|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Gadgeteer Community |  |  |
|  | XBeeClient | PM:  Dev/test: | Paul Mineau pamin@microsoft.com |
|  | Technical Design Specification |  |  |

# Overview

This Document describes the technical design of the Gadgeteer Community XBeeClient module for XBee. It describes the test design, API, and contains technical notes. Users wishing to quickly use XBeeClient should visit the short ‘Getting Started’ document instead.

The XBeeClient is a Gadgeteer module which makes it easy to send and receive messages using XBee devices. It provides a simple user friendly API. Functional tests are a first class citizen and are an integral part of the client API (see Section 4).

The XBeeClient can be used on the PC as well. The Interactive Console is a PC console application you can use to do ad hoc testing.

The XBeeClient also manages sending large messages using a simple protocol for chunking large messages (see section 3.1).

The XBee Client is modeled after the System.Net.WebClient class because of its simple API and eventing model, and offers asynchronous methods. Synchronous methods aren’t offered at this time and may be added later if there is demand.

XBee AT commands can be easily invoked and the responses generically parsed, allowing the user to add new functionality. (Move to section on issuing AT commands). For example, there is no XBeeClient method for querying the voltage, but you can issue an AT command ‘V%’, and parse the response payload bytes into an integer.

The API currently has a method for sending pictures, UploadPictureAsync. As pictures taken on Gadgeteer can be hundreds of thousands of bytes, they result in messages that have thousands of packets. They will work but take quite a long time. A design is needed to allow the most rapid delivery of pictures.

## Goals

* Provide an easy to use API for sending and receiving XBee messages, so easy it can be used in High School classroom settings
* Provide an easy mechanism to execute AT commands, local and remote
* Provide high quality sample applications demonstrating capabilities
* Make Test a First Class Citizen, as an integral part of the API, so functional tests can be easily run locally and remotely
* Invest in monitoring and diagnostics
* Send large messages by chunking them

## Non-Goals

* Supporting every XBee device on the market
* Support TCP/IP or other network protocols like HTTP (this may be addressed in future versions)

# Scenarios

## Send short broadcast messages

Users getting started will want to simply send a hello world to all connected XBees.

## Misconfigured XBees

To get started, users must have their XBees configured with API Mode 2 (w/ escaped bytes), and the coordinator and router must be in the same PAN.

This is a source of great frustration when getting started, and using XCTU can be frustrating. To make this part of the process easy, step by step documentation and tutorials will guide them through, as well as user friendly error messages that give precise instructions.

## Sending messages to one destination device

Messages destined for a single device will not be broadcast. They should be able to use the full Serial Number and optionally MY address, or use a friendly identifier like “Smoke Detector in Kitchen”.

## Sending large messages

The API needs to support sending strings, pictures, and other large messages reliably. Event handlers on the client will notify when strings, pictures, and data has been received. Sending a picture will be one line of code as will be receiving.

## Local and Remote AT Commands

The user will like to execute any AT command without having to add code to the client library.

# Design

The client library has a simple eventing model similar to the System.Net.WebClient. Here is the method for sending a ‘hello world’ string. The following code snippet shows one XBee Client sending a string to all other XBees (broadcast).

// In your main method in the project sending a message

XBeeClient xbeeClient = new XBeeClient(11);

xbeeClient.UploadStringAsync(“Hello World”);

To have another XBee client receive this message, wire it up as follows:

// In your main method in your project receiving the message

XBeeClient = new XBeeClient(6);

xbeeClient.StringReceived += new StringReceivedEventHandler(xbeeClient\_StringReceived);

…

void xbeeClient\_StringReceived(object sender, StringReceivedEventArgs e)

{

Debug.Print("String Received: " + e.Message);

}

## Typed Multipart Messages

The XBee protocol doesn’t support sending messages that span multiple frames, e.g. sending a picture, or a large string. XBeeClient overcomes this by introducing typed messages. The class that generates multipart messages is MultipartMessageBuilder. If the payload is over 100 bytes, it splits the message into multiple messages. Ideally we will get the maximum message size from an AT command, so far this has been unsuccessful.

A typed message does not need to be multipart, so the class MultipartMessageBuilder should be renamed.

A typed transmit message uses the first 4 bytes of the payload to describe the typed message. A transmit message is specified as a typed message if the first byte in the message payload is 0x7C. The second byte specifies the total number of packets in the message. The third specifies the packets index. The fourth byte specifies the data type of the message. The data types are specified in the MessageDataTypeEnum, and there are 3 types so far: String, Picture, and Data.

# API

## Public Methods

Note that an entirely Async model is used. It’s possible that future versions will have some synchronous methods as well.

|  |  |  |
| --- | --- | --- |
| API Method | Implemented? | Notes |
| UploadStringAsync(string message) | green-checkmark.jpg | Only SH so far, need every AT command |
| UploadStringAsync(NodeDescription nodeDescription, string message) | green-checkmark.jpg |  |
| RunFunctionalTests(int numExpectedPeers) | green-checkmark.jpg |  |
| DiscoverNodesAsync() | green-checkmark.jpg |  |
| SendLocalATCommandAsync(string command) | green-checkmark.jpg |  |
| PrintBytesReceievedHistory() | green-checkmark.jpg | If the RecordBytesReceivedHistory property is set to true, this API method will print all the bytes received on serial line. |
| UploadPictureAsync(byte[] data) | green-checkmark.jpg |  |
| UploadPictureAsync(NodeIdentifier nodeIdentifier, byte[] data) | green-checkmark.jpg |  |
| UploadStringAsync(string destinationNodeIdentifier, string message) |  |  |
| UploadDataAsync(byte[] data) |  |  |
| SendLocalATCommandAsync(string command, byte[] val) |  |  |
| SendRemoteATCommandAsync(NodeDescription nd, string command, byte[] val) |  |  |

## Public Events

/// <summary>Occurs when an asynchronous discover nodes operation completes.</summary>

public event DiscoverNodesCompletedEventHandler DiscoverNodesCompleted;

/// <summary>Occurs when an asynchronous upload string operation completes.</summary>

public event UploadStringCompletedEventHandler UploadStringCompleted;

/// <summary>Occurs when you send a message and receive the status.</summary>

public event TransmitStatusReceivedEventHandler TransmitStatusReceived;

/// <summary>Occurs when a response from an AT command is received.</summary>

public event ATResponseReceivedEventHandler ATResponseReceived;

/// <summary>Occurs when a transmit receive message is received (we're receiving data).</summary>

public event TransmitReceiveEventHandler TransmitReceive;

/// <summary>Occurs when data is received.</summary>

public event DataReceivedEventHandler DataReceived;

/// <summary>Occurs when a string is received.</summary>

public event StringReceivedEventHandler StringReceived;

/// <summary>Occurs when a test completes, this will get fired for each test in your test run.</summary>

public event TestCompletedEventHandler TestCompleted;

# Test Plan

## Functional Tests

* The XBeeClient has a built in functional test feature. Execute it by calling the XBeeClient method RunFunctionalTests. To be notified when each test finishes, use the TestCompleted even handler.

xbeeClient.TestCompleted += new TestCompletedEventHandler(xbeeClient\_TestCompleted);

// pass in the number of expected peers

BaseTest[] testResults = xbeeClient.RunFunctionalTests(1);

The test completed event can be used for printing results to a console window so you can see progress.

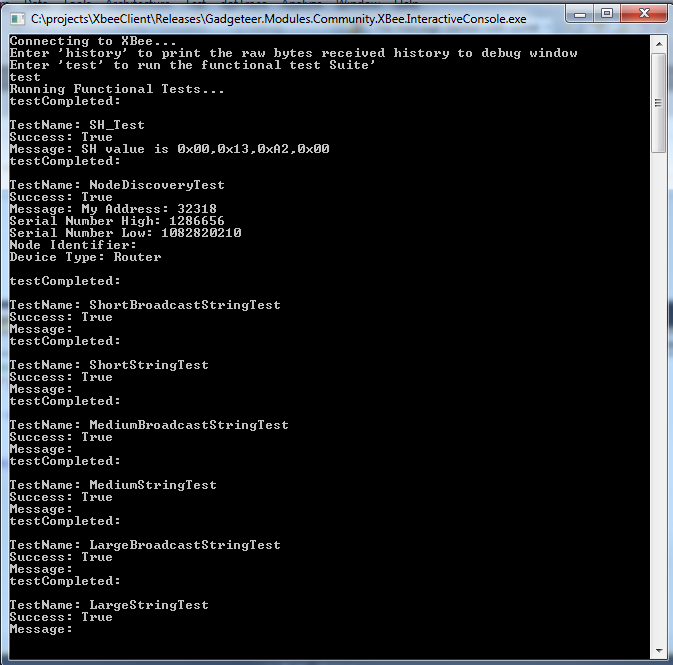
static void xbeeClient\_TestCompleted(object sender, TestCompletedEventArgs e)

{

Console.WriteLine("testCompleted: " + e.Test.ToString());

}

* You can run the tests from the Interactive Console by typing ‘test’.



* Functional Test Matrix

|  |  |  |
| --- | --- | --- |
| Test | Passed? | Notes |
| AT Commands | green-checkmark.jpg | Only SH so far, need every AT command |
| Node Discovery | green-checkmark.jpg |  |
| Upload Short Broadcast String | green-checkmark.jpg |  |
| Upload Short String | green-checkmark.jpg |  |
| Upload Medium Broadcast String | green-checkmark.jpg |  |
| Upload Medium String | green-checkmark.jpg |  |
| Upload Large Broadcast String | green-checkmark.jpg |  |
| Upload Large String | green-checkmark.jpg |  |
| Upload Small Data | green-checkmark.jpg | The tests for medium, large, as well as broadcast need to be written |

## Unit Tests

Unit tests are the specification of behavior. The project currently needs more unit tests. Functional tests are useful for helping customers identify issues. Unit tests are useful for verifying builds and catching bugs earlier. Both unit tests and functional tests need to be run before checkin.

|  |  |  |
| --- | --- | --- |
| Test | Passed? | Notes |
| TransmitReceivedWithFullMessage | green-checkmark.jpg |  |
| TransmitStatusReceivedWithFullMessage | green-checkmark.jpg |  |
| TransmitReceivedDataCorrect | green-checkmark.jpg |  |
| ToUShort | green-checkmark.jpg | Convert byte array to ushort |
| DiscoverNodesWithMessageInTwoChunks | green-checkmark.jpg |  |
| DiscoverNodesWithMessageInTwoChunks  WhereFirstChunkIsBeforeLengthByte | green-checkmark.jpg |  |
| DiscoverNodesWithFullMessage | green-checkmark.jpg |  |
| ToUInt | green-checkmark.jpg |  |

## Scenario Tests

Scenario tests are implemented in full sample projects. See the site wiki for example projects that use XBeeClient.