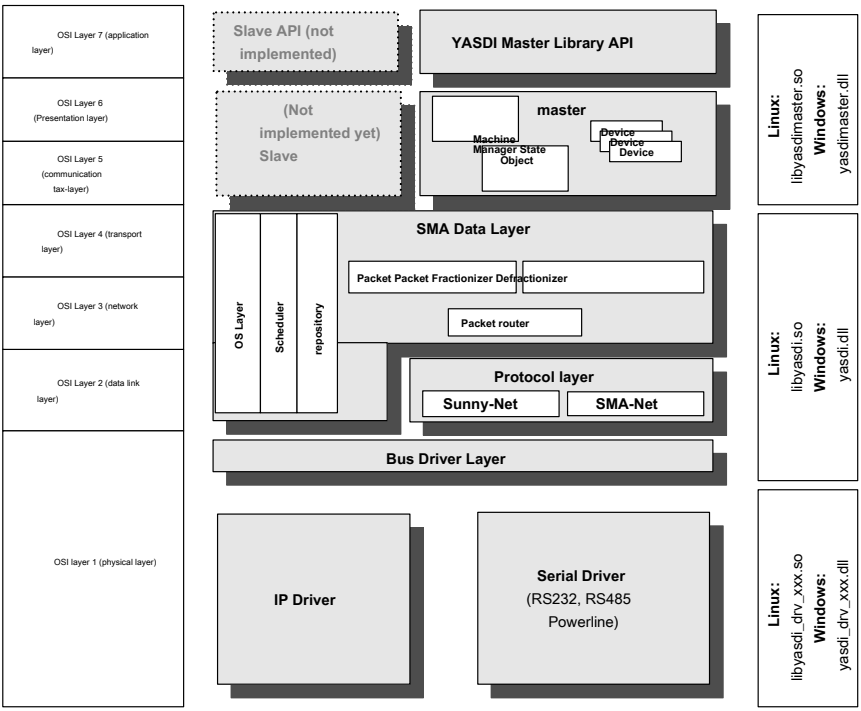
The software is written in the programming language "C" and allows for maximum portability to other target platforms.

The implementations for Windows and Linux are designed as dynamic libraries. Use as a static library in the context of a monolithic program is also possible.

Currently, YASDI as source code available (license "LGPL") and compiled (for Windows only).

2.1 Software Overview

When implementing YASDI this was inspired by the OSI model for network protocols. The individual layers are summarized in certain libraries that can be taken from the following table:



Data types used 3

In the interface descriptions data types are used, which are listed below (as defined in the file "smadef.h"):

data type	Data Type Description
DWORD	32-bit unsigned
WORD	16-bit unsigned
BYTE	8-bit unsigned
BOOL	8-bit unsigned
DOUBLE	64-bit double value (8 bytes)

4 Function Reference

The library "yasdimaster" contains functions for controlling the so-called Masters. This allows the active retrieval of information or measured values of SMA devices.

4.1 functions for initialization

4.1.1 YasdiMasterInitialize

Initialization of YASDI master library. The function must be called exactly once before all other functions of the library.

(Void yasdiMasterInitialize char * cINIFileName,
DWORD * pDriverCount);

Parameter:

cINIFileName:

- File name of the configuration file (yasdi.ini) including path. More to the configuration file in Chapter 4 pDriverCount:

- Refers to a variable in which the number of currently loaded in YASDI interface driver be returned after the call.

4.1.2 YasdiMasterShutdown

This function call memory used and resources of YASDI Master Library are released. The function should always be called by using YASDI. There are no parameters required for handover.

yasdiMasterShutdown void (void);

Parameters: - None -

Return Value: -keiner-

4.1.3 YasdiReset

The function performs a full reset of the software. All currently detected devices may be removed. The software is then in a state as "yasiMasterInitialize (...)" after calling the function.

yasdiReset void (void)

Parameters: - None -

Return Value: -keiner-

4.1.4 YasdiMasterAddEventListener

The function inserts a callback function that can be serviced with the asynchronous events from YASDI.

void yasdiMasterAddEventListener (void * vpEventCallback,

BYTE bEventType);

Parameter:

vpEventCallback:

- Function pointer for hooking. For each type event class has its own function pointer can be suspended. BEventType:

- The type of event that is to be observed. The following types are currently available: YASDI_EVENT_DEVICE_DETECTION:

Übewachung of equipment acquisitions. It is jumped with the following type the function:

typedef void (* TYasdiEventDeviceDetection) (

BYTE subEvent, DWORD devHandle);

"SubEvent" is one of the following events:

- YASDI_EVENT_DEVICE_ADDED:
  - New device: device handle as "devHandle"
- YASDI_EVENT_DEVICE_REMOVED:
  - Device removed: device handle as "devHandle"
- YASDI_EVENT_DEVICE_SEARCH_END:
  - Acquisition completed: (no parameters)

YASDI_EVENT_CHANNEL_NEW_VALUE:

- Monitoring channel value queries that have been started by GetChannelValueAsync. The prototype of the callback function is this:

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

Parameter:

dChannelHandle: the channel handle

dDeviceHandle: the device handle dValue: the

numeric value of the channel

cpTextValue: reference to the textual value of the measured value iErorrCode:

result of the channel search:

- YE_TIMEOUT: timeout, device not reached
- YE_OK: channel value is valid

4.1.5 YasdiMasterRemEventListener

The function removes a previously inserted by yasdiMasterAddEventListener call-back function.

void yasdiMasterRemEventListener (
void * event callback, BYTE
bEventType)

Parameter:

EventCallback:

- The function that was inserted with yasdiMasterAddEventListener. BEventType:

- The event type of the callback function. is possible here:
- YASDI_EVENT_DEVICE_DETECTION
- YASDI_EVENT_CHANNEL_NEW_VALUE

4.2 controlling the interface driver

The function requested all available YASDI interfaces (RS232, RS485, Powerline, IP, ...). YASDI can use at runtime several interfaces simultaneously. The configuration of the interfaces via the YASDI configuration file (ini file).

DWORD YasdiMasterGetDriver (DWORD * dwpDriverIDArray,
int iMaxDriverIDs);

Parameter:

dwpDriverIDArray:

- Pointer to DWORD array to hold the IDs of the interface driver iMaxDriverIDs:

- Maximum number of IDs to be supplied interfaces

Return Value:

Number of returned interface IDs

4.2.1 YasdiMasterGetDriverName

Provides an interface ID returns the corresponding interface name (eg "COM1").

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

It provides, for example, the text "COM1" in the first serial port.

Parameter:

dDriverID:

- The ID of the driver

cpDestBuffer:

- Pointer to the buffer for receiving the name

dMaxBufferSize:

- maximum size of the buffer for receiving the name

Return Value:

The function returns the number of characters in the target buffer used back.

4.2.2 YasdiMasterSetDriverOnline

An interface of YASDI can switch to online with this function. This means that this interface can be used by YASDI immediately.

BOOL yasdiMasterSetDriverOnline (DWORD dDriverID);

Parameter:

dDriverID:

- Specifies the interface to be turned on.

Return Value:

The function returns on success "TRUE" (<> 0). This means that the interface could be turn. In this error "FALSE" (0) returns. Perhaps in this case the interface is already in use by another program.

4.2.3 YasdiMasterSetDriverOffline

This call activates an interface. The interface can then no longer be used by YASDI.

void yasdiMasterSetDriverOffline (DWORD dDriverID);

Parameter:

dDriverID:

- Specifies the interface to be disabled.

Return Value:

none

4.2.4 YasdiMasterDoDriverIoCtrl

Running driver-specific commands.

int yasdiMasterDoDriverIoCtrl (DWORD DriverID, int cmd,

BYTE * params)

Parameter:

DriverID: the driver ID

- cmd The command string to execute
- Params: If necessary parameter of the command

Return Value:

The return value is command specific.

4.3 Device Management

4.3.1 DoStartDeviceDetection

The following function allows you to find SMA devices.

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

Parameter:

iCountDevsToBePresent:

- Number of devices that are included in the current system bWaitForDone:

- Waiting for the completion of the acquisition (true: Function blocks until the detection end, false: function returns immediately (synchronous) In case of waiting, the events over the function YASDI_EVENT_DEVICE_DETECTION be sent..

Return Value:

YE_OK:

- Number of specified device was found.

YE_NOT_ALL_DEVS_FOUND:

- The number of devices could not be found.

YE_DEV_DETECT_IN_PROGRESS:

- It already runs a detection

4.3.2 DoStopDeviceDetection

The function aborts a running asynchronous collection. The termination is possibly carried out delayed.

int DoStopDeviceDetection (void);

Parameter:

- no -

Retrun value:

YE_OK: acquisition completed.

4.3.3 RemoveDevice

The function removes a device from the internal lists of YASDI.

int RemoveDevice (DWORD Device handle);

Parameter:

"Device Handle": The device handle

Return Value:

YE_OK: device removed

YE_UNNOWN_HANDLE: Unknown device handle

4.3.4 GetDeviceHandles

The function allows the query of all devices currently covered. Per device, a handle is returned exactly.

DWORD GetDeviceHandles (DWORD * pdHandles,

DWORD iHandleCount)

Parameter:

pdHandles:

- Refers to an array to accommodate the handle of the device. Each handle corresponds to a data type DWORD. If a NULL pointer is passed here, the function returns only the exact number of devices back.

iHandleCount:

- Maximum size of the array, which may be included.

Return - Value:

The return value of the function is the number of actually stored in the array handles.

4.3.5 GetDeviceName

The function returns a device handle the device name. Each device name is currently composed of the type of device, the string "SN" as well as its serial number together.

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

Parameter:

dDevHandle: the device handle

cpDestBuffer: buffer for receiving the name iLen: size

of the name buffer

Return Value:

The return value of the function corresponds to the footprint of the string actually used (string length).

4.3.6 GetDeviceSN

This function allows the exact serial number of a device is requested. The serial number corresponds to a 32-bit value.

int GetDeviceSN (DWORD pdDevHandle, DWORD * pdSNBuffer);

Parameter:

pdDevHandle: The device handle

pdSNBuffer: Pointer to a DWORD for holding the serial number.

Return Value:

YE_OK: ok, delivered serial number

YE_UNNOWN_HANDLE: Unknown handle Device unknown.

4.3.7 GetDeviceType

This device type can be determined. The device type corresponds to a string with currently 8 character-length (see SMA Data Specification "SMA DAT-12: ZD").

int GetDeviceType (DWORD DevHandle, char * Buffer least, int len);

Parameter:

DevHandle: The device handle least Buffer: buffer for

receiving the type len: size of the buffer for receiving the

type

Return Value:

YE_OK: copied device type

YE_UNNOWN_HANDLE: Unknown device handle

4.4 query functions for measurement channels

4.4.1 GetChannelHandlesEx

This all current visible channels of a device can be determined. Corresponding to the current access level.

DWORD GetChannelHandlesEx (DWORD pdDevHandle,
DWORD * pdChanHandles, DWORD
dMaxHandleCount, TChanType
chantype);

Parameter:

pdDevHandle:

- The device handle

pdChanHandles:

- Reference to a DWORD array to accommodate the retail

dMaxHandleCount:

- The array size in number of DWORD

chantype:

Channel types to be copied. The following are possible:

- SPOT CHANNELS: All instantaneous value channels
- PARAMCHANNELS: All parameter channels
- TEST CHANNELS: All test channels
- ALLCHANNELS: All of the above channels.

Return Value:

Returns the number of channel handles that have been copied to the array.

The current access level can be changed as necessary using the funtion "yasdiMasterSetAccessLevel" to also if necessary to get protected channels.

4.4.2 Find Channel Name

Using this function, a channel can be searched by name.

DWORD Find Channel Name (DWORD pdDevHandle, char * cpChanName);

Parameter:

pdDevHandle: the device handle

cpChanName: Pointer to the buffer to receive the name

Return Value:

INVALID_HANDLE: (<0): handle invalid

> 0: The handle of the channel that corresponds to the name

4.4.3 GetChannelName

Function returns the name of the channel.

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

Parameter:

dChanHandle: The Channel Handle

cpChanName: The buffer to receive the channel name

dChanNamemaxBuf: Buffer Size

Return Value:

YE_OK: everything ok

YE_UNNOWN_HANDLE: Unknown Handle

4.4.4 GetChannelValue

This function returns the value of a channel. As may be "old" the channel value, the function can be specified. The function operates synchronously, that is, they blocked until the channel value was determined or an error has occurred. The Blo-ckieren can possibly take several seconds.

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

Parameter:

dChannelHandle: The channel handle

dDeviceHandle: The device handle

dblValue: reference to the numerical value channel. ValText:

Reference to the optional textual value dMaxValtextSize:

Size of buffer textual

dMaxChanValAge: Maximum age of the channel value in seconds

A "dMaxChanValAge" value of "0" will force a pick-up of the current value. Channel values shall be cached. A value of 10 means that the channel value must be less than 10 seconds old. If he is older, he is automatically requested by the device.

Return Value:

YE_OK: Everything Ok: Channel value is valid.

YE_UNNOWN_HANDLE: Unknown Handle

YE_TIMEOUT: value could not be determined by the device (no answer)

YE_VALUE_NOT_VALID: value is not valid

4.4.5 GetChannelValueAsync

This function returns asynchronously the value of a channel. As may be "old" the channel value, the function can be specified. The function does not block the call. It will be sent an event over the callback function when the value is available.

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

Parameter:

dChannelHandle: The channel handle

dDeviceHandle: the device handle

dMaxChanValAge: maximum age of the channel value in seconds Return Value:

YE_OK: everything is OK, query runs

YE_UNNOWN_HANDLE: A channel handle is unknown

YE_NO_ACCESS_RIGHTS: No current access to the channel

The channel value is passed via the callback function that "yasdi- MasterAddEventListener ()"
per function can be suspended.

4.4.6 GetChannelValueTimeStamp

The function call returns the time stamp of a channel value. The time is passed in the time zone Greenwich Mean Time (GMT + 0) as UNIX time.

DWORD GetChannelValueTimeStamp (DWORD dChannelHandle, DWORD dDeviceHandle)

Parameter:

dChannelHandle: The channel handle

dDeviceHandle: The device handle Return value:

Unix time (GMT + 0)

4.4.7 GetChannelUnit

Returns the channel unit of a channel as a string.

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

Parameter:

dChannelHandle: Channel Handle

cChanUnit, reference to the buffer for receiving the unit

cHcnaUnitMaxSize: the buffer size

Return Value:

YE_OK: No error

YE_UNNOWN_HANDLE: Unknown Handle

4.4.8 SetChannelValue

Sets the numeric value of a channel. The function block could be set to the value, or a timeout has occurred.

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

Parameter:

dChannelHandle: The channel handle

dDevHandle: the device handle

dblValue: The numeric value of the channel for setting

Return Value:

YE_OK: No error

YE_NO_ACCESS_RIGHTS passed unknown Handle: YE_UNKNOWN_HANDLE No permission to set the channel YE_VALUE_NOT_VALID: The value of the channel is outside the permitted limits

YE_TIMEOUT: device does not respond, not set value

4.4.9 SetChannelValueString

The function sets a channel value that the string is passed. The string must be a valid status text value. Numerical values are set using the "Set Channel Value ()". The function blocks until the value is set or a timeout has occurred.

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

Parameter:

"DChannelHandle": the channel Handle

"dDevHandle": the device handle

"CpChanvalstr": the status text that should been set

Return Value:

YE_OK: everything ok

YE_TIMEOUT passed Unknown Handle: YE_UNKNOWN_HANDLE

device has not responded

YE_CHAN_TYPE_MISMATCH: The channel is not a text channel

YE_INVAL_ARGUMENT: The text value is not valid

4.4.10 GetChannelStatTextCnt

If there is a channel status text, this function returns the number of channel text of this channel back. The texts can then then simply "GetChannelStatText (...)" of the function to be queried.

int GetChannelStatTextCnt (DWORD dChannelHandle)

Parameter:

dChannelHandle: the channel handle.

Return Value:

Number of channel text of this channel.

4.4.11 GetChannelStatText

This function returns a particular status text of the channel back. The number of texts should the function GetChannelStatTextCnt (...) are queried before.

i nt GetChannelStatText (DWORD dChannelHandle,
int istat text index, char * Text
Buffer, int buffer size);

Parameter:

"DChanneHandle": The channel Handle

"Istat text index": the index of the queried text.

"TextBuffer" pointer to the memory area in which the text is copied. "Buffer Size":

buffer size

Return Value:

YE_OK: everything OK (result valid)

YE_INVALID_HANDLE: Invalid channel handle was passed.

4.4.12 YasdiMasterGetChannelPropertyDouble

This function SMAData protocol-specific information can be retrieved.

int yasdiMasterGetChannelPropertyDouble (DWORD chan handle,
char * propertyStr, double *
result

Parameter:

Chan handle: The handle channel

PropertyStr: The name of the property to query Result: The retrieved value (double value). Return Value:

YE_OK: ok, copied value

YE_INVAL_ARGUMENT: Unknown property

As properties are currently available:

"Smadata1.cindex" ("Channel Index")

"smadata1.ctype" (channel type)

"Smadata1.ntype" (numerical channel type)

"smadata1.level" (Access Level value)

"smadata1.gain" (gain) "smadata1.offset" channel offset value

Further information about the properties are in the SMAData (1) look up the specification.

4.4.13 GetChannelAccessRights

The function provides more access rights to a channel.

int GetChannelAccessRights (DWORD dchannelHandle,
BYTE * access rights);

Parameter:

"DchannelHandle": The channel Handle

"Access rights": Provides binary-coded access rights of the channel. The following applies:

- CAR_READ: Channel can be read in the current access level
- CAR_WRITE: Channel is writable in the current access level

Return Value:

YE_OK: ok

YE_UNNOWN_HANDLE: The handle was invalid.

4.4.14 GetChannelArraySize

Returns the size of a back channel value, that is, how many values of a single channel value is (array depth).

int GetChannelArraySize (chan handle DWORD, WORD * wpArrayDeep);

Parameter:

channel handle: The handle channel

wpArrayDeep: Reference to result value (WORD)

Return Value:

YE_OK: everything ok

YE_UNNOWN_HANDLE: unknown Handle

4.4.15 GetChannelValRange

The function returns the possible channel value range that is possible for this channel.

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

Parameter:

dChannelHandle: The Channel Handle

min: Reference to double value for receiving the minimum value max: Reference to double value for receiving the maximum value

Return Value:

YE_OK: ok, valid values

YE_UNKNOWN_HANDLE: hand Unknown Handle

-3: There is no dedicated range of values for the channel

4.5 API Using an example

A typical function call order of YASDI API for querying data SMA devices might look like this:

/ * Initialize YASDI ming * /
yasdiMasterInitialize (...)

* Return knows / all interface drivers that YASDI
and unlock the necessary to ... * / yasdiGetDriver (...)
yasdiSetDriverOnline (...)

/ * Find all connected devices (in this case just one) * / DoMasterCmdEx ("detect", 1, NULL, NULL);

/ * Give me all device handles * / GetDeviceHandles
(...)

/ * Give me about this device handles all channel * / GetChannelHandles (...)

Checking / * channel values or set * / While (YOUWANT)

{SetChannelValue (...) or GetChannelValue (...)}

off / * interfaces used again * / yasdiSetDriverOffline (...)

/ * End YASDI * /
yasdiMasterShutdown ()

It is important to ensure that to Begin using the "yasdiMasterInitialize ()" and end "yasdiMasterShutdown ()" in each case to be called once. All other functions of the Master API can be used any number of times in any order.

5 Configuration File

The configuration file used by YASDI uses the familiar Windows INI format. This is also used in the Linux distribution.

The path to the configuration file is passed to the master function "yasdiMasterInitialize (...)" (internal path automatically becomes the "yasdiInitialize (...)" - function passed). The file consists of various sections:

Section: " **Driver Modules** "	
entry	**description**
"Driver0" ... "Driver9	"The used interface driver (eg. B. **yasdi_drv_serial.dll** Windows or **libyasdi_drv_serial.so** Linux) to be loaded at runtime.

Serial Driver Sections: "COMx" (x = COM port number)	
entry	**description**
Device	The file name that corresponds to the serial interface of the operating system. (For the first serial port Windows: "COM1" Linux: "/ dev / ttyS0"
Media	The medium should use the serial driver. currently **Power Line, RS232** and **RS485** supported.
baud rate	The speed of the serial interface in bits per second. Common for Sunny Boys are "1200", and a Sunny Boy Control "19200".
Protocol	The transport protocol to use. The choices are " **SunnyNet** " or " **SMANet** ". Older Sunny Boy Control only use SunnyNet. The newer devices can speak both protocols. It can also be simultaneously used both: **"SMANet, SunnyNet"**

IP driver Section: " IP1 "	
Device0, Device 1, ... DeviceX	Specifying the IP address of the device (Sunny Boy Control with Ethernet Piggy-Back), z. B. Device0 = 192.168.18.1. You can specify any number of addresses, each with its own "DeviceX" entry.
Protocol	As in the serial module (see below)
Master mode	(Optional) The default is "master mode" == 0 "master mode": <> 0: "Slave Mode" section " **Misc** "

entry	**description**
StatisticOutput	File specification (path + file) for issuance of statistical values. It is held to the devices in a file in XML format memory consumption of YASDI and the number of packets sent and received. The entry is optional. The entry is only used for troubleshooting.
DebugOutput	Specifying a file to be saved in the debug information: It can be a file and the information "stdout" or specify "stderr". Section: " **Master**"

entry	**description**
NetAddress	Optional parameter for setting the network address of the SMA data master. The range is from "0" (default) and "65535".
ReadParamChanTimeout	Timeout time in seconds when reading parameter values of the devices. The entry is optional and by default "6".
ReadParamChanRetry	Number of retries when reading parameter channels (after timeout). The entry is optional. Default value: "4" (repetitions).
WriteParamChanTimeout	Optional parameter to specify the timeout writing parameter channels. Default value: "6".

entry	**description**
WriteParamChanRetry	Optional parameter to specify the repetitions in the parameter setting (default "4").
Read Spot Chan timeout	Optional parameter to specify the timeout when reading from spot value channels. Default value: "6"
ReadSpotChanRetry	Number of repetitions when reading spot value channels (by timeout). Entry is optional (Default "4").
DeviceAddrRangeLow	The lower limit of the permitted range network address (device address portion) of a detected device. The entry is optional. The range of values extending from "0" (default) to "255" inclusive. (default), inclusive. Section: **" Master"**
DeviceAddrRangeHigh	The upper limit of the allowable range network address (device address portion) of a detected device. The entry is optional. The range of values extending from "0" to "255" (default), inclusive.
DeviceAddrBusRangeLow	The lower limit of the permitted network address range optional. The range of values extending from "0" to "255" (bus address allotment) of a detected device. The entry is optional. The range of values extending from "0" to "255" (default), inclusive.
DeviceAddrBusRangeHigh	The upper limit of the permitted network address range (bus address allotment) of a detected device. The entry is optional. The range of values extending from "0" to "255" (Address strand portion) of a detected device. The entry is (default), inclusive.
DeviceAddrStringRangeLow	The lower limit of the permitted network address range (Address strand portion) of a detected device. The entry is optional. The range of values extending from "0" to "255" (default), inclusive.
DeviceAddrStringRangeHigh	The upper limit of the permitted network address range

A configuration file on Windows might look like this:

Example:

[Driver Modules]
Driver0 = yasdi_drv_serial.dll

Portion of the first serial interface [COM1]

Device = COM1 Media =
RS232 baud rate = 1200
Protocol = SMANet

[Misc]

[Master]

6 Source - Directory Structures

YASDI uses the following directory structure to briefly two described ben:

subdirectory	description
core	The core of YASDI.
driver	Here are stored all the drivers that supports YASDI. Currently, these are the serial driver and a driver for communication over IP
include	Various include files
libs	The C-API interfaces
master	Implementation of the SMA data master
os	OS abstraction layers: Windows, Linux, MacOSX ....
protocol	Implementations of transport protocols "SMA-Net" and "SunnyNet".
smalib	(Eg read INI file) Various external support modules
projects	Special builds with which YASDI can be assembled in different system. Currently: Windows-lib: makefile generated for Borland C ++ Builder 5 "Generic cmake" makefile for creating YASDI means CMAKE (see www.cmake.org)
shell	A small sample program (command line tool) which uses YASDI.

7 compiling YASDI

YASDI (see www.cmake.org) assembled with the build system "CMAKE". CMAKE created Makefiles for different compilers and operating systems. A partial list is for example the following.:

- GNU Makefile (for UNIX, and MinGW on Windows)
- Microsoft Visual Studio
- Xcode (Mac OS X)

Below compiling YASDI on Linux for GNU Makefile is described in a Bash:

Example:

bash> cd <YASDI-source>

bash> cd builds \ generic-cmake

bash> cmake.

bash> make

bash> # start YasdiShell

bash> export LD_LIBRARY_PATH =.

Bash> ./yasdishell

Disclaimer

Apply The general terms of delivery of SMA Solar Technology AG.

The content of these documents is continually reviewed and adjusted if necessary. Nevertheless, deviations can not be excluded. There is no assurance of completeness. The latest version can be downloaded or purchased through the usual sales channels on the Internet at www.SMA.de.

Warranty and liability claims for damages of any kind are excluded if they are due to one or more of the following:

- transport damage
- Improper or incorrect use of the product
- The product is operated in a non-intended environment
- If the product ignoring the relevant statutory safety at the site
- Failure to observe the warning and safety in all relevant product documentation
- The product is operated under faulty conditions of safety and protection
- Unauthorized modification or repair of the product or the supplied software
- Failure of the product by the action of connected or adjacent devices outside the legal limits
- Disasters and force majeure

Use the supplied software produced by SMA Solar Technology AG is also subject to the following conditions:

- SMA Solar Technology AG rejects any liability for direct or indirect damages arising from the use of software developed by SMA Solar Technology AG, from. This also applies to the provision or non-provision of support activities.
- Supplied software not developed by SMA Solar Technology AG is subject to the respective licensing and liability agreements of the manufacturer.

SMA Factory Warranty

The current guarantee conditions come enclosed with your device. If necessary, you can download www.SMA.de or purchased through the usual sales channels on paper this on the Internet.

trademark

All trademarks are recognized, even if these are not marked separately. A lack of identification does not mean that a product or symbol is free.

SMA Solar Technology AG

Sonnenallee 1 34266

Niestetal Germany

Tel. +49 561 9522-0 Fax +49

561 9522-100 www.SMA.de

Email: 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 Email:

Vertrieb@SMA.de Free Call: 0800

SunnyBoy

Free Call: 0800 78669269