One of the most exciting features of the NXT is its built-in Bluetooth capability. The NXT's Bluetooth functionality comes from a CSR BlueCore4 chip, which has its own micro controller and an external 8MB Flash memory. It is entirely hidden from view, installed on the inside of the NXT, and has a range of approximately 10m or 33ft. In order for you to use wireless communication with your computer, it must have either built-in Bluetooth capability or a Bluetooth adapter. There are 3 main uses of Bluetooth in the NXT:

1. **Wireless Communication with a Bluetooth-capable computer.**
   Bluetooth communications can be used to connect to your PC instead of the USB link. You can do all the functions available over the USB link via Bluetooth except that NXT firmware can only be upgraded via USB. The USB communications is far faster than Bluetooth but requires a tethered link. Bluetooth, because it is wireless, is un-tethered. Using the official NXT software on your computer, you can download programs to your NXT, manage files on your NXT, and receive live feedback from your NXT.

2. **Wirelessly communicate with other NXT's (up to three, but one at a time).**
   Bluetooth can be used to connect multiple NXT’s together. When two NXT’s are connected via Bluetooth, one is the master device and one is the slave device. An NXT in master mode can support up to three slaves. An NXT in slave mode only supports a single Bluetooth connection.

3. **Wirelessly communicate with other Bluetooth-enabled devices such as cell phones.**
   You can use your cell phone to control an NXT by downloading an application from the MINDSTORMS website. This is possible as long as the phone communicates with the protocol specifications in the NXT Bluetooth Development Kit.

1. **How the master/Slave relationship works**

   When multiple NXT’s are connected together using Bluetooth communications, a master/slave relationship is established. The master and the slave NXT behave differently. The master is always the side that created the Bluetooth connection. If you create the connection from the Bluetooth menu of the NXT, that NXT is the master.

   The principle behind the master-slave relationship is simple: only the master is supposed to initiate communication. A message from the master may carry program-supplied data, or it may ask the slave for data that the program on the slave sent to the master. When the master sends data to the slave, it may request an acknowledgement or it may request that the slave accepts the data without acknowledging it. When the master requests data from the slave, the response contains the data if data is available, or an error code if no data is available or if no program is running on the slave.
One drawback to this master/slave relationship is that the LEGO programmers have designed the firmware in such a way that the NXT cannot serve as a master and a slave at the same time.

This means that a slave NXT/device cannot simply send a message to the master NXT; the master may be in communications mode with one of the other two possible devices and will not be listening to this particular slave. So, the slave must ‘buffer’ its message and wait for a polling request from the master asking for a buffer message.

The communications between NXT’s and other Bluetooth devices is done through one of four channels. Channel 0 - This channel is reserved for slave communication to the master. By default all slaves will send data to the master on channel 0. Channels 1, 2, 3 - The other three channels are used by the NXT master to send data or instructions to the slave on that channel.

2. Reliability of Bluetooth Communications for the NXT

The NXT's Bluetooth communication uses a Bluetooth application protocol called the Serial-Port Profile (SPP). This application-level protocol is implemented on top of a low-level protocol called RFCOMM. RFCOMM is a reliable transport protocol. RFCOMM is not allowed to loose data.

But this does not mean that the NXT Bluetooth protocol does not loose messages: it does, because the mailbox queues are short. Each mailbox queue can contain up to 5 messages. If a message must be pushed into a full queue, the oldest message in the queue is deleted. Also, when there is no program running on the slave, the slave’s firmware ignores Bluetooth-communication messages. Therefore, if the master sends messages before the slave's program starts, these messages will be lost. Also, if the slave sends a message (or a few) and ends it's program, these last messages are typically lost, because the queues are cleared before the master polls for them.

3. Setting up your NXT for Bluetooth communication

The NXT GUI (Graphical User Interface) provides manual capabilities to manage the Bluetooth configuration on the NXT. You can set up and tear down connections, search for devices, remove items from the contacts list and many more functions.

To connect the two units together via Bluetooth follow these steps:

1. Turn both units on, and make sure Bluetooth is enabled on both, and that both are set to 'visible'.
2. On the master NXT: navigate to Bluetooth -> Search
3. A search diagram will appear on the screen, be patient while the search progresses.
4. When the menu reappears select the receiver unit from the list, and then select connection slot #1.
5. A connection screen will appear, again be patient.
6. You will be prompted for a passkey on both units. Enter the same key on both and press enter to confirm. Leaving the default of 1234 is perfectly acceptable.
7. On the master, again select the option for the receiver
8. At this point you should see the receiver unit displayed with a small '1' in the upper left corner. Select it one last time and then back out to the main menu. You are now configured for Bluetooth communication between multiple NXT’s.
4. Sample code for Bluetooth Communication

This is some sample code for the master NXT. This code and more can be found at: http://lejos.sourceforge.net/ Download leJOS NXJ and look for the samples folder.

```java
public class BTConnectTest {
    public static void main(String[] args) throws Exception {
        String name = "Lejos2";

        RemoteDevice btrd = Bluetooth.getKnownDevice(name);

        if (btrd == null) {
            LCD.clear();
            LCD.drawString("No such device", 0, 0);
            LCD.refresh();
            Thread.sleep(2000);
            System.exit(1);
        }

        BTConnection btc = Bluetooth.connect(btrd);

        if (btc == null) {
            LCD.clear();
            LCD.drawString("Connect fail", 0, 0);
            LCD.refresh();
            Thread.sleep(2000);
            System.exit(1);
        }

        LCD.clear();
        LCD.drawString("Connected", 0, 0);
        LCD.refresh();

        DataInputStream dis = btc.openDataInputStream();
        DataOutputStream dos = btc.openDataOutputStream();

        for(int i=0;i<100;i++) {
            try {
                LCD.drawInt(i*30000, 8, 0, 2);
                LCD.refresh();
                dos.writeInt(i*30000);
                dos.flush();
            } catch (IOException ioe) {
                LCD.drawString("Write Exception", 0, 0);
                LCD.refresh();
            }

            try {
                LCD.drawInt(dis.readInt(),8, 0,3);
                LCD.refresh();
            } catch (IOException ioe) {
                LCD.drawString("Read Exception ", 0, 0);
                LCD.refresh();
            }
        }
    }
}
```
This is some sample code for the slave NXT

public class BTReceive {

    public static void main(String[] args) throws Exception {
        String connected = "Connected";
        String waiting = "Waiting...";
        String closing = "Closing...";

        while (true) {
            LCD.drawString(waiting, 0, 0);
            LCD.refresh();

            BTConnection btc = Bluetooth.waitForConnection();

            LCD.clear();
            LCD.drawString(connected, 0, 0);
            LCD.refresh();

            DataInputStream dis = btc.openDataInputStream();
            DataOutputStream dos = btc.openDataOutputStream();

            for (int i = 0; i < 100; i++) {
                int n = dis.readInt();
                LCD.drawString(n, 7, 0, 1);
                LCD.refresh();
                dos.writeInt(-n);
                dos.flush();
            }

            dis.close();
            dos.close();
            Thread.sleep(100); // wait for data to drain
            LCD.clear();
            LCD.drawString(closing, 0, 0);
            LCD.refresh();
            btc.close();
            LCD.clear();
        }
    }
}

Sources:
http://lejos.sourceforge.net/nxj-downloads.php
http://www.tau.ac.il/~stoledo/lego/btperformance.html