Bluetooth

BASIC! implements Bluetooth in a manner which allows the transfer of data bytes between an Android device and some other device (which may or may not be another Android device). Before attempting to execute any BASIC! Bluetooth commands, you should use the Android "Settings" Application to enable Bluetooth and pair with any device(s) with which you plan to communicate.
When Bluetooth is opened using the **Bt.open** command, the device goes into the Listen Mode. While in this mode it waits for a device to attempt to connect.
For an active attempt to make a Bluetooth connection, you can use the Connect Mode by successfully executing the **Bt.connect** command. Upon executing the **Bt.connect** command the person running the program is given a list of paired Bluetooth devices and asked. When the user selects a device, BASIC! attempts to connect to it.
You should monitor the state of the Bluetooth using the **Bt.status** command. This command will report states of Listening, Connecting and Connected. Once you receive a "Connected" report, you can proceed to read bytes and write bytes to the connected device.
You can write bytes to a connected device using the **Bt.write** command.
Data is read from the connected device using the **Bt.read.bytes** command; however, before executing **Bt.read.bytes**, you need to find out if there is data to be read. You do this using the **Bt.read.ready** command.
Once connected, you should continue to monitor the status (using **Bt.status**) to ensure that the connected device remains connected.
When you are done with a particular connection or with Bluetooth in general, execute **Bt.close**.
The sample program, f35_bluetooth, is a working example of Bluetooth using two Android devices in a "chat" type application.

Bt.open {0|1}

Opens Bluetooth in Listen Mode. If you do not have Bluetooth enabled (using the Android Settings Application) then the person running the program will be asked whether Bluetooth should be enabled. After **Bt.open** is successfully executed, the code will listen for a device that wants to connect.
The optional parameter determines if BT will listen for a secure or insecure connection. If no parameter is given or if the parameter is 1, then a secure connection request will be listened for. Otherwise, an insecure connection will be listened for. It is not possible to listen for either a secure or insecure connection with one **Bt.open** command because the Android API requires declaring a specific secure/insecure open.
If **Bt.open** is used in graphics mode (after **Gr.open**), you will need to insert a **Pause 500** statement after the **Bt.open** statement.

Bt.close

Closes any previously opened Bluetooth connection. Bluetooth will automatically be closed when the program execution ends.

Bt.connect {0|1}

Commands BASIC! to connect to a particular device. Executing this command will cause a list of paired devices to be displayed. When one of these devices is selected the **Bt.status** will become "Connecting" until the device has connected.
The optional parameter determines if BT will seek a secure or insecure connection. If no parameter is given or if the parameter is 1, then a secure connection will be requested. Otherwise, an insecure connection will be requested.

Bt.disconnect

Disconnects from the connected Bluetooth device and goes into the Listen status. This avoids having to use **Bt.close** + **Bt.open** to disconnect and wait for a new connection.

Bt.reconnect

This command will attempt to reconnect to a device that was previously connected (during this Run) with **Bt.connect** or a prior **Bt.reconnect**. The command cannot be used to reconnect to a device that was connected following a **Bt.open** or **Bt.disconnect** command (i.e. from the **Listening** status).
You should monitor the Bluetooth status for **Connected** (3) after executing **Bt.reconnect**.

Bt.status {{<connect_var>}{, <name_svar>}{, <address_svar>}}

Gets the current Bluetooth status and places the information in the return variables. The available data are the current connection status (in <connect_var>), and the friendly name and MAC address of your Bluetooth hardware (in <name_svar> and <address_svar>).
All parameters are optional; use commas to indicate omitted parameters (see Optional Parameters).
If the connection status variable <connect_var> is present, it may be either a numeric variable or a string variable. The table shows the possible return values of each type:

Numeric Value	String Value	Meaning
-1	Not enabled	Bluetooth not enabled
0	Idle	Nothing going on
1	Listening	Listening for connection
2	Connecting	Connecting to another device
3	Connected	Connected to another device
4	Lost	Connection lost and try again
5	Failed	Getting a connection failed and try again

If the device name string variable <name_svar> is present, it is set to the friendly device name. If your device has no Bluetooth radio, the string will be empty.

If the address string variable <address_svar> is present, it is set to the MAC address of your Bluetooth hardware, represented as a string of six hex numbers separated by colons: "00:11:22:AA:BB:CC".

OnBtStatus:
Interrupt label that traps if the status of the bluetooth connection has changed. BASIC! executes the statements following the **OnBtStatus:** label until it reaches a **Bt.onStatus.resume**.

Bt.onStatus.resume
Resumes execution at the point in the BASIC! program where the **OnBtStatus:** interrupt occurred.

Bt.write {<exp> {,|;}} ...

Bt.utf_8.write {<exp> {,|;}} ...

Writes data to the Bluetooth connection.
If the comma (,) separator is used then a comma will be printed between the values of the expressions.
If the semicolon (;) separator is used then nothing will separate the values of the expressions.
If the semicolon is at the end of the line, the output will be transmitted immediately, with no newline character(s) added.
The parameters are the same as the **Print** parameters. This command is essentially a **Print** to the Bluetooth connection, with two differences:

- Only one byte is transmitted for each character; the upper byte is discarded. Binary data and ASCII text are sent correctly, but Unicode characters may not be.
  If you need the full Unicode character set use **Bt.utf_8.write**.
- A line that ends with a semicolon is sent immediately, with no newline character(s) added.

This command with no parameters sends a newline character to the Bluetooth connection.

Bt.read.ready <nvar>

Reports in the numeric variable the number of messages ready to be read. If the value is greater than zero then the messages should be read until the queue is empty.

OnBtReadReady:

Interrupt label that traps the arrival of a message received on the Bluetooth channel (see "Interrupt Labels"). If a Bluetooth message is ready (**Bt.read.ready** would return a non-zero value) BASIC! executes the statements after the **OnBtReady:** label, where you can read and handle the message. When done, execute the **Bt.onReadReady.Resume** command to resume the interrupted program.

Bt.onReadReady.resume
Resumes execution at the point in the program where it was interrupted by the Bluetooth Read Ready event.

Bt.read.bytes <svar>

Bt.utf_8.read.bytes <svar>

The next available message is placed into the specified string variable. If there is no message then the string variable will be returned with an empty string ("").
Each message byte is placed in one character of the string; the upper byte of each character is 0. This is similar to **Byte.read.buffer**, which reads binary data from a file into a buffer string.
If you need the full Unicode character set use **Bt.utf_8.read.bytes**.

Bt.device.name <svar>

Returns the name of the connected device in the string variable. A run-time error will be generated if no device (Status <> 3) is connected.

Bt.set.UUID <sexp>

A Universally Unique Identifier (UUID) is a standardized 128-bit format for a string ID used to uniquely identify information. The point of a UUID is that it's big enough that you can select any random 128-bit number and it won't clash with any other number selected similarly. In this case, it's used to uniquely identify your application's Bluetooth service. To get a UUID to use with your application, you can use one of the many random UUID generators on the web. Many devices have common UUIDs for their particular application. The default BASIC! UUID is the standard Serial Port Profile (SPP) UUID: "00001101-0000-1000-8000-00805F9B34FB". You can change the default UUID using this command.
~~Some information about 16 bit and 128 bit UUIDs can be found at:~~
~~http://farwestab.wordpress.com/2011/02/05/some-tips-on-android-and-bluetooth/~~