Doayee.

MICROCHIP RN52 LIBRARY

COMMAND GUIDE

Version 1.00

For additional support, please visit doayee.co.uk

Contents

RN52 Library adapted from the SoftwareSerial library with added functionality to interface with the RN52.

SoftwareSerial library by ladyada, Mikal Hart, Paul Stoffregen, Garrett Mace, and Brett Hagman.

RN52 Library written by Thomas Cousins for http://doayee.co.uk

Visit http://doayee.co.uk/bal/library for all related downloads.

RN52 Library Command Guide

Setup commands:

#include <RN52.h>

Whenever you are using the RN52 library, you must include it at the top of your sketch. This allows the IDE to recognise commands from the library.

RN52 rn52(RX, TX);

To begin using the library you must first define the class, in this case 'rn52', but you could call it anything. The pins you are connecting to the RN52 with must be specified in the RX and TX fields.

The library is based off the SoftwareSerial library, and the same limitations apply when using it:

- If using multiple software serial ports, only one can receive data at a time.
- Not all pins on the Mega and Mega 2560 support change interrupts, so only the following can be used for RX: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69).
- Not all pins on the Leonardo and Micro support change interrupts, so only the following can be used for RX: 8, 9, 10, 11, 14 (MISO), 15 (SCK), 16 (MOSI).

Eg. RN52 rn52(10,11);   //Set RX to 10 and TX to 11

From now on all commands must start with your class name, then a dot. This links the command to the library, and without it the sketch will not be able to compile.

begin(baud rate);

To start communication with the RN52, you must use the *begin* command. Within the begin command you must specify the baud rate you are communicating at. We recommend using a baud rate of 38400 or below, as above this the library can be temperamental due to the interrupt service routines onboard the AVR micro-controllers. If you are using the library with a BAL then the RN52 will already be set to communicate at 38400, but if not you will have to set this yourself.

Eg. rn52.begin(38400);   //Begin communication with the RN52 at 38400baud

General Commands:

reboot();

Eg. rn52.reboot(); *//Reboots the RN52.*

setDiscoverability(discoverability state);

Eg. rn52.setDiscoverability(1); *//Sets the module discoverable*

toggleEcho();

Command used to toggle the echo feature of the RN52, which simply sends back every character you send it, giving you a real time view of what you are sending to it. Useful for debugging code.

Eg. rn52.toggleEcho();

factoryReset();

Does what it says on the tin, requires a restart after use.

Eg. rn52.factoryReset()

idlePowerDownTime(Optional: new time);

The idle power down time is the time it takes for the RN52 to power down when it is receiving no input. By default it is set to 0, which means it will never power down.

Eg. rn52.idlePowerDownTime(30); *//Sets the idle power down time to 30 secs*

Eg. int i = rn52.idlePowerDownTime(); *//reads the idle power down time into the integer i.*

name(Optional: “New Name”);

Command used to either set, or read the name of the RN52. If the “New Name” field is left blank it will return the current name of the RN52 as a String.

Eg. rn52.name(“BAL-New”) *//Sets the name to BAL-New*

Eg. String currentName = rn52.name(); *//Sets the string currentName to the current name of the RN52*

volumeOnStartup(Optional: New Volume);

Command used to either set, or read the volume level on the RN52 start-up. If the New Volume field is left blank it will return the volume as an int. Volume is between 1 and 15.

Eg. rn52.volumeOnStartup(10); *//Sets the volume on start-up to 10.*

Eg. int i = rn52.volumeOnStartup(); *//Sets the integer i to the current volume on start-up.*

println(data to send);

Command used to send the raw ascii data input to the RN52, as you would if you were using a command line interface.

Eg. rn52.println("v"); *//Request RN52 firmware version*

print("data to send);

Command used to send the raw ascii data input to the RN52, as you would if you were using a command line interface. Does not include the newline character which is necessary for the RN52 to accept the command. For this use println (see above).

Eg. rn52.print("v"); *//send "v" character to RN52*

available();

Command used to return the number of bytes of data in the input buffer sent from the RN52.

Eg. int i = rn52.avaliable(); *//sets an integer i to the number of bytes sent from the RN52 since data was last read ie. After a command is sent the RN52 responds with "AOK" which in this case would set i to 3 (one byte per character)*

Audio Commands:

Most of these commands are fairly self-explanatory.

volumeUp();	*Eg. rn52.volumeUp();*
volumeDown();	*Eg. rn52.volumeDown();*
playPause();	*Eg. rn52.playPause();*
nextTrack();	*Eg. rn52.nextTrack();*
prevTrack();	*Eg. rn52.prevTrack();*
trackTitle();	*Eg. String title = rn52.trackTitle();*
album();	*Eg. String album = rn52.album();*
artist();	*Eg. String artist = rn52.artist();*
genre();	*Eg. String genre = rn52.genre();*

getMetaData();

Outputs the entire available MetaData as one long string. This may vary depending upon the audio source capabilities. It may include genre or track length, in addition to title, album and artist.

Eg. String data = rn52.getMetaData();

GPIO Commands:

Only GPIO pins 5, 6, 10, 11, 12, and 13 are available for use as GPIO on the BAL. The others have separate reserved functions. Changing the mode on any pins other than these will have no effect on the RN52, as the library prohibits it.

GPIOPinMode(pin, state);

This command is used to set the pinMode of the desired pin. It works exactly like the normal pinMode command used by the Arduino IDE. The pin field specifies which pin you are declaring as an input or output, and the state field specifies which state it should be.

Eg. rn52.GPIOPinMode(12, 1); //Sets pin 12 to an output

GPIODigitalWrite(pin, state);

This command is used to set an output as logic high or logic low, or to enable or disable the internal pull-up resistors on an input. It works exactly like the normal digitalWrite command used by the Arduino IDE. The pin field specifies which pin you want to change, and the state field specifies which state to change it to.

Eg. rn52.GPIODigitalWrite(12, 0); //Writes pin 12 low.

GPIODigitalRead(pin);

This command reads back the state of any pin, output or input. It is a boolean function, so it returns either a 1 or a 0 depending on the state of the pin (1 for HIGH, 0 for LOW).

Eg. while(rn52.GPIODigitalRead(11)); //Waits while pin 11 is high

Extended Features:

These commands all alter the extended features of the RN52. They all accept either a Boolean input (1 or 0 – 1 for true, 0 for false), or read back the current state of that particular feature.

AVRCPButtons();

This setting converts certain GPIO pins into dedicated buttons for controlling volume and tracks, as well as a play/pause button. Buttons should be connected from the module to ground. Internal pull-up resistors are present in the RN52.

GPIO10	Volume Down
GPIO11	Previous Track
GPIO12	Next Track
GPIO13	Play/Pause
GPIO5	Volume Up

Eg. rn52.AVRCPButtons(1); *//turn AVRCP buttons on.*

Eg. boolean state = rn52.AVRCPButtons(); *//reads the current state of the AVRCP buttons into the state variable.*

This format follows for all the following commands.

powerUpReconnect();

Determines whether the RN52 will attempt a reconnect on power up, it will attempt to reconnect with any device previously paired.

startUpDiscoverable();

Determines whether the RN52 is discoverable on start-up.

rebootOnDisconnect();

Determines whether the RN52 will restart after a disconnect, this is useful as after a disconnect the RN52 will not be discoverable.

volumeToneMute();

Mutes the volume tone (a tone which sounds whenever you change the volume).

systemTonesDisabled();

Disables the system tones, which will sound on power up and on connection.

powerDownAfterPairingTimout();

Enables the feature which will power down the RN52 after pairing is unsuccessful after a certain time.

resetAfterPowerDown();

Enables the reset after power down on the RN52, which will cause it to reset on every power cycle.

reconnectAfterPanic();

Panic is a mode the RN52 enters when it encounters a situation it has not been programmed for (like an overload of data). This feature enables the RN52 to reconnect after entering panic mode.

tonesAtFixedVolume();

Keeps the tones at a fixed volume independent of the media volume.

autoAcceptPasskey();

Disables the passkey.

Extended features – Advanced commands:

All the previous functions rely on the following two functions to operate correctly, if you'd rather edit the 16 bit Hexadecimal value yourself, you can use the following two functions to achieve this.

setExtFeatures(Option 1: 16-Bit Hex Value. Option 2: Boolean State, Int bit);

This command can be used to either set a new hex value, or to alter the state of a particular bit in the hex value. The bits are as follows:

Bit 0 – Enable AVRCP buttons

Bit 1 – Enable reconnect on power-on

Bit 2 – Bluetooth Discoverable on start up

Bit 3 – Codec indicators PIO7 (AAC) and PIO6 (aptX)

Bit 4 – Reboot after disconnect (otherwise must power cycle)

Bit 5 – Mute volume up/down tones

Bit 6 – Enable voice command button on PIO4

Bit 7 – Disable system tones

Bit 8 – Power off after pairing timeout

Bit 9 – Reset after power off

Bit 10 – Enable list reconnect after panic

Bit 11 – Enable latch event indicator PIO2

Bit 12 – Enable track change event

Bit 13 – Enable tones playback at fixed volume

Bit 14 – Enable auto-accept passkey in Keyboard I/O Authentication mode

Bit 15 – Blank

Bit 16 – Blank

Eg. rn52.setExtFeatures(0x50F6); *//Set the extended features hex to 50F6.*

Eg. rn52.setExtFeatures(1, 7); *//Disable the system tones.*

getExtFeatures();

This command returns the current value of the extended features. It is returned as a short i.e. two byte (16 bit) value.

Eg. Serial.println(rn52.getExtFeatures(), HEX); *//Prints the current extended features value as a hexadecimal value to the Serial port from the Arduino.*