

Handbuch AMB2360 SDK

Version 1.8

AMBER wireless GmbH
Albin-Köbis-Straße 18
51147 Köln
Tel. 02203-6991950
Fax 02203-459883
eMail info@amber-wireless.de
Internet <http://www.amber-wireless.de>

Inhaltsverzeichnis

1	Allgemeines	3
1.1	Inbetriebnahme „Step-by-step“	3
1.2	Bestandteile des SDK's.....	4
1.3	Klassenbeschreibung und Verwendung	4
2	Funktionsbeschreibung.....	5
2.1	Init()	5
2.2	Init()	5
2.3	StartInquiry()	5
2.4	StartInquiry ()	5
2.5	StopInquiry().....	6
2.6	OnDeviceResponded().....	6
2.7	OnInquiryComplete().....	6
2.8	BondQuery().....	7
2.9	Bond().....	7
2.10	CreateConnection().....	7
2.11	OnClientStateChange()	7
2.12	SppReleaseLink().....	8
2.13	Close().....	8
2.14	SendData().....	8
2.15	OnDataReceive()	8
2.16	GetDllVersion().....	8
2.17	GetSdkVersion().....	9
2.18	IsDeviceReady().....	9
2.19	SendLMXCommand()	9
2.20	UnBond().....	10
2.21	ChangeUARTSpeed()	10
3	Ablauf der Funktionsaufrufe.....	11
4	Anhang.....	13
4.1	Tabelle BLUE_NICE_RETURN_CODE.....	13
4.2	Tabelle SPP_STATE_CODE	14
5	Referenzen	14
6	Wichtige Hinweise.....	15
6.1	Haftungsausschluss.....	15
6.2	Warenzeichen	15
6.3	Gebrauchsbeschränkung.....	15

1 Allgemeines

Dieses Software Development Kit (SDK) stellt Funktionen zur Ansteuerung des AMB2360 Bluetooth-Sticks aus einer C++ Bibliothek zur Verfügung.

1.1 Inbetriebnahme „Step-by-step“

1. Treiberinstallation

Vor der ersten Inbetriebnahme des USB-Sticks ist die Installation des Treibers für die virtuelle, serielle Schnittstelle erforderlich.

Dazu muss zunächst das zip-Archiv [1] heruntergeladen, danach auf dem eigenen PC entpackt und die Installationsroutine „CP210x_VCP_Win2K_XP_Vista.exe“ bzw.

„CP210x_VCP_Win7.exe“ per Doppelklick gestartet werden.

Dann den angezeigten Anweisungen folgen, bis die komplette Installation erfolgreich abgeschlossen wurde.

2. Anschluss des USB-Sticks

Der Stick wird zum Anschluss an den PC an einen freien USB-Port eingesteckt.

Windows erkennt nun automatisch das neu angeschlossene Gerät. Die Windowsroutine zur Zuordnung des Treibers wird automatisch gestartet:

Achtung: Falls der Treiber nicht korrekt zugeordnet werden kann lesen Sie bitte [2].

3. Serielle Schnittstelle

Der Treiber erzeugt nun einen virtuellen, seriellen Port. Die Schnittstellenparameter des Sticks sind auf 9600 Baud 8 Datenbits, keine Parität und 1 Stoppbit eingestellt.

Achtung: - Über die DTR-Leitung des virtuellen Ports wird der Hardware-Reset des USB-Sticks gesteuert: Low → Reset; High → Normalbetrieb.

4. Einbinden der Bibliothek

Das Einbinden der Bibliothek in eigene C++ Projekte ist unter Kapitel 1.2 und 1.3 erläutert.

5. Nun kann der AMB2360-Stick mit den Bibliotheksfunktionen gesteuert werden.

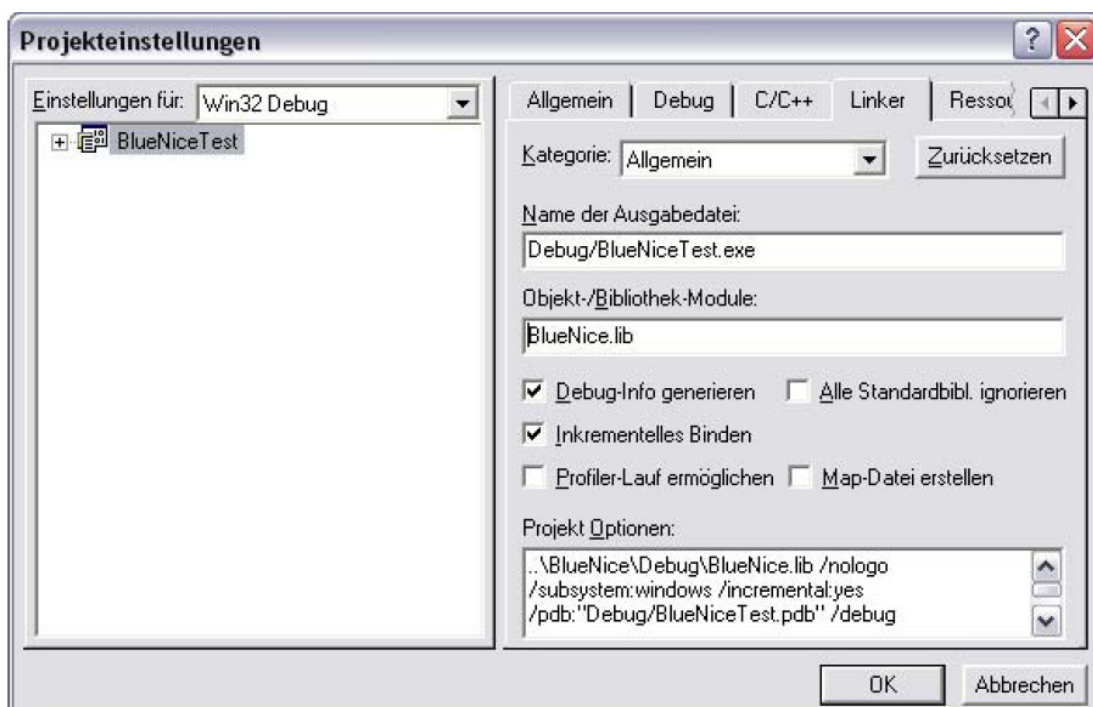
**Achtung: - Die Bibliothek kann nur in Projekten eingebunden werden, die mit Microsoft Visual C++ erstellt wurden.
- Der PIN für das Pairing über Bluetooth ist auf 1234 eingestellt.**

1.2 Bestandteile des SDK's

Das SDK besteht aus einer DLL, der dazugehörigen Library Datei und zwei Header Dateien.

- BlueNice.dll
- BlueNice.lib
- BlueNice.h
- BlueNiceDefinitions.h

Zur Verwendung in einem MFC Projekt sind folgende Einstellungen notwendig:



1.3 Klassenbeschreibung und Verwendung

Die Klasse CBlueNice beinhaltet eine Schnittstelle zur Suche von Bluetoothgeräten in Reichweite und die Verbindung zum Dienst SerialPortProfile (SPP) des Gerätes.

Zur Verwendung der Funktionen ist eine Klasse von CBlueNice abzuleiten.

```
class CBlueNiceTestDlg : public CDialog, public CBlueNice
{
}

```

2 Funktionsbeschreibung

2.1 Init()

Diese Funktion initialisiert und öffnet den Serialport und prüft ob ein AMB2360 vorhanden ist.

Prototype: `BLUE_NICE_RETURN_CODE Init(int nPortNumber)`

Parameters: `nPortNumber` Nummer des Serialports, der geöffnet werden soll
z.B. `Init(9)`; zum Öffnen von COM9

Returns: `BLUE_NICE_RETURN_CODEs` (siehe Tabelle)

2.2 Init()

Diese Funktion initialisiert und öffnet den Serialport und prüft ob ein AMB2360 vorhanden ist.

Prototype: `BLUE_NICE_RETURN_CODE Init(int nPortNumber, EBaudrate Baudrate)`

Parameters: `nPortNumber` Nummer des Serialports, der geöffnet werden soll
`Baudrate` Baudrate des entsprechenden Ports

Returns: `BLUE_NICE_RETURN_CODEs` (siehe Tabelle)

2.3 StartInquiry()

Startet die Suche nach Geräten in der Umgebung. Während die Funktion Inquiry läuft, wird die virtuelle Funktion OnDeviceResponded für jedes gefundene Gerät aufgerufen. Dies kann dazu genutzt werden, eine Liste von gefundenen Geräten zu erstellen.

Prototype: `BOOL StartInquiry()`

Parameters: `None`

Returns: `TRUE` → Inquiry gestartet
`FALSE` → Inquiry Kommando nicht gesendet

2.4 StartInquiry ()

Startet die Suche nach Geräte in der Umgebung. Während die Funktion Inquiry läuft, wird die virtuelle Funktion OnDeviceResponded für jedes gefundene Gerät aufgerufen. Dies kann dazu genutzt werden, eine Liste von gefundenen Geräten zu erstellen.

Prototype `BOOL StartInquiry(BYTE bTime)`

Parameters: `bTime` → Bereich von 0x01 bis 0x30, wobei 0x01 = 1,28s
Werte außerhalb des Bereichs werden auf 0x10 gesetzt

Returns: `TRUE` → Inquiry gestartet
`FALSE` → Inquiry Kommando nicht gesendet

2.5 StopInquiry()

Hat keinerlei Funktion. Ist nur der Kompatibilität halber implementiert.

Prototype: `void StopInquiry()`

Parameters: None

Returns: void

2.6 OnDeviceResponded()

Diese virtuelle Funktion wird für jedes Gerät aufgerufen, das in der Bluetooth Umgebung gefunden wurde.

Prototype: `virtual void OnDeviceResponded(BD_ADDR bda, DEV_CLASS devClass, BD_NAME bdName, BOOL bConnected)`

Parameters: `bda` → Die Adresse des gefundenen Gerätes

`devClass` → Klasse des gefundenen Gerätes

-`devClass[0]` in Verbindung mit den höchste 3 Bits aus `devClass[1]` ergibt die Service – Klasse

-`devClass[1]` die unteren 5 Bits ergeben die Major-Geräte-Klasse

-`devClass[2]` die unteren 6 Bits ergeben die Minor-Geräte-Klasse

`bdName` Name des Gerätes als Null-terminierter String, hat die Länge Null, wenn das Gerät nur die Adresse zurück gibt

`bConnected` TRUE → wenn das Gerät mit dem lokale Gerät verbunden ist

Returns: void

2.7 OnInquiryComplete()

Diese virtuelle Funktion wird aufgerufen, wenn das Suchen nach Bluetoothgeräten beendet ist. Die Anzahl aller gefundenen Geräte wird übergeben.

Prototype: `virtual void OnInquiryComplete (BOOL success, short num_responses)`

Parameters: `success` TRUE → wenn Inquiry erfolgreich war, sonst gibt es einen Gerätefehler

`num_responses` Anzahl der gefundenen Geräte

Returns: void

2.8 BondQuery()

Diese Funktion prüft, ob zwischen dem gewünschten Gerät und dem lokalen Gerät eine erfolgreiche paarweise Verbindung bestand.

Prototype: `BOOL BondQuery(BD_ADDR bda)`
Parameters: `bda` Adresse des Gerätes, das verbunden werden soll
Returns: `TRUE` → wenn bereits eine paarweise Verbindung bestand
`FALSE` → wenn das Gerät nicht paarweise verbunden ist

2.9 Bond()

Diese Funktion löst das Pairing der beiden Geräte aus.

Prototype: `BOOL Bond(BD_ADDR bda, BT_CHAR *pin_code, BYTE pin_length)`
Parameters: `bda` Adresse des zu bindenden Gerätes
`pin_code` PIN Code der zum Verbinden benutzt werden soll
`pin_length` Länge des Pin Codes
Returns: `TRUE` → Pairing erfolgreich, sonst `FALSE`

2.10 CreateConnection()

Diese Funktion stellt eine Verbindung zum Dienst SerialPortProfile (SPP) des Endgerätes her. Nach dem Aufruf der Funktion erhält die Applikation einen Event Callback der den Status der Verbindung mit Hilfe der Funktion `OnClientStateChange` übermittelt.

Prototype: `void CreateConnection(BD_ADDR bda)`
Parameters: `bda` Adresse des Gerätes, mit dem eine Serial-Verbindung hergestellt werden soll
Returns: `void`

2.11 OnClientStateChange()

Diese virtuelle Funktion informiert über den Zustand der Verbindung zum SPP Server.

Prototype: `virtual void OnClientStateChange(BD_ADDR bda, SPP_STATE_CODE state)`
Parameters: `bda` Adresse des SPP Servers
`state` Status der Verbindung (siehe Tabelle)
Returns: `void`

2.12 SppReleaseLink()

Die SPP-Verbindung wird aufgehoben. Der Serialport steht nun für weitere Verbindungen zur Verfügung.

Prototype: `void SppReleaseLink()`
Parameters: `None`
Returns: `void`

2.13 Close()

Schließt den Serialport, wenn die SPP-Verbindung getrennt wurde.

Prototype: `BOOL Close()`
Parameters: `None`
Returns: `TRUE` → Serialport geschlossen
 `FALSE` → SPP – Verbindung besteht noch, Serialport kann nicht geschlossen werden. Zuerst `SppReleaseLink()` ausführen

2.14 SendData()

Wenn die Verbindung aufgebaut ist, kann diese Funktion benutzt werden um Daten im Transparent-Mode zu senden.

Prototype: `void SendData(BYTE *pData, int nLength)`
Parameters: `BYTE *pData` Zeiger auf Daten, die übertragen werden sollen
 `Int nLength` Anzahl der Zeichen
Returns: `void`

2.15 OnDataReceive()

Es sind Daten empfangen worden, die durch diese virtuelle Funktion an die Anwendung übergeben werden.

Prototype: `virtual void OnDataReceive(BYTE *pData, DWORD dwRead)`
Parameters: `pData` Zeiger auf die empfangenen Zeichen
 `dwRead` Anzahl der empfangenen Zeichen
Returns: `void`

2.16 GetDllVersion()

Liefert die Version der DLL

Prototype: `BOOL GetDllVersion(BT_CHAR *p_version_info, int nSize)`

Parameters: p_version_info Zeiger auf die Versionspuffer
 nSize Wie viel Zeichen stellt der Puffer zur Verfügung

Returns: TRUE → Version konnte in Puffer eingetragen werden
 FALSE → Version konnte nicht in Puffer eingetragen werden, da der
 Puffer zu klein war.

2.17 GetSdkVersion()

Liefert die Version des SDK's

Prototype: Bool GetSdkVersion(BT_CHAR *p_version_info, int nSize)

Parameters: p_version_info Zeiger auf die Versionspuffer
 nSize Wie viel Zeichen stellt der Puffer zur Verfügung

Returns: TRUE → Version konnte in Puffer eingetragen werden
 FALSE → Version konnte nicht in Puffer eingetragen werden, da der
 Puffer zu klein war.

2.18 IsDeviceReady()

Funktion zum Prüfen der Schnittstelle

Prototype: Bool IsDeviceReady()

Parameters: None

Returns: TRUE → Schnittstelle OK
 FALSE → Schnittstelle gestört

2.19 SendLMXCommand()

Schnittstellenfunktion zum Senden von LMX-Kommandos. Diese Funktion kann nach dem Init() des Moduls aufgerufen werden. Befindet sich das Modul im Transparent-Modus, kann die Funktion keine Kommandos senden.

Daten werden mit der Funktion OnDataReceive() empfangen.

Prototype: BOOL SendLMXCommand(BYTE opcode,WORD length,BYTE *pData)

Parameters: opcode Kommando Opcode
 Length Datenlänge
 pData Zeiger auf die Daten

Returns: TRUE → Kommando kann gesendet werden
 FALSE → Kommando kann nicht gesendet werden

2.20 UnBond()

Diese Funktion löscht die Paarung zwischen den Geräten.

Prototype: BOOL UnBond(BD_ADDR bda)

Parameters: bda Adresse des Geräts, mit dem die Paarung aufgehoben werden soll

Returns: TRUE → Paarung aufgehoben

 FALSE → Kommando fehlgeschlagen, oder es besteht keine Paarung

2.21 ChangeUARTSpeed()

Funktion zum Ändern der Baudrate.

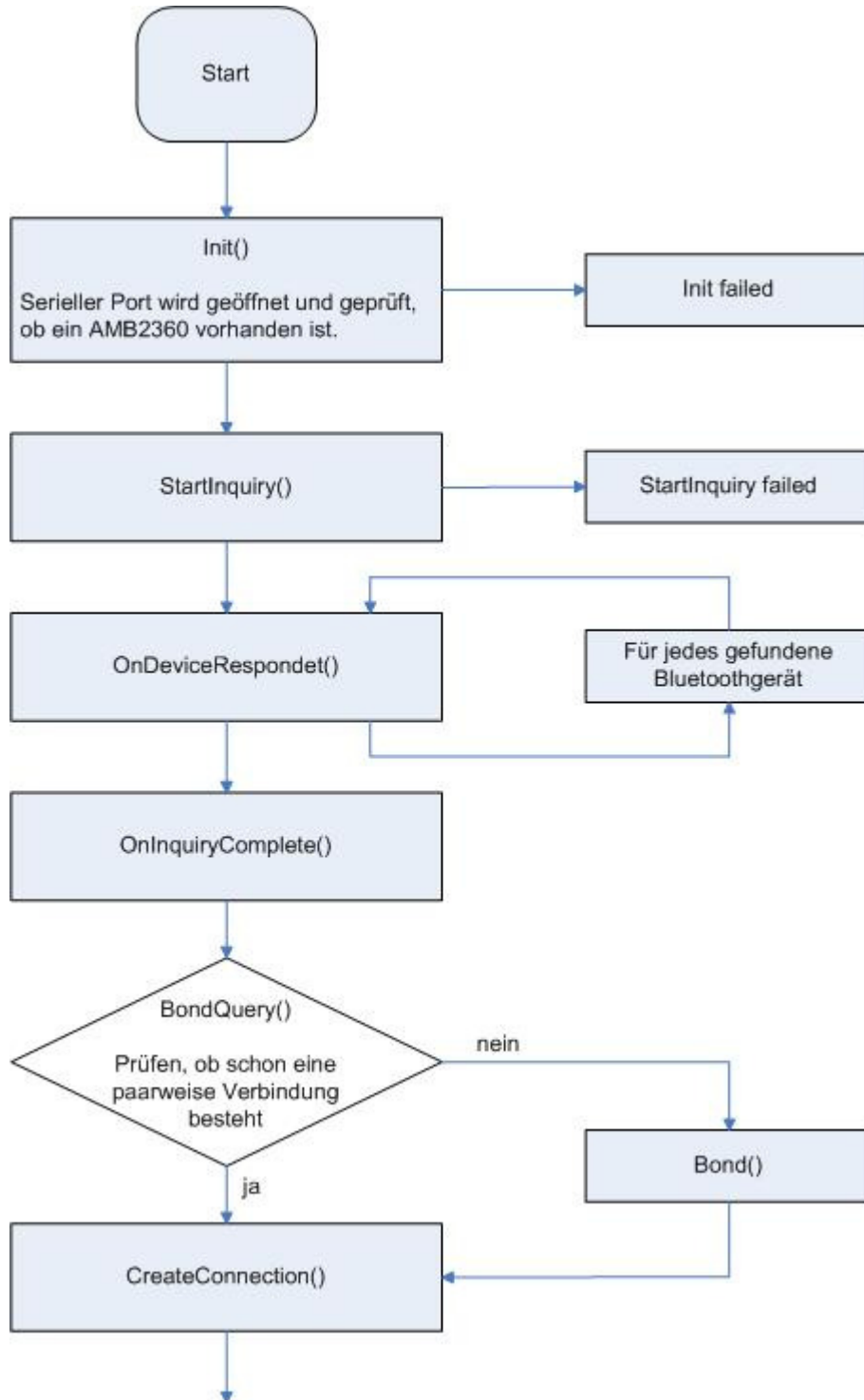
Prototype: BOOL ChangeUARTSpeed(EBaudrate Baudrate)

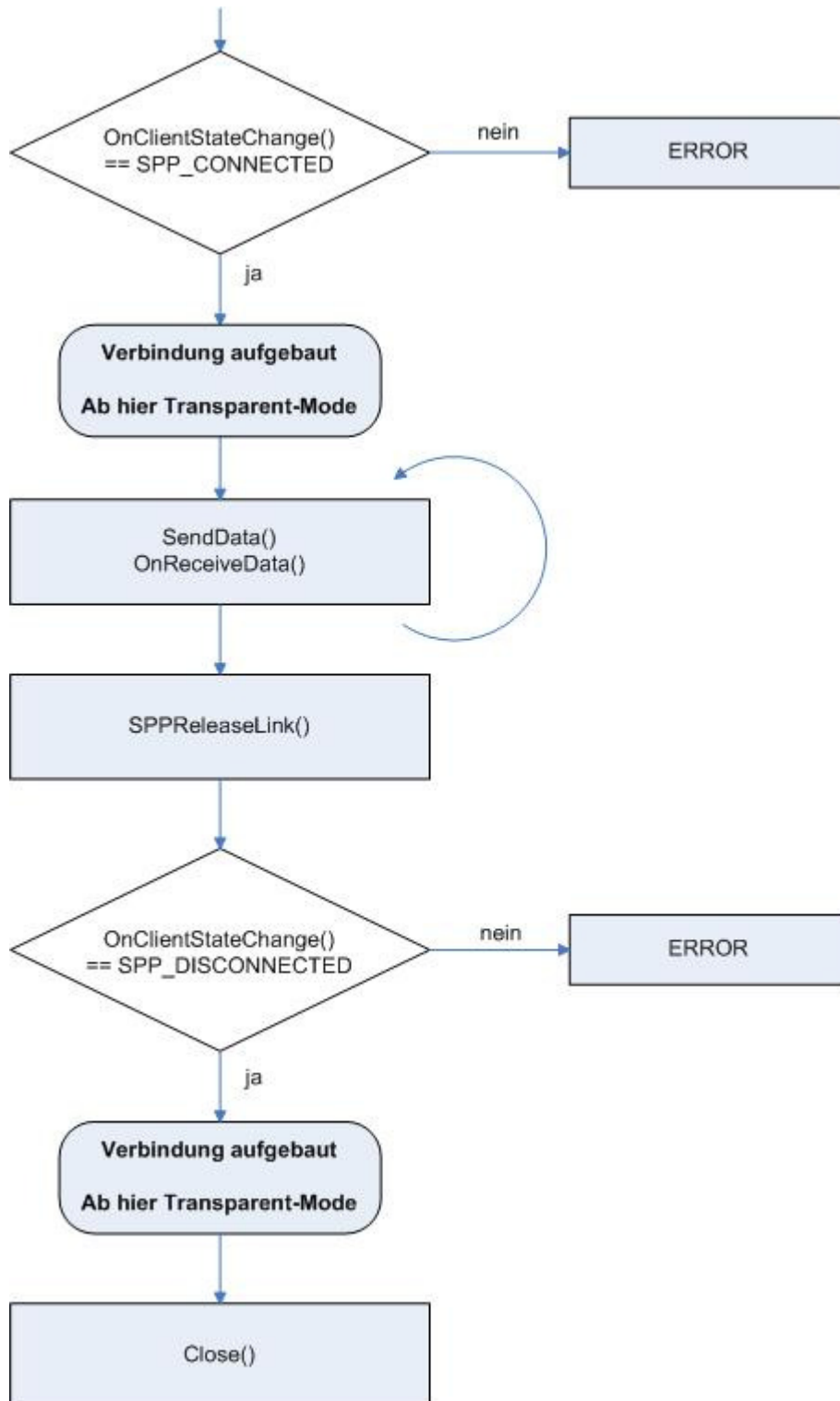
Parameters: Baudrate Baudrate, mit welcher die Schnittstelle initialisiert werden soll.

Returns: TRUE → Baudrate akzeptiert, RESET des Moduls notwendig. Dies wird dem
 User über die Funktion OnClientStateChange() mitgeteilt.

 FALSE → Kommando fehlgeschlagen

3 Ablauf der Funktionsaufrufe





4 Anhang

4.1 Tabelle BLUE_NICE_RETURN_CODE

Rückgabewert	Beschreibung
SUCCESS	Kein Fehler
NO_BT_SERVER	COM Server kann nicht gestartet werden
ALREADY_CONNECTED	Kann nicht verbunden werden, wenn die vorherige Verbindung nicht geschlossen wird
NOT_CONNECTED	Versuch eine Verbindung die nicht besteht zu schließen
NOT_ENOUGH_MEMORY	Lokaler Prozessor kann keinen Speicher reservieren
INVALID_PARAMETER	Ungültige Parameter
UNKNOWN_ERROR	Nicht bekannter Fehler
NO_EMPTY_PORT	
LICENSE_ERROR	Lizenz Fehler
NO_BT_MODUL	Kein Bluetooth-Modul vorhanden
COM_PORT_ERROR_OPEN	SerialPort kann nicht initialisiert werden
COM_PORT_UNAVAILABLE	SerialPort nicht verfügbar
COM_PORT_ERROR_SETTINGS	SerialPort Einstellungen können nicht gemacht werden
COM_PORT_ERROR_HANDSHAKE	SerialPort Handshake kann nicht eingestellt werden
COM_PORT_ERROR_TIMEOUT	SerialPort TimeOut definieren
CMD_EVENT_ERROR	Fehlermeldung für Warte-Event

4.2 Tabelle SPP_STATE_CODE

Rückgabewert	Beschreibung
SPP_CONNECTED	Verbindung hergestellt
SPP_DISCONNECTED	Verbindung aufgehoben
SPP_RFCOMM_CONNECTION_FAILED	RFCOMM Verbindung fehlgeschlagen
SPP_PORT_IN_USE	Port belegt
SPP_PORT_NOT_CONFIGURED	Port nicht Konfiguriert
SPP_SERVICE_NOT_FOUND	Dienst nicht gefunden
SPP_ALLOC_SCN_FAILED	
SPP_SDP_FULL	

5 Referenzen

- [1] Treiber für USB-Wandler Silabs CP2102
 Download auf <http://amber-wireless.de> → Support → Download → Auswahl „AMB2560“ bzw. „AMB8460“ → Silabs USB-Driver Win2K, XP, Vista / Silabs USB-Driver Win7
- [2] Download auf <http://amber-wireless.de> → Support → Download → Auswahl „AMB2560“ bzw. „AMB8460“ → „AN04“

6 Wichtige Hinweise

6.1 Haftungsausschluss

AMBER wireless GmbH geht davon aus, dass die hierin befindlichen Angaben zum Zeitpunkt der Veröffentlichung zutreffend sind. AMBER wireless GmbH behält sich jedoch das Recht vor, technische Spezifikationen oder Funktionen der eigenen Produkte zu ändern, die Herstellung dieser Produkte oder den Support für eines dieser Produkte einzustellen, ohne dass es einer schriftlichen Ankündigung oder Mitteilung der Kunden bedarf. Der Kunde hat sicherzustellen, dass die ihm zur Verfügung stehenden Informationen gültig sind. AMBER wireless GmbH übernimmt keinerlei Haftung für den Gebrauch ihrer Produkte. Amber wireless GmbH erteilt weder Lizenzen an ihren Patentrechten, noch an anderen Rechten an ihrem geistigen Eigentum oder an Rechten Dritter. Der Kunde ist dafür verantwortlich, dass sein System oder seine Einheit, in das die AMBER wireless Produkte integriert wurden, den entsprechenden gesetzlichen Bestimmungen entspricht.

6.2 Warenzeichen

AMBER wireless® ist ein eingetragenes Warenzeichen der AMBER wireless GmbH.

Alle anderen Warenzeichen, eingetragene Warenzeichen und Produktnamen sind das ausschließliche Eigentum der jeweils Berechtigten.

6.3 Gebrauchsbeschränkung

AMBER wireless Produkte sind nicht freigegeben für den Gebrauch in lebensunterstützenden oder lebenserhaltenden Systemen oder Einheiten, oder anderen Systemen, bei den davon ausgegangen werden kann, dass eine Fehlfunktion zu einem wesentlichen Personenschaden beim Nutzer führt. AMBER wireless Produkte sind weiterhin nicht freigegeben für den Gebrauch als wesentlicher Bestandteil jeglichen(r) lebensunterstützenden(r) oder lebenserhaltenden(r) Systems oder Einheit, dessen/deren Fehlfunktion zum Ausfall des/der lebensunterstützenden oder lebenserhaltenden Systems oder Einheit führen kann, oder dessen/deren Sicherheit oder Effektivität beeinflusst werden kann. AMBER wireless Kunden, die diese Produkte in solchen Applikationen verwenden oder sie für solche Verwendungen verkaufen, handeln auf eigenes Risiko und müssen AMBER wireless GmbH von allen Schäden freistellen, die durch den Verkauf zu ungeeigneten Zwecken oder die ungeeignete Verwendung entstehen.

Durch die Verwendung von AMBER wireless Produkten ist der Nutzer an diese Bedingungen gebunden.

© 2009, AMBER wireless GmbH. Alle Rechte vorbehalten.

AMBER wireless GmbH
Albin-Köbis-Straße 18
51147 Köln
Tel. +49 (0) 2203-6991950
Fax +49 (0) 2203-459883
eMail info@amber-wireless.de
Internet <http://www.amber-wireless.de>