Referenz

# Bluetooth with NXT

**Referenz zur Kommunikation mit Bluetooth zwischen mehreren NXT Prozessoren basierend auf NXC**

**Veröffentlicht von:** Martin Stypinski <martisty@ee.ethz.ch>
Zelglistrasse 42
8122 Binz

# Inhaltsverzeichnis

# 1 Abstracts

In dieser Referenz werden alle wichtigen Informationen zum NXT und der Bluetoothkommunikation erläutert. Die Referenz basiert auf der Firmware 1.05 und der BricX Command Center Version 3.3.7.17.
Diese Refrenz umfasst alle HighLevel Befehle und ist somit für die Verwendung mit BricX CC gedacht.

# 2 NXT Hardwareaufbau

Die Kommunikation zwischen mehreren NXT Prozessoren über Bluetooth erfolgt auf der HighLevel Ebene auf einem sehr einfachen Prinzip. Es können grundsätzlich beliebig viele NXT Prozessoren miteinander kommunizieren. Dabei ist jedoch zu beachten, dass ein Prozessor den Master übernimmt. Die restlichen Prozessoren übernehmen hierbei die Slaves. Jede Kommunikation zwischen den Slaves ist unmöglich. Die Slaves können jeweils nur mit dem Master interagieren und der Master mit den Slaves.

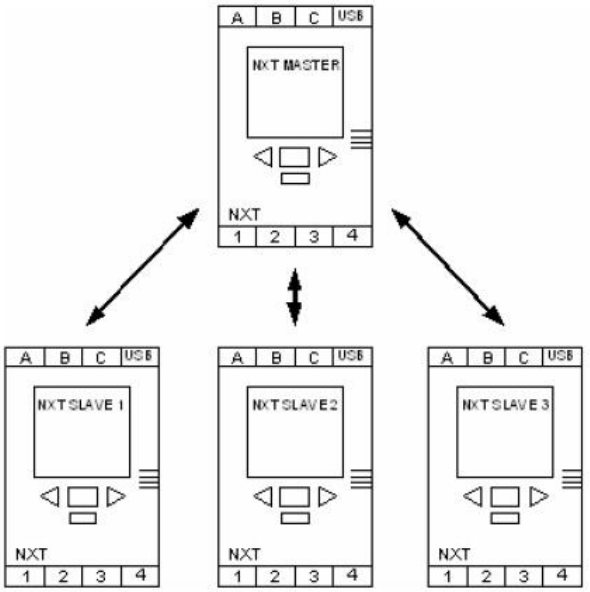


Abbildung 1: Bluetooth Netzwerk

Am Beispiel aus Abb. 1 kann man unschwer erkennen, wie die Kommunikation aufgebaut ist.
Weiter kommt dazu, dass jeder NXT eine Art In und Outbox hat für Nachrichten die per Bluetooth übertragen werden. Die Outbox des Mastes muss jeweils mit der Inbox des Slaves und umgekehrt harmonieren. Genau so wie die Connection. Die Connection wird jeweils mit einer Nummer definiert. Der Master hat die Nummer 0.

# 3 NXC Programmierung

## 3.1 Programmierbeispiel

### 3.1.1 Master

```
//MASTER
#include "NXCDefs.h"

#define BT_CONN 1
#define OUTBOX 5
#define INBOX 1

sub BTCheck(int conn){
   if (!BluetoothStatus(conn)==NO_ERR){
      TextOut(5,LCD_LINE2,false,"Error");
      Wait(1000);
      Stop(true);
   }
}

task main(){
   int ack, i;
   BTCheck(BT_CONN);
   TextOut(10,LCD_LINE1,false,"Master sending");
   while(true){
     i = Random(512);
     TextOut(0,LCD_LINE3,false,"                    ");
     NumOut(5,LCD_LINE3,false,i);
     ack = 0;
     SendRemoteNumber(BT_CONN,OUTBOX,i);
     until(ack==0xFF) {
       until(ReceiveRemoteNumber(INBOX,true,ack) == NO_ERR);
     }
     Wait(250);
   }
}
```

### 3.1.2 Slave

```
//SLAVE
#include "NXCDefs.h"

#define BT_CONN 1
#define OUT_MBOX 1
#define IN_MBOX 5

sub BTCheck(int conn){
   if (!BluetoothStatus(conn)==NO_ERR){
      TextOut(5,LCD_LINE2,false,"Error");
      Wait(1000);
      Stop(true);
   }
}

task main(){
   int in;
   BTCheck(0);
   TextOut(5,LCD_LINE1,false,"Slave receiving");
   SendResponseNumber(OUT_MBOX,0xFF); //unblock master
   while(true){
     if (ReceiveRemoteNumber(IN_MBOX,true,in) != STAT_MSG_EMPTY_MAILBOX) {
       TextOut(0,LCD_LINE3,false,"                    ");
       NumOut(5,LCD_LINE3,false,in);
       SendResponseNumber(OUT_MBOX,0xFF);
     }
     Wait(10); //take breath (optional)
   }
}
```

## 3.2 Referenz

### 3.2.1 SendRemoteBool

**SendRemoteBool(connection, queue, bool_value)**
Die Funktion überträgt einen boolschen Wert an den NXT mit der Angegebenen „connection“ und der Wert wird in die Warteschlange („queue“) auf dem Slave abgelegt.

### 3.2.2 SendRemoteNumber

**SendRemoteNumber(connection, queue, value)**
Die Funktion überträgt einen numerischen Wert an den NXT mit der Angegebenen „connection“ und der Wert wird in die Warteschlange („queue“) auf dem Slave abgelegt.

### 3.2.3 SendRemoteString

**SendRemoteString(connection, queue, value)**
Die Funktion überträgt einen String an den NXT mit der Angegebenen „connection“ und der Wert wird in die Warteschlange („queue“) auf dem Slave abgelegt.

### 3.2.4 SendResponseBool

**SendResponseBool(queue, bool_value)**
Die Funktion überträgt einen boolschen Wert als Antwort auf eine empfangene Nachricht. Der Wert wird in die Warteschlange („queue“) (+10) des Slaves gelegt und kan über automatisches Abfragen des Masterbricks übertragen werden.

### 3.2.5 SendResponseNumber

**SendResponseNumber(queue, value)**
Die Funktion überträgt einen numerischen Wert als Antwort auf eine empfangene Nachricht. Der Wert wird in die Warteschlange („queue“) (+10) des Slaves gelegt und kan über automatisches Abfragen des Masterbricks übertragen werden.

### 3.2.6 SendResponseBool

**SendResponseBool(queue, value)**
Die Funktion überträgt einen String als Antwort auf eine empfangene Nachricht. Der Wert wird in die Warteschlange („queue“) (+10) des Slaves gelegt und kan über automatisches Abfragen des Masterbricks übertragen werden.

### 3.2.7 ReceiveRemoteBool

**ReceiveRemoteBool(queue, bool_remove, bool_value)**
Die Funktion wird auf dem Masterbrick aufgerufen um einen boolschen Wert aus der Warteschlange eines Slaves zu erhalten. Mit dem bool_remove Bool, können übertragene Werte aus der Warteschlange gelöst werden.

### 3.2.8 ReceiveRemoteNumber

**ReceiveRemoteNumber(queue, bool_remove, value)**
Die Funktion wird auf dem Masterbrick aufgerufen um einen numerischen Wert aus der Warteschlange eines Slaves zu erhalten. Mit dem bool_remove Bool, können übertragene Werte aus der Warteschleife gelöst werden.

### 3.2.9 ReceiveRemoteString

**ReceiveRemoteString(queue, bool_remove, value)**
Die Funktion wird auf dem Masterbrick aufgerufen um einen String aus der Warteschlange eines Slaves zu erhalten. Mit dem bool_remove Bool, können übertragene Werte aus der Warteschleife gelöst werden.

### 3.2.10 ReceiveRemoteMessageEx

**ReceiveRemoteMessageEx(queue, bool_remove, out strval, out val, out bool_val)**
Die Funktion wird auf dem Masterbrick aufgerufen um einen unbestimmten Wert aus der Warteschlange eines Slaves zu erhalten. Mit dem bool_remove Bool, können übertragene Werte aus der Warteschleife gelöst werden.

### 3.2.11 SendMessage

**SendMessage(queue, msg)**
Diese Funktion schreibt den Inhalt des Messagebuffers in die angegebene Mailbox oder Warteschlange („queue“). Die maximale Länge darf dabei 58bytes nicht überschreiten.

### 3.2.12 RecieveMessage

**RecieveMessage(queue, bool_remove, out buffer)**
Diese Funktion empfängt den Inhalt des Messagebuffers und schreibt ihn in einen Buffer. Mit bool_remove kann die erhaltene Nachricht auf dem Slave gelöst werden.

### 3.2.13 BluetoothStatus

**BluetoothStatus(connection)**
Diese Funktion überprüft den Status einer Bluetooth Verbindung

#### 3.2.14 BluetoothWrite

**BluetoothWrite(connection, buffer)**
Diese Funktion überträgt auf Firmwareebene direkt Daten und schreibt sie in den Buffer des verbundenen NXT's.

#### 3.2.15 RemoteMessageRead

**RemoteMessageRead(connection, queue)**
Diese Funktion schreibt den Inhalt des Messagebuffers in die angegebene Mailbox oder Warteschlange („queue"). Die maximale Länge darf dabei 58bytes nicht überschreiten.

#### 3.2.16 RemoteMessageWrite

**RemoteMessageWrite(connection, queue, msg)**
Diese Funktion sendet einen direkten Befehl an den Slavebrick über die angegebene „connection". Mit BluetoothStatus kann der aktuelle Zustand der Übertragung abgefragt werden.

#### 3.2.17 RemoteStartProgram

**RemoteStartProgram(connection, filename)**
Mit dieser Funktion können direkt Programme auf einem Slave aufgerufen werden. Mit BluetoothStatus kann der Zustand abgefragt werden.

#### 3.2.18 RemoteStopPrgram

**RemoteStopProgram(connection)**
Diese Funktion kann ein Programm auf dem Slave NXT beenden. mit BluetoothStatus kann wiederum der Zustand abgefragt werden.

#### 3.2.19 RemotePlaySoundFile

**RemotePlaySoundFile(connection, filename, bool_loop)**
Mit dieser Funktion können direkt Soundfiles auf einem Slavebrick abgsepielt werden. Bool_loop ist hierbei ein Boolean der angibt ob sich das File loopen soll oder nicht. Mit BluetoothStatus kann der Zustnad zurückgegeben werden.

#### 3.2.20 RemotePlaySoundFile

**RemotePlayTone(connection, freq, lenght)**
Die Funktion kann auf dem Slave NXT direkt einen Ton ausgeben. Die Parameter freq und lenght bestimmen hierbei die Frequenz und Länge. Mit BluetoothStatus können die Zustände ausgegeben werden.

### 3.2.21 RemoteStopSound

**RemoteStopSound(connection)**
Diese Funktion beendet die Soundausgabe am Slave NXT. Mit der Funktion BluetoothStatus kann der Zustand direkt ausgegeben werden.

### 3.2.22 RemoteKeepAlive

**RemoteKeepAlive(connection)**
Diese Funktion sendet ein Datenpaket an einen Slave. Das Datenpaket hält die Verbindung aufrecht. Mit BluetoothStatus kann der Status ausgegeben werden.