Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Interfaz Software Estándar en C/C++ para Comunicación mediante Módulos XBee

Autor: Antonio Rafael Navea Jiménez

Tutor: Daniel Rodríguez Ramírez

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Proyecto Fin de Carrera

Ingeniería de Telecomunicación

Interfaz Software Estándar en C/C++ para Comunicación mediante Módulos XBee

Autor:

Antonio Rafael Navea Jiménez

Tutor:

Daniel Rodríguez Ramírez

Profesor Titular

Dep. Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2016

Proyecto Fin de Carrera: Interfaz Software Estándar en C/C++ para Comunicación mediante Módulos XBee

Autor: Antonio Rafael Navea Jiménez
Tutor: Daniel Rodríguez Ramírez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En especial quiero dar mi más sincero agradecimiento a mi familia, por su apoyo incondicional en todos estos largos años de mucho esfuerzo y sacrificios económicos, porque nunca han dejado de creer en mí: a mis padres y hermanos, a mis abuelos por haber sido siempre como unos segundos padres para mí y un espejo en el que mirarme, y a mi tía Julia.

Y a todas aquellas buenas e inestimables amistades que me han acompañado en algún momento de este largo período de mi vida, porque han puesto su granito de arena para animarme a seguir siempre adelante, y así no rendirme ni dejar de luchar nunca hasta lograr alcanzar mi sueño de convertirme en ingeniero, tanto en mis años en Sevilla, como en mi estancia Erasmus en Milano; muy especialmente quiero acordarme aquí de Lydia, Marta, Lorena, Ele, Fer, Esther, Fran, Pablo, Ana, Irene, Inés y Montse.

Y no puedo olvidarme tampoco de los maestros y profesores que me han acompañado a lo largo de cada etapa educativa, así como de la serie de animación "*Erase una vez... la vida*" y "*Erase una vez... los inventores*" que emitía Televisión Española en la década de los 90, la cual despertó en mí esa curiosidad innata por descubrir el funcionamiento del mundo que ya me acompañará toda mi vida, así como el interés por la ciencia en general y la ingeniería en particular.

Antonio Rafael Navea Jiménez

Sevilla, septiembre 2016

Resumen

Los módulos *XBee*® RF embebidos (de Digi International) son utilizados en numerosas plataformas de sensores inalámbricos para realizar intercambios de datos mediante el uso de radiofrecuencia. El acceso a estos módulos puede realizarse a través de un puerto serie estándar, y por tanto, su uso no tiene porqué estar estrictamente restringido a dichas redes de sensores. En el presente proyecto se ha desarrollado una librería en código C/C++ que permite el uso de módulos *XBee-PRO*® *868 MHz* a través de interfaces de comunicación serie, facilitando la creación de programas que hacen uso de las características de los *XBee*® para transmitir o recibir datos sobre plataformas muy diversas (actualmente con soporte para sistemas operativos POSIX: Windows/Cygwin y Linux con GCC), tales como PC's o dispositivos Raspberry Pi.

Abstract

The embedded *XBee*® RF modules (from Digi International) are widely used in several wireless sensors platforms to exchange data over a radiofrequency link. The access to these modules can be achieved through a standard serial port, so its use is not strictly restricted to sensor networks. In this project, it has been developed a C/C++ library that allows the use of the modules *XBee-PRO*® *868 MHz* through serial communication interfaces, providing an easier way to create programs that make use of the features of the XBee to transmit or receive data on many different platforms (currently supports POSIX operating systems: Windows/Cygwin and Linux with GCC) such as PCs or Raspberry Pi devices.

Índice Abreviado

Índice

Parte I.

State-of-the-Art

1 Introducción

En la primera parte de este documento se realiza una introducción al contexto tecnológico, temática, y motivación del presente trabajo, así como la descripción del "Estado del Arte", consistente en un estudio previo sobre las soluciones *software* ya existentes para problemáticas similares a la abordada en este proyecto, particularizado para los módulos ***XBee-PRO®868*** que serán, como se explicará, los dispositivos seleccionados para el presente proyecto.

En la segunda parte del documento, se incorpora la *Guía de Usuario y Documentación del Código*. Primeramente se describirá el código de la librería que se ha desarrollado para este trabajo. Posteriormente, se hace una breve explicación de las tramas API usadas en el protocolo de los módulos *XBee-PRO®868*, para poder entender adecuadamente el uso y funcionalidad de cada una de las funciones de la librería, que se ha denominado *UniversalXBee*.

1.1 Motivación del proyecto

Los módulos *XBee®* RF embebidos (de Digi International) son utilizados en numerosas plataformas de sensores inalámbricos para realizar intercambios de datos mediante el uso de radiofrecuencia. El acceso a estos módulos puede realizarse a través de un puerto serie estándar, y por tanto, su uso no tiene porqué estar restringido a dichas redes de sensores. Puesto que no existe una librería o API que de forma fácil e intuitiva permita aprovechar dicha característica sobre plataformas como PC's o dispositivos *Raspberry Pi*, se estudiarán las soluciones *software Open-Source* ya existentes para otras plataformas, como *Waspmote* o *Arduino*, y que permiten utilizar módulos *XBee®* como interfaz de comunicaciones.

Del anterior estudio, se extraerá el conocimiento necesario para desarrollar una librería en algún lenguaje de programación (como Python, Java, C, C++, o algún otro) que sea de fácil uso y mantenimiento, que sea soportado por una gran variedad de dispositivos, y que permita el uso de módulos *XBee®* a través de interfaces de comunicación serie, facilitando la codificación de programas creados *ad-hoc* que hagan uso de las características de los *XBee®* para transmitir o recibir datos a través de un puerto serie.

1.2 Contexto tecnológico

Aunque en la actualidad, el término *Internet-of-Things* (IoT) se ha convertido en un vocablo familiar para todo tipo de públicos, no fue hasta 1999 cuando el término fue acuñado por Kevin Ashton, co-fundador y director ejecutivo del *Auto-ID Center*[1] del *Massachusetts Institute of Technology* (MIT), que para llamar la

[1] *Auto-ID Center* es una federación de universidades investigadoras, que fue fundada en 1999 para desarrollar una arquitectura estándar abierta que permitiera crear una red global sin fronteras de objetos físicos conectados mediante RFID, idea primigenia del *Internet-of-Things*. Fue fundada en parte por EPCglobal, gobierno, industria, y seis de las universidades más prestigiosas del mundo: MIT, Universidad de Cambridge, Universidad de Adelaide, Universidad de Keio, Universidad de Fudan, y la Universidad de St. Gallen. Posteriormente, *Auto-ID Center* evolucionó en el actual *Auto-ID Labs* (*http://autoidlabs.org/*).

atención de los asistentes utilizó *Internet of Things* como título de una presentación que realizó para explicar como se podría aplicar RFID en la cadena de suministro de la empresa P&G (Procter & Gamble).[2]

Posteriormente, el paradigma del *Internet de las Cosas* no comenzó realmente a emerger de forma práctica y viable hasta hace poco más de una década. Como constatación de este hecho, en 2005, la *International Telecommunication Union* (ITU) publicó un artículo[3] en el que adscribió la futura evolución del *Internet-of-Things* como función de un conjunto de avances tecnológicos de diferentes campos de estudio, específicamente de áreas como la conectividad móvil e inalámbrica, nano-tecnología, identificación por radiofrecuencia (RFID), y tecnologías de sensores inteligentes. Los avances en estas tecnologías, cuando se combinasen, ayudarían a llevar a cabo dispositivos miniaturizados, embebidos, y automáticamente conectados a Internet, permitiendo que puedan comunicarse de forma frecuente y relativamente sin esfuerzo. Claramente, de la anterior descripción del IoT se deduce la inherente importancia en este paradigma de las aplicaciones y servicios *Machine-To-Machine* (M2M)[4].

En este tiempo, el número de dispositivos conectados al *Internet-of-Things* ha crecido de forma imparable a un ritmo vertiginoso difícilmente imaginable hace una década. Este crecimiento se debe, en parte, al incremento de disponibilidad en el mercado de gran variedad de *hardware* pequeño, portable, y de bajo costo, y por otra parte, al auge de las soluciones de código abierto o *software* libre *Open-Source*[5], como por ejemplo las creadas por la *FSF*[6] y extensamente utilizadas *GNU General Public License* (GPL) y *GNU Lesser General Public License* (LGPL).

A finales de 2015, Cisco[7] estimaba el número total de dispositivos conectados a Internet, incluyendo teléfonos móviles, parquímetros, termostatos, dispositivos en viviendas domóticas, monitores sanitarios (cardíacos,...), carreteras, coches, artículos en supermercados, y muchos otros tipos de objetos de uso cotidiano, en el entorno de los 16300 millones (recordar aquí que la población mundial en 2016 se estima en el entorno de los 7400 millones).

En este contexto tecnológico, se han desarrollado multitud de dispositivos para IoT, kits de prototipado y placas de desarrollo, tanto para un público profesional, como para los usuarios aficionados al *Do-It-Yourself* (DIY), que se engloban bajo la misma filosofía que sigue la anteriormente comentada cultura *Open-Source*, y que se conoce como *Open-Source Hardware* (OSH), o simplemente, *open-hardware*. Este *hardware* combina el uso de micro-controladores, micro-procesadores, chipsets, puertos de comunicación, y otros componentes, para crear dispositivos tipo *System-on-Chip* (SoC)[8] o *Computer-on-Module* (CoM)[9] listos para usar y/o programar por el usuario. Dichos sistemas SoC permiten gran cantidad de diferentes y variadas configuraciones, desde pequeños chips alimentados por una pequeña batería, hasta mini-ordenadores del tamaño de una tarjeta de crédito alimentados mediante un puerto USB, y que permiten la comunicación mediante tecnologías como Bluetooth, WiFi, o radiofrecuencia (RF).

---

[2] "That Ínternet of Things'Thing" (By Kevin Ashton). Disponible en: *http://www.rfidjournal.com/articles/view?4986*

[3] "ITU Internet Report 2005: The Internet of Things." Disponible en: *http://www.itu.int/osg/spu/publications/internetofthings/* y *http://www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf*

[4] Vodafone define M2M como el intercambio de datos que se produce de forma remota e inalámbrica entre dos o más dispositivos, o entre estos y una estación central, la cual realiza las funciones de monitoreo y control en remoto de dichos dispositivos y sus procesos (artículo publicado por Vodafone el 29 de abril de 2010, bajo el título "M2M: A new way of working." Disponible en: *http://www.vodafone.com/business/global-enterprise/a-new-way-of-working-2010-04-29*).

[5] Open Source Initiative: *https://opensource.org/*

[6] Free Software Foundation: *http://www.fsf.org/*

[7] "Cisco, Visual Networking Index 2016." Disponible en: *http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html*

[8] System-on-Chip (SoC) hace referencia al uso de tecnologías de fabricación que integran todos o gran parte de los módulos que componen un computador, o cualquier otro sistema informático o electrónico, en un único circuito integrado o chip, especialmente en los sistemas embebidos.

[9] Computer-on-Module (CoM) es un tipo de Single-Board-Computer (SBC), una extensión del concepto de System-on-Chip (SoC) y del System-in-Package (SiP).

2 Descripción del Estado del Arte

En este capítulo se presenta un breve estudio sobre las alternativas *software* ya existentes para problemáticas similares a la abordada en este proyecto, tomando como punto de inicio la descripción del *hardware* implicado, y finalizando con un análisis del código de las soluciones encontradas que sean susceptibles de ser utilizadas como base de estudio para la librería de este proyecto.

2.1 Plataforma Waspmote (Libelium)

De las plataformas listadas en la Tabla 2.1 (pequeña lista de los dispositivos *open-hardware* embebido tipo SoC o CoM más conocidos y utilizados), a continuación se describirá brevemente la desarrollada por la empresa española *Libelium*, y comercializada bajo el nombre de [19], ya que jugará un importante papel en el desarrollo de este proyecto.

Tabla 2.1 Ejemplos de Plataformas *open-hardware*.

- Arduino:	*https://www.arduino.cc/*
- BeagleBone:	*https://beagleboard.org/*
- Nanode:	*http://www.nanode.eu/*
- Odroid:	*http://www.hardkernel.com/*
- Raspberry Pi:	*https://www.raspberrypi.org/*
- Waspmote:	*http://www.libelium.com/products/waspmote/*

La arquitectura *hardware*[1] de los dispositivos *Waspmote* ha sido especialmente diseñada con el objetivo de minimizar al máximo posible el consumo de potencia. Switches digitales permiten conectar y desconectar, según sea necesario, cualquiera de los módulos de comunicación radio o interfaces de sensores. Además, los *Waspmote* incorporan dos modos de bajo consumo o *sleep mode*.

Por otro lado, *Waspmote* dispone de más de 110 sensores compatibles diferentes, que abarcan todo tipo de usos: gases, sensores de presencia, medición de la humedad del suelo y ambiental, temperatura, radiación, líquidos, luminosidad, etc. Dichos sensores se pueden comprar listos para su uso en las denominadas *Sensor Boards*[2].

Además, actualmente hay disponibles 17 diferentes interfaces de comunicación inalámbrica para los *Waspmote*, que pueden utilizarse de forma aislada o combinando dos de ellas mediante la *Expansion Radio Board*[3]. Las interfaces disponibles se pueden clasificar según su radio de cobertura en:

- Long range: 3G, GPRS, LoRaWAN, LoRa, Sigfox, 868 MHz, 900 MHz
- Medium range: protocolos WiFi, *ZigBee*, *IEEE 802.15.4*

[1] Waspmote Hardware: *http://www.libelium.com/products/waspmote/hardware/*
[2] Waspmote Sensor Boards: *http://www.libelium.com/products/waspmote/sensors/*
[3] Waspmote Wireless Interfaces, Expansion Radio Board, y Gateway: *http://www.libelium.com/products/waspmote/interfaces/*

- Short range: RFID, NFC, Bluetooth v4.0

No es propósito de este proyecto abordar el resto de características y módulos disponibles de *Waspmote*, ni tampoco profundizar en las anteriormente comentadas, por lo que se insta al lector que desee obtener más información a consultar la bibliografía [19].

Una de las características más apreciadas y usadas de *Waspmote* es su compatibilidad con *XBee*® , unos módulos RF embebidos fabricados por la empresa *Digi International* [25]. *XBee*® comprende una amplia gama de productos, diseñados para trabajar en las bandas ISM para dispositivos de baja potencia en las radiofrecuencias no licenciadas de 868 MHz, 915 MHz, y 2.4 GHz.

En Europa, el uso de la banda *Industrial, Scientific and Medical (ISM)*[4] es cubierto por la normativa sobre dispositivos de corto alcance, *Show Range Devices* (SRD)[5]. Se debe tener en cuenta que la banda SRD, entre 863 MHz y 870 MHz, es de uso exclusivo en Europa, mientras que la banda ISM comprendida entre 902 MHz y 928 MHz se utiliza en EE.UU., Canadá, Australia e India (en Europa dicha banda es utilizada por servicios GSM-900 de telefonía móvil). Por último, la banda ISM entre 2.4 GHz y 2.5 GHz está armonizada a nivel mundial para su uso por protocolos de área personal, como *IEEE 802.15.4* o *ZigBee*.

2.2 Módulos XBee-PRO® 868

De entre los dispositivos *XBee*® habilitados para trabajar en Europa, para este proyecto se escogió el ***XBee-PRO® 868*** (Tabla 2.2), que trabaja en la banda SRD g3 (frecuencia central = 869.525 MHz), utilizando un único canal de 250 KHz de ancho de banda, caracterizados por su alta sensibilidad en recepción ($-112dBm$), elevada potencia de transmisión (hasta $+25dBm$), y largo alcance en la comunicación (hasta 80 Km LOS[6] con una antena de alta ganancia), y que utilizan un protocolo punto-a-multipunto propietario de *XBee*® [22].

Este módulo de bajo coste, bajo consumo, y alto rendimiento, no necesita de ninguna configuración *out-of-the-box*[7] adicional para realizar las primeras comunicaciones RF, e implementa un protocolo simple, pero completo y versátil: cuenta con una variedad de tramas que, si bien puede parecer pequeña en comparación con otros módulos (por ejemplo, los que implementan el protocolo *ZigBee*), permiten realizar tareas de red avanzadas, soportando *point-to-point*, *point-to-multipoint*, y *peer-to-peer* (P2P) como topologías de red.

No en vano, estudiando en detalle tanto el protocolo usado por los módulos *XBee*® que implementan *ZigBee* como el protocolo de los módulos *XBee-PRO® 868*, se comprueba rápidamente que las tramas de este último constituyen en realidad un sub-conjunto del set total de tramas que contempla el protocolo de los módulos *XBee*® *ZigBee*, por lo que la librería que se desarrolle en este proyecto será fácilmente extrapolable/ampliable al protocolo usado por los módulos *ZigBee*.

Estos módulos permiten diseñar una red inalámbrica punto-a-punto y punto-a-multipunto de largo alcance utilizando un bajo número de nodos, lo que redunda en la reducción de los costes de despliegue y mantenimiento de la infraestructura de red.

XBee® dispone en la página web oficial de Digi International diversos recursos y documentación (utilidades, manuales de usuario, especificaciones técnicas, etc.) así como de una Base de Conocimiento (*Knowledge Base* [23]) con multitud de artículos que ilustran el uso de *XBee*® , dando respuesta a las preguntas, problemas y dudas más frecuentes en su uso.

Entre las utilidades destaca el *software* gratuito multi-plataforma ***XCTU*** [26] (Plataforma de Configuración de Nueva Generación para Dispositivos XBee/RF), que incluye una herramienta intuitiva y de fácil uso para desarrollar e interpretar las diferentes tramas *XBee-API* y tramas *XBee-AT* existentes para cada uno de los módulos *XBee*® , y que será de gran utilidad para el aprendizaje, desarrollo y *testing* de la librería creada en este proyecto. Asimismo, *XCTU* aporta sendas consolas para el envío/recepción de tramas a través de puerto serie a/desde un módulo *XBee*® , así como la posibilidad de administrar y configurar varios dispositivos

[4] Las bandas de radiofrecuencia ISM están reservadas internacionalmente para emisiones de potencia radioeléctrica en usos industriales, científicos y médicos, diferentes a las emisiones que son propias de los servicios de telecomunicaciones.

[5] El uso de SRD (Dispositivos de Corto Alcance) sigue la "Recomendación ECC 70-03". Más información en: *http://www.etsi.org/index.php/technologies-clusters/technologies/radio/short-range-devices*

[6] *Line-Of-Sight* (LOS), o Línea de Visión Directa radioeléctrica entre dos puntos (antenas)

[7] *Out-of-the-box*: referido a un producto tecnológico, es una característica o funcionalidad de un producto por la que este funciona inmediatamente después de su instalación o encendido, sin necesidad de realizar configuraciones o modificaciones adicionales.

XBee/RF conectados, tanto en local como de forma remota, visualizando fácilmente la topología de red existente entre ellos.

Tabla 2.2 Principales características de los módulos RF *XBee-PRO® 868*.

Módulo	*XBee-PRO® 868*
Banda RF	Banda SRD g3 (869.4 – 869.65 MHz)
Descripción	Módulo de largo alcance punto-a-multipunto para Europa
Alcance RF en interiores o entornos urbanos	Hasta 550 metros
Alcance RF LOS en exteriores	Hasta 40 km con antena dipolo de 2.0 dB Hasta 80 km con antena de alta ganancia
Potencia de Transmisión	$+25dBm$ ($315mW$)
Sensibilidad en Recepción (1% PER)	$-112dBm$
RF Data Rate	$24kbps$
Consumo (TX)	85 - 500 mA, dependiendo de la configuración
Consumo (RX)	65 mA (@3.3 V)

No obstante, el *software XCTU* no es de utilidad si se piensa utilizar *XBee®* como interfaz de comunicaciones de un nodo sensor autónomo, formando parte de algún *software* creado *ad-hoc* para transmitir datos dentro de la red a la que pertenece. Por ello, se hace necesario utilizar una librería o API diseñada *ex profeso* que permita realizar la comunicación del host o nodo sensor con el módulo *XBee®* a través de la interfaz serie que este último proporciona, y conforme al protocolo y estructura de tramas que aplique al módulo en cuestión, permitiendo así, en última instancia, la comunicación RF exitosa con otros nodos de la red.

Con este propósito en mente, se ha realizado una búsqueda en la web de recursos (librerías, APIs) ya existentes que satisfagan las necesidades explicadas en el anterior párrafo, comprobando especialmente si aportan soporte para el módulo *XBee-PRO® 868*, en el cual estamos interesados. A continuación, se lista el resultado de dicha búsqueda. En posteriores apartados (Sección 2.3 y Sección 2.4) de este documento se describirá en profundidad las características de cada item.

- Plataforma *XBee®* (Digi International): proporciona una librería portable y multi-plataforma, escrita conforme al estándar ANSI-C.
- Plataforma *Waspmote*: proporciona API e IDE propios, basados en el paquete *avr-libc*.
- Plataforma Arduino: proporciona una librería creada por un usuario, basada en el paquete *avr-libc*.

Todas las librerías listadas anteriormente tienen una característica en común: son recursos *Open-Source*, y basan su desarrollo en el exitoso y conocido lenguaje de programación C. En este punto, se recomienda leer el Apéndice A (incluyendo el Apéndice A.1 y el Apéndice A.2) para entender diferentes conceptos, relativos al lenguaje C y C++, que aparecerán en los próximos capítulos y secciones.

2.3 Digi International: librería "XBee® ANSI-C"

Esta librería es una colección de código portable y multi-plataforma, escrito en lenguaje C, bajo el estándar *ANSI-C*, desarrollada por Digi International [9] para usar módulos RF *XBee®* en *API-mode*. Esta librería es *Open-Source* (Mozilla Public License v2.0), y actualmente soporta las siguientes plataformas:

- Sistemas operativos POSIX (Windows/Cygwin, Mac OS X, Linux, BSD) con el compilador gcc.
- Windows con MinGW y gcc.
- DOS usando el compilador OpenWatcom.
- Micro-procesadores tipo Rabbit (usando Dynamic C 10.70 o posterior).
- Freescale HCS08 con CodeWarrior 10.x (parte del "Programmable XBee Dev Kit").

Aunque a primera vista, la librería *Digi XBee ANSI-C* parece satisfacer los requisitos buscados, un estudio en profundidad alerta rápidamente de diversos problemas:

1. La última edición del código data del mes de noviembre del año 2012, lo que supone cuatro años sin haber sido actualizado y adaptado a los cambios en el *firmware* de los diferentes módulos *XBee*® .
2. La estructura del código es compleja y costosa en términos de memoria y computación, ya que para resolver tareas sencillas, se realizan multitud de llamadas a funciones dentro de otras funciones, lo que supone, para un programa relativamente corto, ocupar una gran cantidad de memoria en la pila de llamadas de funciones (Figura 2.1).

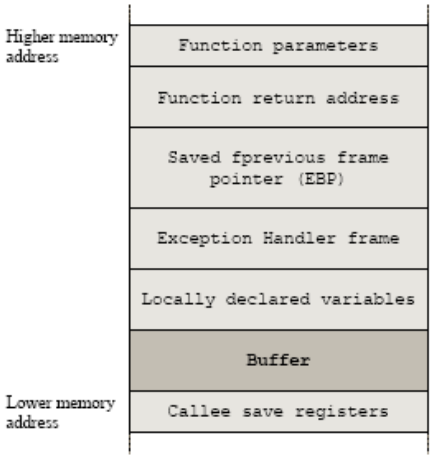


Figura 2.1 Ejemplo ilustrativo de la composición de la pila en memoria durante la llamada a una función en lenguaje C.

3. La librería, en su objetivo de ser portable al máximo número de plataformas diferentes posibles, hace uso **estrictamente** del estándar ANSI-C (admite características de C89/C90 y C99). Cabe destacar que no contempla ni hace uso de ninguna extensión del compilador, incluyendo aquellas propias de GNU. Además, se incluye en el código una "capa de abstracción *hardware*" para diferenciar entre las diferentes plataformas objetivo (Rabbit, Freescale, Win32, etc.).
4. Por la anterior razón, y debido a la inexistencia de un "Manual de Usuario", el uso de la librería resulta laborioso, poco intuitivo, y solo al alcance de usuarios con amplios conocimientos, tanto en programación, como del *firmware* y composición de las tramas y protocolos de los diferentes módulos *XBee*® existentes.
5. La librería carece de ejemplos prácticos de cómo usarla para recibir o transmitir datos de forma efectiva mediante el uso de tramas una vez configurados los *XBee*® en *API-mode*. Esto reduce su usabilidad y favorece la comisión de errores para usuarios con poca experiencia o conocimientos.
6. Se han encontrado multitud de comentarios en el código haciendo alusión a posibles mejoras o fallos a solucionar por los desarrolladores en futuras versiones del código (muchas de ellas aparecen anotados en los comentarios del código con la palabra clave *TODO*), así como ciertas funciones o características de determinadas tramas o protocolos que no han sido implementadas, instándose a desarrollar dicho código en el futuro (como por ejemplo, incluir métodos *reset/clear* en ciertas clases que son reutilizadas frecuentemente, o contemplar el uso de ciertas tramas que no han sido implementadas en la librería).
7. Una de las limitaciones más importantes de esta librería es que, para poder usarla, es necesario que los módulos RF *XBee*® estén configurados obligatoriamente en *API-mode* con caracteres de escape habilitados (modo que se selecciona configurando AP=2). Esto impide al usuario poder usar la librería sin caracteres de escape (*API-mode* con AP=1).
8. Se ha comprobado que gran parte de las funciones tienen caracter "universal", es decir, están programadas para aceptar cualquiera de los protocolos para los que se ha diseñado la librería *Digi XBee ANSI-C* (concretamente DigiMesh, *ZigBee*, o Smart Energy *firmware* en *API-mode*), lo cual es ineficiente, pues dada una topología de red y un protocolo para la misma (por ejemplo, ZigBee), se sabe con

exactitud el módulo *XBee®* que se va a utilizar, y por tanto, el tipo de código o librería especializada que sería necesario escoger (tal y como se hace en la plataforma *Waspmote*), siendo el código del resto de protocolos o soluciones, de nula utilidad.

Por todo lo anterior, se descarta el uso de la librería *Digi XBee ANSI-C* como base de estudio para este proyecto; no obstante, se tendrán en cuenta las diferentes razones aportadas anteriormente, transformándolas en requerimientos a evitar en la librería que se desarrollará.

2.4 Waspmote y Arduino: librería "avr-libc"

Resulta de gran interés y utilidad la plataforma *Waspmote*, que proporciona un IDE propio, así como una API [21] que contiene todas las librerías necesarias para compilar programas que hagan uso de módulos *XBee®* como interfaz de comunicación.

La primera versión *hardware* de esta plataforma fue lanzada en 2009 bajo el nombre comercial de ***Waspmote v1.1***, siendo ***Waspmote API v0.33*** (enero 2014) la última versión de código soportada por dicho *hardware*. En enero de 2013 se lanzó al mercado una versión revisada y mejorada del *hardware*, bajo el nombre de ***Waspmote Pro v1.2***[8] (Figura 2.2), siendo, en el momento de escribir este documento, ***Waspmote Pro API v0.23*** (publicada en mayo de 2016) la versión más reciente de la API, en continua actualización, mejora, y corrección de errores por parte de *Libelium*.

Una de las grandes diferencias que se han encontrado entre el conjunto de librerías *Waspmote API v0.33* y *Waspmote Pro API v0.23* es la significativa reducción de funciones (sin pérdida de funcionalidad) en la segunda de ellas, lo que redunda en un código más legible y fácil de seguir y entender, así como la acotación del número de llamadas a funciones que se cargan en la pila (Figura 2.1).

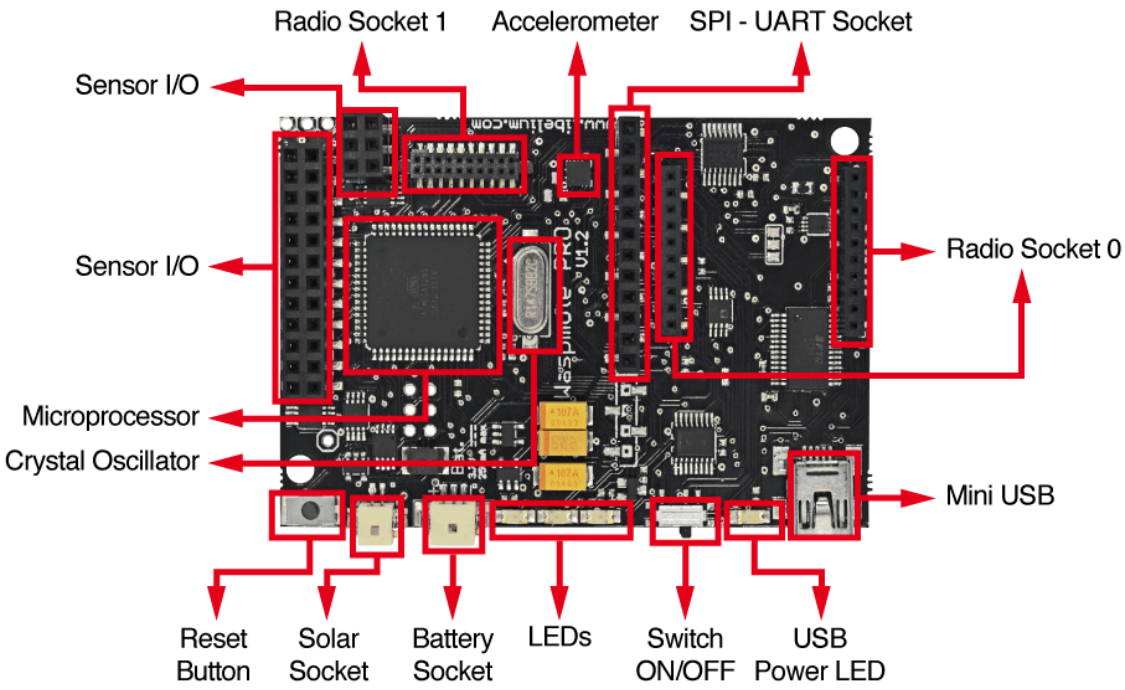
Concretamente, se han utilizado las librerías ***Waspmote Pro API*** (versiones v0.20 a v0.23) como base de estudio para la librería que se ha desarrollado en este proyecto, y más concretamente, las librerías necesarias para utilizar el módulo *XBee-PRO® 868*. Tanto el IDE (Entorno Integrado de Desarrollo) de Waspmote, la metodología de programación, como las librerías contenidas en la API, se asemejan mucho a las utilizadas por ***Arduino***, y no es casualidad: *Waspmote* utiliza el micro-controlador ATmega1281, de la empresa *Atmel*.

La plataforma *Arduino* utiliza para sus distintos modelos una arquitectura basada en los micro-controladores de la serie *ATmega* de *Atmel*, que hacen uso de la librería ***avr-libc***[3], la cual es, en esencia, un sub-conjunto *Open-Source* de librerías del lenguaje C estándar específicamente desarrollado para el compilador ***avr-gcc*** y micro-controladores AVR 8-bit RISC de la empresa Atmel. Esto fomenta que resulte atractivo el uso de *Waspmote* para la extensa red de usuarios y desarrolladores existentes a nivel mundial para *Arduino*, y que ya están familiarizados con el lenguaje de programación utilizado en dicha plataforma.

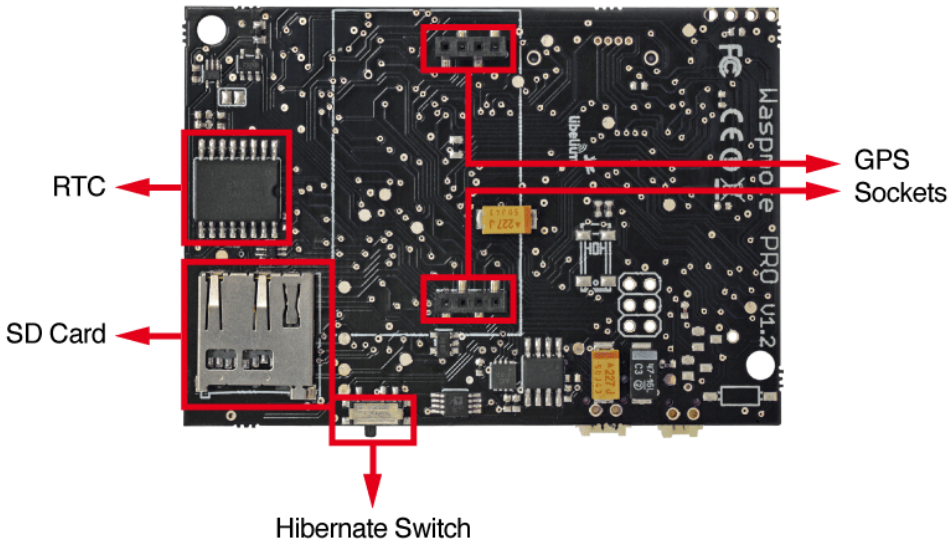
La librería *avr-libc* no contempla todas las características de estándares del lenguaje C, como *ANSI-C* o C99, y por tanto, no estarán disponibles todas ellas. No obstante, *avr-libc* incluye características de otros estándares, como *POSIX-1*, además de ciertas extensiones propias del *hardware* de dispositivos AVR. Todo esto supone una limitación importante a la hora de desarrollar una librería o API que pueda ser usada sobre otro tipo de *hardware*, como PC's o *Raspberry Pi* (arquitectura ARM). Por otro lado, aunque *avr-libc* soporta la programación en **lenguaje C++**, no soporta ***libstdc++***, la librería estándar necesaria para soportar una implementación completa de C++.

Por otra parte, es posible encontrar en la web de *Arduino* [1] una librería desarrollada bajo la licencia libre GNU por el usuario *Andrew Rapp* [2] que permite usar ciertos módulos *XBee* con el *hardware* de *Arduino*. Tal y como se ha descrito anteriormente, cualquier librería desarrollada para *Arduino* debe usar el entorno *avr-libc*; por tanto, se dispone actualmente de dos recursos, basados en *avr-libc*, pero programados en **lenguaje C++** (a pesar de que no cuenta con soporte completo para *libstdc++* como se ha comentado en el párrafo anterior), como base de estudio y análisis para este proyecto: *Waspmote Pro API* por un lado, y la librería desarrollada por *Andrew Rapp* para *Arduino* por otro.

[8] El código de las librerías *Waspmote API* (*hardware* v1.1) y *Waspmote Pro API* (*hardware* v1.2) no son compatibles entre sí; no obstante, el código de la v1.1 es posible "traducirlo" al v1.2 sin excesiva dificultad, pero no a la inversa.



(a) Vista Superior.



(b) Vista Inferior.

Figura 2.2 Principales elementos *hardware* de *Waspmote PRO v1.2*.

2.4.1 Librería XBee para Arduino

La librería fue creada por *Andrew Rapp* a lo largo de varios años (2009-2014), y está pensada para usarse con módulos *XBee*® basados en *hardware* tipo *SERIES 1* (S1) (módulos *XBee*® que inicialmente tenían soporte para el protocolo *IEEE 802.15.4*) y *SERIES 2* (S2) (módulos *XBee*® con soporte inicial sólo para el protocolo *ZigBee*). Los módulos S1 eran recomendables para usuarios que requerían un producto fácil de configurar y usar, mientras que los módulos S2 necesitaban un conocimiento más profundo del *firmware* de los *XBee*® y del protocolo *ZigBee*. Los módulos S1 y S2 no son compatibles entre sí.

Actualmente, la mencionada correspondencia entre el tipo de *hardware* (S1 o S2) y el protocolo soportado

ha cambiado: los módulos con *hardware* S1 dan soporte al protocolo DigiMesh® propietario de Digi, y los módulos S2 se han "descatalogado" como tal, y se han sustituido por los S2C (soporte tanto para el protocolo *ZigBee* como para el protocolo IEEE 802.15.4, simplemente "cambiando" el *firmware*, del módulo *XBee*®) y S2D (módulos con soporte para el protocolo *ZigBee* que pueden actualizarse al protocolo *Thread*)[9].

Además, al igual que ocurría con la librería *Digi XBee ANSI-C*, esta librería tampoco contempla el modo de funcionamiento API modo 1 (AP = 1, sin caracteres de escape), por lo que obliga al usuario a utilizar los módulos en modo API 2 (AP = 2).

Por otro lado, esta librería se compone exclusivamente de dos ficheros: *XBee.cpp* y *XBee.h*. En ellos, el código está estructurado en diversas clases base (*XBee*, *XBeeAddress*, *XBeeRequest*, y *XBeeResponse*), de las cuales heredan diversas sub-clases, existiendo hasta un total de 5 niveles de *herencia*. Al ser una librería optimizada para su uso sobre plataformas *Arduino*, contiene numerosas funciones de bajo nivel (control de pines, lectura de bytes, etc.) propias de las librerías *Arduino*.

2.4.2 Librería XBee® para Waspmote

El conjunto ***Waspmote Pro API*** está estructurado siguiendo una arquitectura modular, con multitud de librerías diferentes, organizadas en dos partes bien diferenciadas: una que contiene las librerías propias de la plataforma *Waspmote*, para su uso y configuración, tanto como dispositivo *stand-alone* como plataforma base de otros dispositivos (entre los que se incluye *XBee*® , cuya librería base asociada es *WaspXBeeCore.h*), y otra parte que contiene las librerías necesarias para hacer uso de los muchos dispositivos comerciales (*Sensor Boards*, interfaces de comunicación, etc.) compatibles con la plataforma *Waspmote*. Entre ellas, se encuentran una serie de librerías para el uso de diferentes protocolos y módulos *XBee*® . En la Figura 2.3 puede observarse el gráfico de dependencias de la librería de *Waspmote* para los módulos *XBee-PRO*®*868* (gráfico obtenido de la versión *online* [20] de la Documentación de la Librería *Waspmote Pro API*).

Si se revisa el código de dichas librerías, puede observarse la existencia de una clase base, *WaspXBeeCore*, de la que hereda la librería *WaspXBee868*, particularizada para hacer uso del protocolo de los módulos *XBee-PRO*®*868*, sin existir más niveles de *herencia* de clases. Al igual que otras librerías estudiadas, esta librería tampoco soporta el modo de funcionamiento API sin caracteres de escape (AP = 1), obligando al uso del modo AP = 2 (API con caracteres de escape). Todas las librerías de *Waspmote Pro API* están adaptadas al *hardware* de la plataforma *Waspmote*.

[9] Puede obtenerse un cuadro-resumen de las principales características de los diferentes módulos *XBee*®, incluyendo el tipo de *hardware* y protocolo asociado a cada módulo, en la dirección *http://www.digi.com/pdf/chart_xbee_rf_features.pdf*

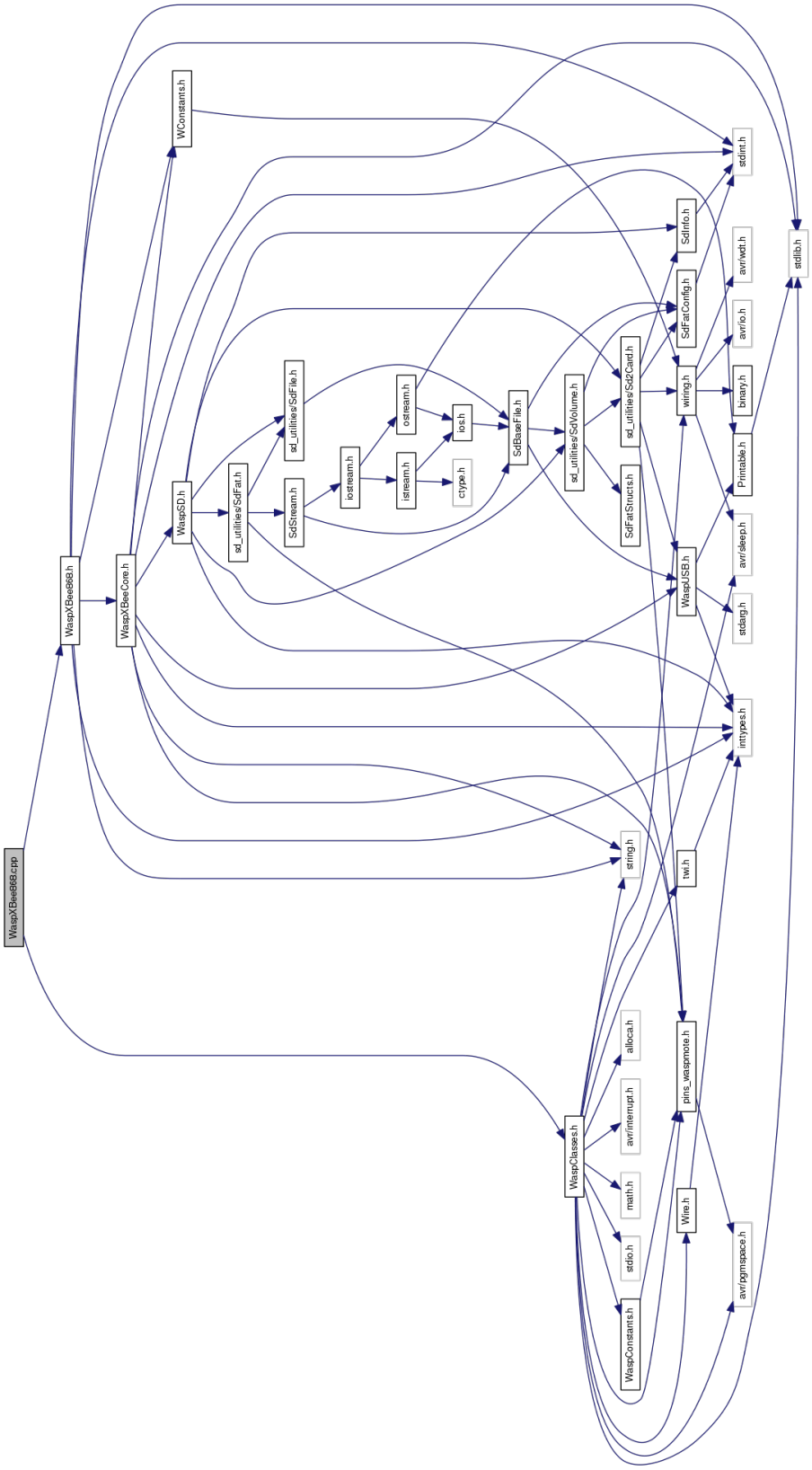


Figura 2.3 Gráfico de dependencias de librerías del fichero WaspXBee868.cpp (generado por Libelium).

Parte II.

Guía de Usuario y Documentación del Código

3 Protocolo XBee-PRO® 868: tramas "API-mode"

En este capítulo se realiza la descripción de las diferentes tramas soportadas por el protocolo de los módulos *XBee-PRO® 868*, necesario para entender y comprender el funcionamiento y objetivo de cada una de las partes (clases, funciones, etc.) que conforman el código de la librería *UniversalXBee*. En el siguiente capítulo (Capítulo 4 se encuentra la documentación, generada mediante la herramienta *Doxygen*, de la librería que se ha desarrollado en este proyecto.

Por simplicidad en los ejemplos de este capítulo, se ha considerado que el módulo *XBee®* está trabajando en *API-mode 1* (AP = 1), es decir, sin caracteres de escape habilitados. Cuando se activa el modo *API-mode 2* (Figura 3.1, con caracteres de escape habilitados, AP = 2), los bytes que puedan interferir en la interpretación de los datos deben incluir un carácter de escape, insertando el byte 0x7D seguido del byte que necesita el carácter de escape, calculado como *("byte non-escaped" XOR 0x20)*[1]. De este modo, los bytes 0x11 (XON), 0x13 (XOFF), 0x7D (Carácter Escape), y 0x7E (Inicio de Trama), se sustituyen por la siguiente secuencia:

- *0x11 XOR 0x20 = 0x31 –> 0x11 = 0x7D + 0x31*
- *0x13 XOR 0x20 = 0x33 –> 0x13 = 0x7D + 0x33*
- *0x7D XOR 0x20 = 0x5D –> 0x7D = 0x7D + 0x5D*
- *0x7E XOR 0x20 = 0x5E –> 0x7E = 0x7D + 0x5E*

Ejemplo: El siguiente ejemplo se ha obtenido de la documentación de los módulos *XBee®* .
Trama de datos UART (antes de incluir caracteres de escape): *0x7E 0x00 0x02 0x23 0x11 0xCB*.
El byte 0x11 necesita incorporar un carácter de escape, tras lo cual, la trama se compondrá de los siguientes bytes: *0x7E 0x00 0x02 0x23 0x7D 0x31 0xCB*
En la serie de bytes anterior, la longitud de los datos (excluyendo el checksum) es 0x0002 y el checksum de los datos sin caracteres de escape (excluyendo el byte "Inicio de Trama" y el campo "longitud") es calculado como: $0xFF - (0x23 + 0x11) = (0xFF - 0x34) = 0xCB$.

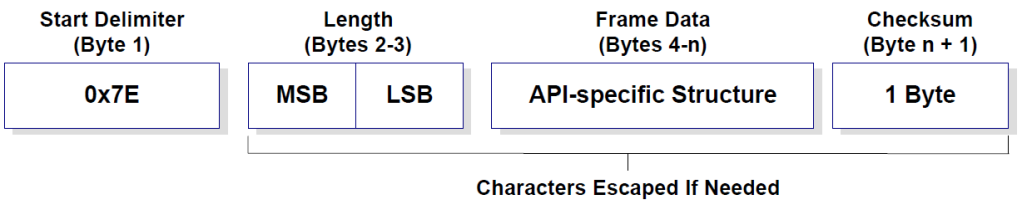


Figura 3.1 Ejemplo gráfico de los bytes que pueden incorporar caracteres de escape si AP = 2. (Imagen extraída de la documentación de los módulos *XBee®*).

[1] XOR (*Exclusive OR*), es una operación lógica a nivel de bits equivalente a la suma en módulo 2: la salida sólo es 1 si las entradas son diferentes, y sólo es 0 si las entradas son iguales.

Aquellos campos cuyo valor es fijo y no puede ser modificado por el usuario ni depende de otros campos (como el byte de inicio 0*x7E*), han sido resaltados en negrita. Para obtener información adicional sobre el protocolo (tramas, comandos, características técnicas, etc.) que usan los módulos *XBee-PRO® 868*, se insta al lector a consultar el Manual de Usuario del Producto [22].

En todos los ejemplos, se han utilizado dos módulos RF *XBee®* , uno que siempre inicia la transmisión, que se ha denominado **NodoA**, o nodo conectado en local (denominado *local* por el hecho de que, en los ejemplos, será el nodo que inicia la comunicación y al cual es posible conectarse y acceder físicamente a su UART para enviar y recibir datos a través de un puerto serie), y otro nodo conectado en remoto, denominado **NodoB**, que recibe por RF las transmisiones/peticiones iniciadas por NodoA y devuelve la respuesta correspondiente a la petición recibida. Para facilitar el seguimiento de los ejemplos, se ilustra a continuación (Figura 3.2, Figura 3.3 y Figura 3.4) un resumen básico del intercambio de tramas que se realiza entre ambos nodos a nivel de UART:

- **NodoA** (local) - 64-bits MAC address: "*00 13 A2 00 40 3A AE ED*".
- **NodoB** (remoto) - 64-bits MAC address: "*00 13 A2 00 40 3A B3 10*".

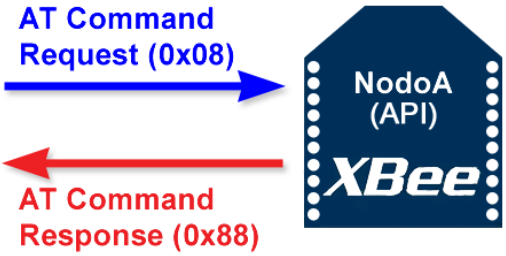


Figura 3.2 Ejemplo gráfico del intercambio de tramas *AT Command* con un nodo local.

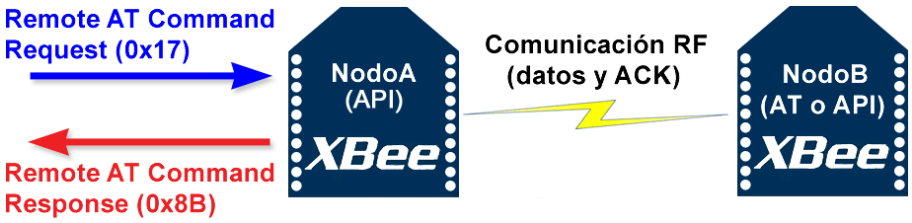


Figura 3.3 Ejemplo gráfico del intercambio de tramas *AT Command* con un nodo remoto.

Figura 3.4 Ejemplo gráfico del intercambio de tramas de datos con un nodo remoto.

3.1 Trama 0x08: AT Command Request

Esta trama es usada para configurar o leer los parámetros del módulo *XBee*® conectado en local (no en remoto). Esta trama aplica los cambios inmediatamente después de ejecutar el comando (los cambios efectuados en cualquier parámetro del módulo sólo se tienen en cuenta una vez son aplicados).

Ejemplo: la Tabla 3.1 ilustra los campos de la trama tipo 0*x*08: *AT Command*. En este ejemplo, se realiza la petición de leer los datos contenidos en el parámetro "NI" (*Node Identifier*, o nombre identificador del nodo: contiene una cadena ASCII de un máximo de 20 carácteres que identifica el módulo *XBee*®). Se envían los siguientes datos en hexadecimal:
"*7E 00 04 08 01 4E 49 5F*".

Tabla 3.1 Frame type = 0*x*08. Trama Tx: "AT Command Request".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre 0*x*7*E*.
Length	1 (MSB) 2 (LSB)	0x00 0x04 (4)	Número de bytes entre *length* y *checksum* (ambos excluidos)
Frame type	3	**0x08**	Identifica el tipo de trama.
Frame ID	4	0x01	Identifica la trama de datos UART del host para relacionarla con el correspondiente ACK (*acknowledgment*). Si *FrameID* = 0*x*00, no se enviará respuesta (ACK).
AT command	5 6	0x4E (N) 0x49 (I)	Nombre del comando (siempre dos carácteres ASCII que identifican el comando AT).
Command parameter value (opcional)			Si está presente (esto incluye el valor 0*x*00), este campo indica el valor que debe asignarse en el registro correspondiente al comando del campo *AT command*. Si este campo no se envía (como en este ejemplo), indica que se está consultado el valor de dicho registro o parámetro.
Checksum	7	0x5F	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser 0*xFF*.

3.2 Trama $0x10$: Transmit Request

Esta trama provoca que el módulo *XBee*® conectado en local empaquete los datos presentes en el campo *Payload data* y los envíe por RF al nodo destino (transmisión *unicast*) especificado en el campo *64-bit Destination Address*. Si se desea realizar una transmisión *broadcast*, dicho campo deberá contener la dirección "00 00 00 00 00 00 FF FF". En respuesta al envío de esta trama, siempre se deberá recibir una trama tipo 0*x*8*B* (*Transmit Status*; ver Página 24, Sección 3.7).

Ejemplo: la Tabla 3.2 ilustra los campos de la trama tipo 0*x*10: *Transmit Request*. En este ejemplo, se muestra como realizar una transmisión *unicast* al nodo destino con dirección MAC "00 13 A2 00 40 3A B3 10" para enviar los datos contenidos en el campo *Payload Data*: "Test". Por tanto, la trama está compuesta por los siguientes datos en hexadecimal:
"*7E 00 12 10 15 00 13 A2 00 40 3A B3 10 FF FE 00 00 54 65 73 74 4B*".

Tabla 3.2 Frame type = 0*x*10. Trama Tx: "Transmit Request".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre $0x7E$.
Length	1 (MSB) 2 (LSB)	0x00 0x12 (18)	Número de bytes entre *length* y *checksum* (ambos excluidos).
Frame type	3	**0x10**	Identifica el tipo de trama.
Frame ID	4	0x15	Identifica la trama de datos UART del host para relacionarla con el correspondiente ACK (*acknowledgment*). Si $FrameID = 0x00$, no se enviará respuesta (ACK).
64-bit destination address (MAC)	5 (MSB) 6 7 8 9 10 11 12 (LSB)	0x00 0x13 0xA2 0x00 0x40 0x3A 0xB3 0x10	Identifica la dirección MAC del nodo/dispositivo de destino de la trama. Se admite la dirección "00 00 00 00 00 00 FF FF" para indicar la dirección de *broadcast*.
Reserved	13 14	**0xFF** **0xFE**	Campo reservado. Siempre $0xFFFE$.
Broadcast radius	15	0x00	Indica el número máximo de saltos permitidos en una transmisión tipo *broadcast*. Si se establece a 0*x*00 (valor por defecto), *broadcast radius* se configurará para dar el máximo número de saltos.
Transmit options	16	0x00	Bitfield (valor por defecto: 0*x*00): bit0 = 1 (0*x*01): ACK desactivado. bit1 = 1 (0*x*02): No realizar "route discovery". Los demás bits deben ser 0.
RF Payload data	17 18 19 20	0x54 0x65 0x73 0x74	Datos que se envían al nodo/dispositivo de destino. En este ejemplo, este campo contiene el valor *"54 65 73 74"* (en ASCII: *Test*).
Checksum	21	0x4B	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

3.3 Trama $0x11$: Explicit Addressing Command Frame

Este tipo de trama (trama con direccionamiento explicito) permite que se puedan especificar campos que son útiles a nivel de aplicación. Esta trama es muy parecida a la trama tipo 0*x*10 (*Transmit Request*; ver Página 18, Sección 3.2), pero requiere que dichos campos de direccionamiento a nivel de aplicación sean especificados (*Endpoints, Cluster ID, Profile ID*).

Una trama de este tipo provoca que el módulo *XBee*® conectado en local empaquete los datos presentes en el campo *Payload data* y los envíe por RF al nodo destino (transmisión *unicast*) especificado en el campo *64-bit Destination Address*, pero usando los nodos extremos origen/destino (*endpoints*) de la comunicación, el *Cluster ID*, y el *Profile ID* especificados. Si se desea realizar una transmisión *broadcast* (a todos los dispositivos de la red entre los dos nodos extremos), el campo de dirección deberá contener "00 00 00 00 00 00 FF FF". En respuesta al envío de esta trama, siempre se deberá recibir una trama tipo 0*x*8*B* (*Transmit Status*, Página 24, Sección 3.7). Igualmente a otras tramas, el campo reservado deberá contener el valor $0xFFFE$.

El parámetro *broadcast radius* (solo aplicable a transmisiones *broadcast*) establece el máximo número de saltos permitidos, a través de diferentes nodos de la red, que una transmisión de datos *broadcast* puede alcanzar, cuyo valor estará comprendido entre un mínimo de 0 (valor por defecto) y un máximo de 0xFF (255). Por otra parte, el resto de campos adquieren los siguientes valores por defecto:

- Cluster ID = 0x11 (**Transparent mode**).
- Source/Destination Endpoint = 0xE8 (Digi data endpoint).
- Profile ID = 0xC105.

Este tipo de tramas también pueden usarse para testar el rendimiento de un link RF entre dos nodos, enviando datos de prueba en una transmisión *unicast* entre dos dispositivos (origen y destino), determinando el ratio de transmisiones exitosas entre ambos nodos. Para simplificar este proceso, denominado *link-testing*, los módulos *XBee*® soportan un modo especial de funcionamiento, que se activa automáticamente al configurar Cluster ID = 0x12 (*loop-back cluster*) sobre los nodos extremos origen/destino tipo Digi (*endpoints* = 0xE8). En este modo, cualquier dato enviado en el campo *payload* de la trama será retransmitido hacía atrás, devolviendo una copia de los datos en *payload* al nodo origen de la transmisión.

- Cluster ID = 0x12 (**Link-testing mode**).
- Source/Destination Endpoint = 0xE8 (Digi data endpoint).
- Profile ID = 0xC105.

Ejemplo: la Tabla 3.3 ilustra los campos de la trama tipo 0*x*11: *Explicit Addressing Command Frame*. En este ejemplo, se muestra como realizar una transmisión *unicast* al nodo destino con dirección MAC "00 13 A2 00 40 3A B3 10" para enviar los datos contenidos en el campo *Payload Data*: "Test", con *Cluster ID = 0xABCD*, *Profile ID = 0x1234*, y con extremos (*endpoints*) origen 0xA0 y destino 0xA1. Por tanto, la trama está compuesta por los siguientes datos en hexadecimal:
"*7E 00 18 11 01 00 13 A2 00 40 3A B3 10 FF FE A0 A1 AB CD 12 34 00 00 54 65 73 74 5F*".

Tabla 3.3 Frame type = 0*x*11. Trama Tx: "Explicit Addressing Command Frame".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre 0*x*7*E*.
Length	1 (MSB) 2 (LSB)	0x00 0x18 (24)	Número de bytes entre *length* y *checksum* (ambos excluidos).
Frame type	3	**0x11**	Identifica el tipo de trama.
Frame ID	4	0x01	Identifica la trama de datos UART del host para relacionarla con el correspondiente ACK (*acknowledgment*). Si $FrameID = 0x00$, no se enviará respuesta (ACK).
64-bit destination address (MAC)	5 (MSB) 6 7 8 9 10 11 12 (LSB)	0x00 0x13 0xA2 0x00 0x40 0x3A 0xB3 0x10	Identifica la dirección MAC del nodo/dispositivo de destino de la trama. Se admite la dirección "00 00 00 00 00 00 FF FF" para indicar la dirección de *broadcast*.
Reserved	13 14	**0xFF** **0xFE**	Campo reservado. Siempre $0xFFFE$.
Source Endpoint	15	0xA0	Nodo extremo que originó la transmisión del mensaje. Valor por defecto: 0*xE*8.
Destination Endpoint	16	0xA1	Nodo extremo al cual se direcciona el mensaje como destino final. Valor por defecto: 0*xE*8.
Cluster ID	17 18	0xAB 0xCD	Cluster ID usado en la transmisión. Valor por defecto: 0*x*11 (*transparent data mode*).
Profile ID	19 20	0x12 0x34	Profile ID usado en la transmisión. Valor por defecto: 0*xC*105.
Broadcast radius	15	0x00	Indica el número máximo de saltos permitidos en una transmisión tipo *broadcast*. Si se establece a 0*x*00 (valor por defecto), *broadcast radius* se configurará para dar el máximo número de saltos.
Transmit options	16	0x00	Bitfield (valor por defecto: 0*x*00): bit0 = 1 (0*x*01): ACK desactivado. bit1 = 1 (0*x*02): No realizar "descubrimiento de ruta". Los demás bits deben ser 0.
RF Payload data	17 18 19 20	0x54 0x65 0x73 0x74	Datos que se envían al nodo/dispositivo de destino. En este ejemplo, este campo contiene el valor *"54 65 73 74"* (en ASCII: *Test*).
Checksum	21	0x5F	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

3.4 Trama $0x17$: Remote AT Command Request

Esta trama es usada para configurar o leer de forma remota los parámetros de otro módulo *XBee*® . Para que esta trama aplique los cambios después de ejecutar el comando (los cambios efectuados en cualquier parámetro del módulo sólo se tienen en cuenta una vez son aplicados), se deberá, o bien activar el bit correspondiente (campo *Remote command options* = 0*x*02), o bien enviar posteriormente al mismo nodo remoto una trama con el comando "AC" (*Apply Changes*).

Ejemplo: la Tabla 3.4 ilustra los campos de la trama tipo 0*x*17: *Remote AT Command Request*. En este ejemplo, se configura de forma remota el parámetro "SP" (*Sleep Time*) del nodo con dirección MAC "00 13 A2 00 40 3A B3 10", configurándolo al valor 0*x*03*E*8 (que equivale a 10000ms), indicando que se apliquen inmediatamente los cambios (campo *Remote command options* = 0*x*02). Se envían los siguientes datos en hexadecimal:
"*7E 00 11 17 01 00 13 A2 00 40 3A B3 10 FF FE 02 53 50 03 E8 68*".

Tabla 3.4 Frame type = 0*x*17. Trama Tx: "Remote AT Command Request".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre $0x7E$.
Length	1 (MSB) 2 (LSB)	0x00 0x11 (17)	Número de bytes entre *length* y *checksum* (ambos excluidos).
Frame type	3	**0x17**	Identifica el tipo de trama.
Frame ID	4	0x01	Identifica la trama de datos UART del host para relacionarla con el correspondiente ACK (*acknowledgment*). Si $FrameID = 0x00$, no se enviará respuesta (ACK).
64-bit destination address (MAC)	5 (MSB) 6 7 8 9 10 11 12 (LSB)	0x00 0x13 0xA2 0x00 0x40 0x3A 0xB3 0x10	Identifica la dirección MAC del nodo/dispositivo de destino de la trama. Se admite la dirección "00 00 00 00 00 00 FF FF" para indicar la dirección de *broadcast*.
Reserved	13 14	**0xFF** **0xFE**	Campo reservado. Siempre $0xFFFE$.
Remote command options	15	0x02	Bitfield (valor por defecto = 0*x*02): bit0 = 1 (0*x*01): ACK desactivado. bit1 = 1 (0*x*02): Aplicar cambios inmediatamente en el dispositivo remoto (si no se indica este valor de 0*x*02, se deberá enviar posteriormente en otra trama el comando "AC" para hacer efectivas las nuevas configuraciones y valores). Los demás bits deben ser 0.
AT command	16 17	0x53 (S) 0x50 (P)	Nombre del comando (siempre dos carácteres ASCII que identifican el comando AT).
Command parameter value (opcional)	18 19	0x03 0xE8	Si está presente (esto incluye el valor 0*x*00), este campo indica el valor que debe asignarse en el registro correspondiente al comando del campo *AT command*. Si este campo no se envía, indica que se está consultado el valor de dicho registro o parámetro.
Checksum	20	0x68	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

3.5 Trama $0x88$: AT Command Response

Esta trama se recibe siempre desde la UART del módulo conectado en local, y **siempre en respuesta** a una trama del tipo $0x08$ (*AT Command Request*; ver Página 17, Sección 3.1) previamente enviada a través de la UART hacia el propio módulo local. Algunos comandos (como el comando "ND" *Node Discovery*) pueden provocar el envío de varias tramas para completar la respuesta.

Ejemplo: la Tabla 3.5 ilustra los campos de la trama tipo $0x88$: *AT Command Response*. Suponiendo que se ha realizado previamente una petición, en una trama con *Frame ID* = $0x01$, de leer el valor del parámetro "NI" (ejemplo Tabla 3.1), si la petición resulta exitosa, se recibirán los siguientes datos en hexadecimal: "*7E 00 0A 88 01 4E 49 00 4E 6F 64 6F 41 0E*".

Tabla 3.5 Frame type = $0x88$. Trama Rx: "AT Command Response".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre $0x7E$.
Length	1 (MSB) 2 (LSB)	0x00 0x0A (10)	Número de bytes entre *length* y *checksum* (ambos excluidos).
Frame type	3	**0x88**	Identifica el tipo de trama.
Frame ID	4	0x01	Identifica la trama de datos UART que se reporta (ACK), conteniendo el mismo valor que se envío en la petición (trama $0x08$). Nota: Si $FrameID = 0x00$ en AT-mode, no se enviará respuesta (ACK).
AT command	5 6	0x4E (N) 0x49 (I)	Nombre del comando (siempre dos carácteres ASCII que identifican el comando AT).
Command status	7	0x00	0x00 = OK. 0x01 = ERROR. 0x02 = Comando inválido. 0x03 = Parámetro inválido.
Command data (opcional)	8 9 10 11 12	0x4E 0x6F 0x64 0x6F 0x41	Valor (en binario, no ASCII) contenido en el registro correspondiente al comando del campo *AT command*. Si este campo no se envía, indica que en la trama tipo $0x08$ se ha configurado algún valor en dicho registro. En este ejemplo, este campo contiene el valor *"4E 6F 64 6F 41"* (en ASCII: *NodoA*).
Checksum	13	0x0E	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

3.6 Trama $0x8A$: Modem Status

Este tipo de tramas son enviadas por el módulo en respuesta ante determinadas circunstancias, especificadas por el campo *Status* de la trama. **Ejemplo:** la Tabla 3.6 ilustra los campos de la trama tipo $0x8A$: *Modem Status*. La trama de este ejemplo es enviada cuando un dispositivo reactiva su alimentación. La trama $0x8A$ consta de los siguientes datos en hexadecimal:
"*7E 00 02 8A 00 75*".

Tabla 3.6 Frame type = $0x8A$. Trama Rx: "Modem Status".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre $0x7E$.
Length	1 (MSB) 2 (LSB)	0x00 0x02 (2)	Número de bytes entre *length* y *checksum* (ambos excluidos)
Frame type	3	**0x8A**	Identifica el tipo de trama.
Status	4	0x00	0x00 = Hardware reset. 0x01 = Watchdog timer reset. 0x0B = La red se ha reactivo/despertado. 0x0C = La red entra en modo de bajo consumo (sleep).
Checksum	5	0x75	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

3.7 Trama $0x8B$: Transmit Status

Siempre que se completa la transmisión de datos a un nodo en remoto, como por ejemplo de tramas tipo $0x10$ (*Transmit Request*; ver Página 18, Sección 3.2) o tipo $0x11$ (*Explicit Addressing Command Frame*; ver Página 19, Sección 3.3), el módulo local envía una trama tipo $0x8B$ (*Transmit Status*) informando de si el paquete RF ha sido transmitido con éxito o si hubo algún error en la transmisión.

Ejemplo: la Tabla 3.7 ilustra los campos de la trama tipo $0x8B$: *Transmit Status*. En este ejemplo, se informa de que la transmisión *unicast* al nodo destino de una trama con identificador *Frame ID = 0x15* se ha considerado fallida tras haber realizado hasta 10 (0x0A) intentos/re-transmisiones de la trama sin recibir un ACK del nodo destino. La trama $0x8B$ consta de los siguientes datos en hexadecimal:
"*7E 00 07 8B 15 FF FE 0A 01 00 57*".

Tabla 3.7 Frame type = $0x8B$. Trama Rx: "Transmit Status".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre $0x7E$.
Length	1 (MSB) 2 (LSB)	0x00 0x07 (7)	Número de bytes entre *length* y *checksum* (ambos excluidos)
Frame type	3	**0x8B**	Identifica el tipo de trama.
Frame ID	4	0x15	Identifica la trama de datos UART del host para relacionarla con el correspondiente ACK (*acknowledgment*). Si $FrameID = 0x00$, no se enviará respuesta (ACK).
Reserved	5 6	**0xFF** **0xFE**	Campo reservado. Siempre $0xFFFE$.
Transmit retry count	7	0x0A (10)	Número total de re-transmisiones a nivel de aplicación que se han realizado hasta completar con éxito el envío de la trama, o agotar el número máximo de intentos (parámetro RR en *unicast*, parámetro MT en *broadcast*).
Delivery status	8	0x01	0x00 = Transmisión exitosa. 0x01 = No se ha recibido ACK. 0x15 = *Endpoint* de destino no válido. 0x21 = Error en la red. 0x25 = Ruta no encontrada.
Discovery status	9	0x00	0x00 = Descubrimiento de ruta no activa. 0x02 = Descubrimiento de ruta.
Checksum	10	0x57	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

3.8 Trama $0x90$: Receive Packet

Cuando el módulo recibe un paquete por RF, los datos que este paquete contiene son enviados a través de la UART del módulo usando una trama tipo $0x90$ (si el parámetro $AO = 0$), independientemente de si la trama recibida es de tipo $0x10$ (*Transmit Request*; ver Página 18, Sección 3.2) o de tipo $0x11$ (*Explicit Addressing Command Frame*; ver Página 19, Sección 3.3).

Ejemplo: la Tabla 3.8 ilustra los campos de la trama tipo $0x90$: *Receive Packet*. En este ejemplo, un dispositivo origen, con dirección MAC "00 13 A2 00 40 3A AE ED", realiza una transmisión *unicast* a un dispositivo remoto, con los siguientes datos contenidos en el campo *Payload Data*: "Test". Si en el nodo destino que recibe dicha transmisión está configurado $AO = 0$, el dispositivo enviará a través de su UART una trama compuesta por los siguientes datos en hexadecimal:
"*7E 00 10 90 00 13 A2 00 40 3A AE ED FF FE 01 54 65 73 74 07*".

Tabla 3.8 Frame type = $0x90$. Trama Rx: "Receive Packet".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre $0x7E$.
Length	1 (MSB) 2 (LSB)	0x00 0x10 (16)	Número de bytes entre *length* y *checksum* (ambos excluidos).
Frame type	3	**0x90**	Identifica el tipo de trama.
64-bit source address (MAC)	4 (MSB) 5 6 7 8 9 10 11 (LSB)	0x00 0x13 0xA2 0x00 0x40 0x3A 0xAE 0xED	Identifica la dirección MAC del nodo/dispositivo de origen de la trama. Se admite la dirección "00 00 00 00 00 00 FF FF" para indicar la dirección de *broadcast*.
Reserved	12 13	**0xFF** **0xFE**	Campo reservado. Siempre $0xFFFE$.
Receive options	14	0x01	0x01 = Se envío ACK (asentimiento) del paquete recibido. 0x02 = El paquete fue una transmisión *broadcast*.
RF Payload data	15 16 18 19	0x54 0x65 0x73 0x74	Datos que se reciben desde el nodo/dispositivo de origen de la transmisión. En este ejemplo, este campo contiene el valor *"54 65 73 74"* (en ASCII: *Test*).
Checksum	20	0x07	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

3.9 Trama $0x91$: Explicit Rx Indicator

Cuando el módulo recibe un paquete por RF, los datos que este paquete contiene son enviados a través de la UART del módulo usando una trama tipo $0x91$ (si el parámetro $AO = 1$), independientemente de si la trama recibida es de tipo $0x10$ (*Transmit Request*; ver Página 18, Sección 3.2) o de tipo $0x11$ (*Explicit Addressing Command Frame*; ver Página 19, Sección 3.3).

Ejemplo: la Tabla 3.9 ilustra los campos de la trama tipo $0x91$: *Explicit Rx Indicator*. En este ejemplo, un dispositivo origen con dirección MAC "00 13 A2 00 40 3A AE ED", envía por RF la trama $0x11$ de la Sección 3.3, realizando una transmisión *unicast* a un dispositivo remoto, con los siguientes datos contenidos en el campo *Payload Data*: "Test", con *Cluster ID = 0xABCD*, *Profile ID = 0x1234*, y con extremos (*endpoints*) origen $0xA0$ y destino $0xA1$. Si en el nodo destino que recibe dicha transmisión está configurado $AO = 1$, el dispositivo enviará a través de su UART una trama compuesta por los siguientes datos en hexadecimal: "*7E 00 16 91 00 13 A2 00 40 3A AE ED FF FE A0 A1 AB CD 12 34 01 54 65 73 74 07*".

Tabla 3.9 Frame type = $0x91$. Trama Rx: "Explicit Rx Indicator".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre $0x7E$.
Length	1 (MSB) 2 (LSB)	0x00 0x16 (22)	Número de bytes entre *length* y *checksum* (ambos excluidos).
Frame type	3	**0x91**	Identifica el tipo de trama.
64-bit source address (MAC)	4 (MSB) 5 6 7 8 9 10 11 (LSB)	0x00 0x13 0xA2 0x00 0x40 0x3A 0xAE 0xED	Identifica la dirección MAC del nodo/dispositivo de origen de la trama. Se admite la dirección "00 00 00 00 00 00 FF FF" para indicar la dirección de *broadcast*.
Reserved	12 13	**0xFF** **0xFE**	Campo reservado. Siempre $0xFFFE$.
Source Endpoint	14	0xA0	Nodo extremo que originó la transmisión del mensaje. Valor por defecto: $0xE8$.
Destination Endpoint	15	0xA1	Nodo extremo al cual se direcciona el mensaje como destino final. Valor por defecto: $0xE8$.
Cluster ID	16 17	0xAB 0xCD	Cluster ID usado en la transmisión. Valor por defecto: $0x11$ (*transparent data mode*).
Profile ID	18 19	0x12 0x34	Profile ID usado en la transmisión. Valor por defecto: $0xC105$.
Receive options	20	0x01	0x01 = Se envío ACK (asentimiento) del paquete recibido. 0x02 = El paquete fue una transmisión *broadcast*.
RF Payload data	15 16 18 19	0x54 0x65 0x73 0x74	Datos que se reciben desde el nodo/dispositivo de origen de la transmisión. En este ejemplo, este campo contiene el valor *"54 65 73 74"* (en ASCII: *Test*).
Checksum	25	0x07	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

3.10 Trama $0x92$: I/O Data Sample Rx Indicator

Esta trama sólo se recibe si se ha configurado el módulo *XBee*® para realizar de forma automática un muestreo analógico (ADC) o digital, usando las opciones Sample Rate (IR) o Change Detect (IC). En caso de "forzar" la toma de muestras mediante el envío de un comando AT tipo "IS" mediante la trama $0x17$ (*Remote AT Command Request*; ver Página 21, Sección 3.4), los datos se recibirán en una trama tipo $0x97$ (*Remote AT Command Response*; ver Página 29, Sección 3.11), en cuyo caso, los datos correspondientes a los campos de la trama $0x92$ "*Number of Samples*", "*Digital Channel Mask*", "*Analog Channel Mask*", "*Digital Samples*" y "*Analog Samples*" estarán contenidos en el campo "Command Data" de la trama $0x97$. Más información [24].

Ejemplo: la Tabla 3.13 ilustra los campos de la trama tipo $0x92$: *I/O Data Sample Rx Indicator*. La Tabla 3.13 muestra los bytes recibidos en una transmisión *unicast* de una trama tipo $0x92$, desde un dispositivo remoto con dirección MAC "00 13 A2 00 40 3A B3 10", si se reciben 2 muestras analógicas; se recibirán los siguientes datos en hexadecimal:
"*7E 00 16 92 00 13 A2 00 40 3A B3 10 FF FE 01 01 00 18 03 00 10 02 2F 01 FE 21*".

El campo "*Digital Channel Mask*" es 0x0018 = 00011000b. Por lo tanto, las líneas DIO-3 y DIO-4 están activas como entradas digitales en el módulo *XBee*® que toma las muestras. El campo "*Digital Samples*" es 0x0010 = 00010000b. Por lo tanto, DIO-4 es HIGH (valor 1), y DIO-3 es LOW (valor 0).

El campo "*Analog Channel Mask*" es 0x03 = 0011b. Por lo tanto, las líneas AD-0 y AD-1 están activas como entradas analógicas (ADC) en el módulo *XBee*® que toma las muestras. Como resultado, se sabe que dos muestras analógicas (de 2-bytes de tamaño cada una) serán incluidas en la trama. ADC-0 será la primera muestra, de valor 0x022F, y ADC-2 la segunda muestra, de valor 0x01FE.

Para convertir dichos valores analógicos en medidas útiles, se debe tener en cuenta que todos los *XBee*® tienen un ADC de 10-bits de resolución, por lo que el rango aceptable de medidas estará comprendido entre 0x0000 y 0x03FF. Para convertir dicha medida en un nivel de voltaje, sólo es necesario aplicar la siguiente fórmula: $ADC/1023(Vref) = Voltaje$.

En la Tabla 3.10 y la Tabla 3.11 se muestran la correspondencia entre bits y entradas digitales a las que representan, aplicable para los campos "*Digital Channel Mask*" y "*Digital Samples*".

Tabla 3.10 Byte superior de los campos "Digital Channel Mask" y "Digital Samples" .

Bit-15 (MSB)	Bit-14	Bit-13	Bit-12	Bit-11	Bit-10	Bit-9	Bit-8
0	0	0	1	1	1	1	1
N/A	N/A	N/A	DIO-12	DIO-11	DIO-10	DIO-9	DIO-8

Tabla 3.11 Byte inferior de los campos "Digital Channel Mask" y "Digital Samples".

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0 (LSB)
1	1	1	1	1	1	1	0
DIO-7	DIO-6	DIO-5	DIO-4	DIO-3	DIO-2	DIO-1	DIO-0

En la Tabla 3.12 se muestra la correspondencia entre bits y entradas analógicas a las que representan, aplicable para los campos "*Analog Channel Mask*" y "*Analog Samples*".

Tabla 3.12 Campo "Analog Channel Mask" y "*Analog Samples*".

Bit-7 (MSB)	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0 (LSB)
1	0	0	0	1	1	1	0
Vdd	N/A	N/A	N/A	ADC-3	ADC-2	ADC-1	ADC-0

Tabla 3.13 Frame type = 0*x*92. Trama Rx: "I/O Data Sample Rx Indicator".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre 0*x*7*E*.
Length	1 (MSB) 2 (LSB)	0x00 0x11 (17)	Número de bytes entre *length* y *checksum* (ambos excluidos).
Frame type	3	**0x92**	Identifica el tipo de trama.
64-bit source MAC (remote address)	4 (MSB) 5 6 7 8 9 10 11 (LSB)	0x00 0x13 0xA2 0x00 0x40 0x3A 0xB3 0x10	Identifica la dirección MAC del nodo/dispositivo de origen de la trama. Se admite la dirección "00 00 00 00 00 00 FF FF" para indicar la dirección de *broadcast*.
Reserved	12 13	**0xFF** **0xFE**	Campo reservado. Siempre 0*xFFFE*.
Receive options	14	0x01	0x01 = Se envío ACK (asentimiento) del paquete recibido. 0x02 = El paquete fue una transmisión *broadcast*. El resto de bits son reservados y deben ser ignorados.
Number of samples	15	**0x01**	Número total de conjuntos de muestras incluidos (siempre 0x01)
Digital channel mask	16 17	0x00 0x18	Máscara de bits - Indica que líneas I/O están activas como entradas digitales en el *XBee*® origen de las muestras (si hay alguna). Por ello, se puede usar para conocer cuántas muestras digitales se incluyen en la trama.
Analog channel mask	18 19	0x00 0x03	Máscara de bits - Indica que líneas I/O están activas como entradas analógicas en el *XBee*® origen de las muestras (si hay alguna). Por ello, se puede usar para conocer cuántas muestras ADC se incluyen en la trama.
Digital Samples (opcional)	20 21	0x00 0x10	Bitfield - Si el campo "Digital channel mask" es 0x0000, entonces este campo no será incluido. Las líneas digitales que no están activas devolverán 0, las que estén activas, 1. Los bits de este campo de 2-bytes tienen la misma correspondencia que la mostrada en el campo "Digital channel mask".
Analog samples #1 (opcional) Analog samples #2 (opcional)	22 23 24 25	0x022F 0x01FE	Bitfield - Si el campo "Analog channel mask" es 0x0000, entonces este campo no será incluido. Cada entrada analógica activa devolverá un valor de 2-bytes indicando la medida A/D en dicha entrada. Las muestras analógicas están ordenadas secuencialmente, desde AD0/DIO0 a AD3/DIO3, y una última entrada mostrando el voltaje de alimentación (Vdd).
Checksum	26	0x21	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser 0*xFF*.

3.11 Trama $0x97$: Remote AT Command Response

Esta trama se recibe siempre desde la UART del módulo conectado en local, y **siempre en respuesta** a una trama del tipo $0x17$ (*Remote AT Command Request*; ver Página 21, Sección 3.4) previamente enviada por RF desde el nodo local a un nodo remoto. Algunos comandos (como el comando "ND" *Node Discovery*) pueden provocar el envío de varias tramas para completar la respuesta.

Ejemplo: la Tabla 3.14 ilustra los campos de la trama tipo $0x97$: *Remote AT Command Response*. Suponiendo que se ha realizado previamente una petición, en una trama tipo $0x17$ con *Frame ID* = $0x01$, de leer el valor del parámetro "NP" (número máximo de bytes que pueden ser transmitidos en el campo *payload* en una transmisión *unicast*) en un dispositivo remoto con dirección MAC "00 13 A2 00 40 3A B3 10", si la petición resulta exitosa, se recibirán los siguientes datos en hexadecimal:
"*7E 00 11 97 01 00 13 A2 00 40 3A B3 10 FF FE 4E 50 00 01 00 D9*".

Tabla 3.14 Frame type = $0x97$. Trama Rx: "Remote AT Command Response".

Frame fields	Offset	Ejemplo	Descripción
Start delimiter	0	**0x7E**	Identifica el inicio de una nueva trama. Siempre $0x7E$.
Length	1 (MSB) 2 (LSB)	0x00 0x11 (17)	Número de bytes entre *length* y *checksum* (ambos excluidos).
Frame type	3	**0x97**	Identifica el tipo de trama.
Frame ID	4	0x01	Identifica la trama de datos UART que se reporta (ACK), conteniendo el mismo valor que se envío en la petición (trama $0x08$). Nota: Si $FrameID = 0x00$ en AT-mode, no se enviará respuesta (ACK).
64-bit source MAC (remote address)	5 (MSB) 6 7 8 9 10 11 12 (LSB)	0x00 0x13 0xA2 0x00 0x40 0x3A 0xB3 0x10	Identifica la dirección MAC del nodo/dispositivo de origen de la trama. Se admite la dirección "00 00 00 00 00 00 FF FF" para indicar la dirección de *broadcast*.
Reserved	13 14	**0xFF** **0xFE**	Campo reservado. Siempre $0xFFFE$.
AT command	15 16	0x4E (N) 0x50 (P)	Nombre del comando (siempre dos carácteres ASCII que identifican el comando AT).
Command status	17	0x00	0x00 = OK. 0x01 = ERROR. 0x02 = Comando inválido. 0x03 = Parámetro inválido.
Command data (opcional)	18 19	0x01 0x00	Valor (en binario, no ASCII) contenido en el registro correspondiente al comando del campo *AT command*. Si este campo no se envía, indica que en la trama tipo $0x17$ se ha configurado algún valor en dicho registro. En este ejemplo, este campo contiene el valor *"01 00"* (en decimal: *256*).
Checksum	20	0xD9	La suma (en 8 bits) de todos los bytes desde el offset 3 al checksum (ambos inclusive) debe ser $0xFF$.

4 Librería "UniversalXBee"

A continuación se muestra la documentación de la librería desarrollada en este proyecto, que se ha denominado *UniversalXBee*. Todo el desarrollo de código, tanto de ficheros fuente (.cpp), ficheros de cabecera (.h), como un archivo *makefile* (Sección 4.1, que contiene diversas funcionalidades incorporadas para reducir el tamaño del código generado por el compilador), se ha realizado usando el *Eclipse IDE para Desarrolladores C/C++. Versión: Mars.1 Release (4.5.1) - Junio 2015* [11]. Todo el código de la librería *UniversalXBee* (makefile, .cpp, .h) ha sido desarrollado bajo la licencia GPLv3 (*GNU General Public License*)[14].

Por otro lado, para documentar el código, se ha utilizado la herramienta *Doxygen* [10], que tal y como se describe en su web, "*Doxygen es* de-facto *la herramienta estándar para generar documentación a partir de código comentado (...)"*.

En el Capítulo 5 se ha incluido una copia integra del documento que *Doxygen* genera a partir de los ficheros de la librería *UniversalXBee* de este proyecto. Al tratarse de un documento susceptible de ser publicado o compartido en Internet u otros medios, no se incluye en dicha documentación el código de los ficheros fuente (.cpp), dónde se definen e implementan las funciones declaradas en los ficheros de cabecera. Se ha incluido un breve resumen de las características básicas para comentar un código usando el formato *Doxygen* en el Apéndice B. Como referencias de aprendizaje o consulta de las extensas características que soporta el lenguaje C++, se han utilizado los siguientes recursos gratuitos y online: [5], [6] y [7].

Cabe destacar algunos aspectos de diseño que se han seguido en el desarrollo de esta librería, las cuales se han tratado seguir en la medida de lo posible en el proceso de programación ciertas directrices y normas de estilo [18] y [8].

- **Evitar las variables globales.** En caso de ser imprescindible su uso, se ha optado por convertirlas en constantes (modificador "**const**") y no definir constantes "mágicas" mediante el uso de "**#Define**".

- **Usar "const" frecuentemente.** Este modificador es "contagioso", es decir, usado en una parte del código, todas los métodos/variables qe quieran hacer uso de él, no podrán modificar su valor y serán de forma inherente también de tipo "const".

- **Simplificar al máximo el uso de la librería.** Se ha organizado el código en tres grandes grupos (*FrameXBee*, *SerialXBee*, *UtilsXBee*), con una única **clase** dentro de cada uno de ellos, denominada con el mismo nombre que el fichero de cabecera que las contiene.

  FrameXBee.h Métodos, clases y variables para manejar tramas *XBee-PRO® 868* y/o configurar las diferentes tramas contempladas en el protocolo de comunicación de los módulos *XBee-PRO® 868*. *XBee®* es "big-endian" en las tramas con datos multi-byte, y esta librería sigue el mismo criterio.

SerialXBee.h Métodos para controlar/configurar conexiones con un módulo *XBee*® a través de una interfaz serie estándar. Esta librería permite abrir/cerrar un puerto serie, configurar el *baudrate*[1] (valores estándar soportados por los módulos, desde 1200 a 230400 bps) de la conexión serie, así como establecer correctamente las opciones por defecto para el puerto: 8-bits, sin paridad, 1-bit de stop.

UtilsXBee.h Utilidades auxiliares para tramas *XBee*® , para realizar tareas concretas de forma especializada y optimizada. Esta librería está pensada para usarse de forma completamente independiente al tipo de trama *XBee*® utilizada.

- Aunque para la documentación y comentarios del código se ha utilizado el español al tratarse de un proyecto desarrollado en dicho idioma, todo el código se ha programado en inglés, favoreciendo así su internacionalización y/o un posible desarrollo y/o mejora en el futuro por cualquier programador del mundo. Además, dado que toda la documentación de la página oficial de los módulos *XBee*® está redactada en inglés, se logra de este modo un seguimiento más sencillo de la utilidad de cada función/clase/estructura del código.

- **Ámbito y duración de las variables.** Se ha seguido la pauta de definir variables y buffers auxiliares localmente (dentro de las propias funciones) siempre que ha sido posible. Esto evita el uso inadecuado de memoria, ya que el espacio que ocupan queda liberado al salir de la función.

- **Paso por referencia de grandes estructuras de datos.** Siguiendo con el anterior punto, las funciones reciben aquellas estructuras de datos (structs, arrays, buffers) de grandes dimensiones por referencia (operador &) o por dirección de memoria (operador "puntero" *). Hacerlo por valor, provoca que en el interior de la función se realice una copia integra de la estructura pasada como argumento.

- **Simplificación de valores devueltos.** Salvo en los casos que ha sido estrictamente necesario seguir otro criterio, todas las funciones devuelven ("return") un valor cero en caso de no producirse ningún error en la ejecución, o un valor negativo (-1) en caso contrario.

- **Evitar excepciones.** No se realizan reservas/liberación dinámica de memoria (fuente común de errores), y se controla el posible desbordamiento de buffers calculando previamente su tamaño.

- **Modo depuración o *verbose*.** Se han incluido en diferentes partes del código, diversas cláusulas *#ifdef #endif* que permiten activar/desactivar a voluntad del usuario la impresión por salida estándar (pantalla, std::out, stdout) de mensajes informativos de los procesos/errores que ocurren en la ejecución, ayudando a identificar errores de forma fácil e intuitiva por el usuario sin necesidad de buscar el origen de una ejecución errónea.

- **Ficheros de cabecera "auto-suficientes".** Un fichero de cabecera (.h) bien diseñado debe poder compilarse sin necesidad de incorporar dependencias de otras librerías (a excepción de algunas librerías estándar del lenguaje C++). Se ha decidido incorporar en todos los .h la librería estándar <*cstdint*>, que define los "*typedef*" de tipos de enteros (int8_t, int16_t,... y sus homólogos con signo uint8_t, etc), ya que depende del sistema tener o no implementados dichos tipos enteros.

- **Valores "por defecto" para las opciones más utilizadas.** Una característica del lenguaje C++ usada profusamente en el código es la inclusión en las declaración de las funciones de valores por defecto en ciertos argumentos. Esto simplifica el uso de dichas funciones, evitando en la mayoría de casos reales, tener que introducir una extensa lista de parámetros en las llamadas a las funciones.

[1] Los *baudrates*[bps] admitidos por la librería son: 0, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, y 230400.

4.1 Makefile: ejecutable para pruebas

En paralelo al desarrollo del código de la librería de este proyecto, *UniversalXBee*, se ha ido desarrollando un fichero de pruebas, llamado *test.cpp*, el cual ha sido de gran utilidad para testar y depurar errores en las diferentes configuraciones y opciones posibles que abarcan las funciones de la librería. Como es bien sabido, para poder compilar un ejecutable, es necesario crear un fichero de nombre *makefile* (sin ningún tipo de extensión de archivo), el cual contiene las reglas para crear el ejecutable y guardarlo en el mismo directorio desde el que se ejecuta el *makefile*. El fichero *makefile* hace uso de la versátil utilidad *GNU make* (que forma parte de GNU GCC: Apéndice A) para compilar (*compiler*), enlazar (*linker*) y crear (*builder*) el ejecutable objetivo (Figura 4.1), que en nuestro caso hemos llamado *testxbee*.

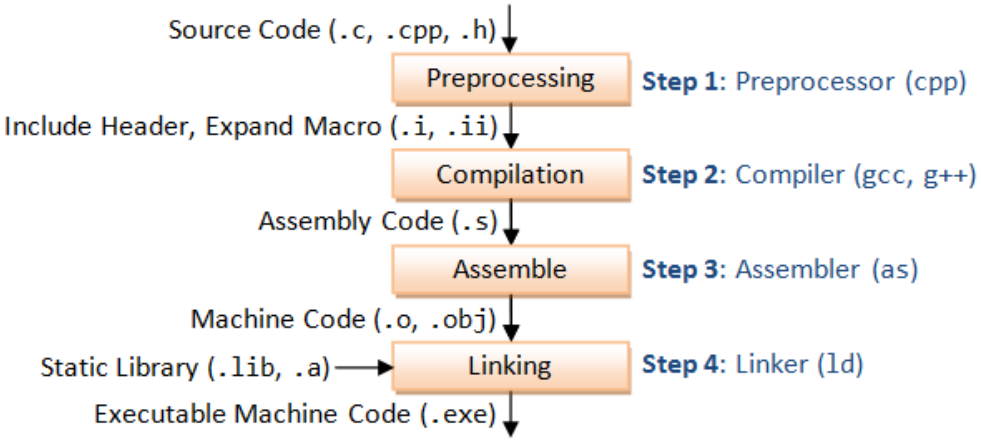


Figura 4.1 Pasos del proceso de compilación seguido por GCC (Imagen extraída de la web [13]).

No es objetivo de este proyecto describir las extensas características de *GNU make*, por lo que se emplaza al lector que desee obtener más información al respecto a consultar la bibliografía correspondiente (referencias [13] y [16]). No obstante, si se va a comentar aquellas opciones que se han incluido en el fichero *makefile* que acompaña este texto, permitiendo así que cualquier usuario conozca el alcance de las acciones que se desarrollan al crear el ejecutable.

Observando el *makefile* (Listing 4.1) pueden observarse rápidamente una estructura básica conocida por cualquier usuario con un mínimo de conocimientos en programación, así como se advierte la presencia de diversas opciones avanzadas que se explicarán más adelante. No obstante, los comentarios contenidos en el propio *makefile* facilitan una ligera comprensión de las diferentes opciones y "macros".

Primeramente, mencionar que en un fichero *makefile*, las líneas que contienen un comentario se indican mediante el carácter "almohadilla" **#** (a diferencia de un código en C/C++ donde una línea de comentario se indica por medio de **//**). Además, se han utilizado denominadores con todas las letras en mayúsculas para diferenciar las "macros" del resto del código y facilitar su identificación y lectura.

A continuación, una breve explicación de las opciones avanzadas utilizadas[2], que implementan varias técnicas simples para minimizar el tamaño del código generado en un programa C++, optimizándolo para su ejecución en entornos embebidos Linux con soporte para el dialecto C++ *-std=gnu++11*:

[2] En las opciones explicadas, el prefijo **-fno** indica que se desactiva dicha opción, y el prefijo **-f** que la opción se activa. Es posible activar/desactivar cualquiera de las opciones simplemente cambiando dicho prefijo.

- **Opciones de optimización:**

  -ffunction-sections Siempre que en un proyecto se usa una librería, el ejecutable final contendrá todas las funciones de esta, incluidas aquellas que no se han usado en el programa. Por ello, al incorporar una extensa librería a un archivo fuente C/C++, y sólo llamar a algunas de sus funciones o métodos de clase, se produce una gran "gasto" de memoria, que si bien en un PC normal no debe suponer ningún inconveniente, en un entorno embebido puede llegar a ser un problema bastante serio.

  La opción **-ffunction-sections** es una bandera de compilación que permite que cada función esté contenida en su propia sección en el ejecutable (el nombre de cada sección vendrá determinado por el nombre de la función), manteniendo a cada función por separado incluso en el caso de que compartan un mismo archivo objeto. Se debe tener en cuenta que esta opción debe usarse siempre junto a la opción *-Wl,–gc-sections*, ya que por si sola creará archivos objeto y ejecutables de gran tamaño. Además, sólo se debe usar esta bandera en aquellos sistemas donde el *linker* soporte esta opción. Se recomienda no usar esta opción conjuntamente con la opción **-g** (depuración).

  Como punto a tener en cuenta, las propias librerías estándar del lenguaje C++ son compiladas por defecto con la opción *-ffunction-sections*, por lo que su uso conlleva una notable reducción del tamaño del código en C++.

  -Os La optimización de tamaño **-Os** activa todas aquellas optimizaciones de -O2 que de forma general no incrementan el tamaño del código, realizando además ciertas acciones que reducen su tamaño.

- **Opciones de "enlazado" (*linker*:)**

  -Wl,–gc-sections La opción **-Wl,–gc-sections** es una bandera del *linker* que aplica una "recogida de basura" (*garbage collection*) sobre todas las secciones creadas por el comando *-ffunction-sections*. Esto significa que el *linker* "lee" línea a línea, y de forma recursiva, todo el código de la función "main()" de nuestro programa, activa una bandera para todas aquellas funciones que son llamadas en algún momento de la ejecución, y aquellas funciones (secciones) que al finalizar no han sido marcadas, son eliminadas del archivo binario con el que se generará el ejecutable final.

- **Excepciones y RTTI:**

  -fno-exceptions -fno-rtti Se haga uso o no de *Run-Time Type Information* (RTTI)[3], el compilador dará soporte para el control del tiempo de ejecución en cualquier caso. Para evitarlo, es posible informar al compilador de que no se está usando *RTTI* y desactivar su suporte con la bandera **-fno-rtti**. En concreto, esta opción desactiva la generación de información sobre cada clase que contenga funciones de tipo *virtual*, las cuales son usadas por C++ en tiempo de ejecución para identificar características de tipos (*dynamic_cast* y *typeid*). Si no se van a usar dichas características de C++ en el código, el uso de esta bandera permite ahorrar cierta cantidad de memoria.

  Por otro lado, el compilador también agrega soporte para manejar las excepciones que se puedan producir en el código, lo que supone generar código extra para permitir la propagación de dichas excepciones. Si en el código programado no se hace uso de excepciones, es posible informar de ello al compilador, desactivando el manejo de excepciones con la bandera **-fno-exceptions**.

  Ha de tenerse en cuenta que ambas opciones, *-fno-exceptions* y *-fno-rtti*, deben pasarse tanto al compilador como al *linker*.

[3] RTTI hace referencia a la habilidad de un sistema de analizar si un objeto es dinámico y ofrecer información en tiempo de ejecución (en contraste con el tiempo de compilación) sobre dicho tipo, lo que permite al programador tener mayor control de los recursos utilizados por un programa.

Listing 4.1 Archivo para generar un ejecutable *testxbee* usando la librería *UniversalXBee*.

```
### Makefile v2.2
### Last edit: 12th-September-2016
### Author: Navea Jimenez, Antonio Rafael (Seville - Spain)
### Project: Standard software interface for communication with XBee RF modules
### Using the library 'UniversalXBee'
### Target executable file: 'testxbee'

### Version history:
##  v2.2  Added to 'linker flags' the "garbage collection" of all sections
##      (-Wl,--gc-sections)
##  v2.1  Disabled exception and RTTI support (-fno-exceptions -fno-rtti)
##      Added this option to compiler and linker flags.
##  v2.0  Makefile will now include advanced tasks to minimize the code size
##
##  v1.4  Added new macros
##  v1.3  Added comments and makefile "header" (author, project, version,...)
##  v1.2  Added 'echo' information strings
##  v1.1  Added some macros
##  v1.0  Simple and functional version of makefile (without macros)

### This file is the Makefile for a test program called 'testxbee' which uses
##  the C++ library created for the above mentioned project.


### Manual Reference (online free resource) for GNU compilers:
##  https://gcc.gnu.org/onlinedocs/gcc/index.html
##
##  Tip: to use variables later in the Makefile, use: $()
##


### Add all the source files here:
SOURCES =  \
test.cpp   \
SerialXBee.cpp \
FrameXBee.cpp \
UtilsXBee.cpp


### Comment in order to add C++ exception and RTTI support:
EXCEPT_FLAGS  = -fno-exceptions -fno-rtti

### Uncomment in order to add debugging information for GDB debugger:
##  DEBUG  = -g

### Uncomment in order to enable some code optimizations:
OPTIMIZE_SIZE = -Os -ffunction-sections

### Dialect (or language standard) used by the compiler:
DIALECT = -std=gnu++11


CPP      = g++
TARGET     = testxbee
COMPILER_FLAGS = -Wall $(OPTIMIZE_SIZE) $(DIALECT) $(DEBUG) $(EXCEPT_FLAGS)
```

```
LINK_FLAGS   = -Wl,--gc-sections $(EXCEPT_FLAGS)


### The following macro replaces the suffix .cpp of all files in the macro
##  SOURCES with the .o suffix and assigns it to the macro OBJS:
##
OBJS = $(SOURCES:.cpp=.o)


### Type the command 'make clean' removes the executable file, as well as all
##  old .o objects files and *~ backup files:
##
clean:
  @echo "Removing the executable file, backup files and object files."
  $(RM) $(TARGET) *.o *~


### Type the command 'make' or 'make all' in order to create the executable:
##
all: $(TARGET)
  @echo Finished building target: $(TARGET)


### Building the executable target called 'testxbee':
##
$(TARGET): $(OBJS)
  @echo "Linking object files..."
  $(CPP) $(LINK_FLAGS) $(OBJS) -o $(TARGET)


### Building the object file test.o:
##
test.o: test.cpp SerialXBee.h FrameXBee.h UtilsXBee.h
  $(CPP) $(COMPILER_FLAGS) -c test.cpp


### Building the object file SerialXBee.o:
##
SerialXBee.o: SerialXBee.cpp SerialXBee.h
  $(CPP) $(COMPILER_FLAGS) -c SerialXBee.cpp


### Building the object file FrameXBee.o:
##
FrameXBee.o: FrameXBee.cpp FrameXBee.h UtilsXBee.h SerialXBee.h
  $(CPP) $(COMPILER_FLAGS) -c FrameXBee.cpp


### Building the object file UtilsXBee.o:
##
UtilsXBee.o: UtilsXBee.cpp UtilsXBee.h
  $(CPP) $(COMPILER_FLAGS) -c UtilsXBee.cpp
```

Librería UniversalXBee v1.1

Desarrollador: Antonio Navea Jiménez

Documento Generado por Doxygen 1.8.11
Septiembre 2016

Índice general

Capítulo 1

Librería UniversalXBee (C/C++)

Los módulos XBee RF embebidos (de Digi International) son utilizados en numerosas plataformas de sensores inalámbricos para realizar intercambios de datos mediante el uso de radiofrecuencia. El acceso a estos módulos puede realizarse a través de un puerto serie estándar, y por tanto, su uso no tiene porqué estar estrictamente restringido a dichas redes de sensores. En el presente proyecto se ha desarrollado una librería en código C/↩ C++ que permite el uso de módulos XBee-PRO 868 MHz a través de interfaces de comunicación serie, facilitando la creación de programas que hacen uso de las características de los XBee para transmitir o recibir datos sobre plataformas muy diversas (actualmente con soporte para sistemas operativos POSIX: Windows/Cygwin y Linux con GCC), tales como PC's o dispositivos Raspberry Pi.

-> Esta librería permite utilizar los módulos XBee en modo API, sin caracteres de escape (AP = 1), y con caracteres de escape (AP = 2).

-> Esta librería utiliza el dialecto -std:GNU++11.

-> Esta librería no proporciona funciones de encriptación o cifrado de los datos enviados.

-> Esta librería se acompaña con un fichero "Makefile" especialmente diseñado para crear programas con diversas optimizaciones para minimizar el tamaño del código del ejecutable generado.

-> Esta librería permite abrir/cerrar un puerto serie, configurar el baudrate (valores estándar soportados por los módulos, desde 1200 a 230400 bps) de la conexión serie, así como establecer correctamente las opciones por defecto para el puerto: 8-bits, sin paridad, 1-bit de stop.

-> UniversalXBee está organizada en tres partes o sub-librerías:

1.- FrameXBee: módulos, clases y variables para manejar tramas XBee-Pro 868.

2.- SerialXBee: métodos para controlar/configurar conexiones con un módulo XBee a través de una interfaz serie estándar.

3.- UtilsXBee: utilidades auxiliares para tramas XBee, para realizar tareas concretas de forma especializada y optimizada.

Capítulo 2

Índice de clases

2.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Capítulo 3

Indice de archivos

3.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

Capítulo 4

Documentación de las clases

4.1. Referencia de la Clase FrameXBee

Clase: Métodos y variables para manejar y/o configurar las diferentes tramas contempladas en el protocolo de comunicación de los módulos XBee-Pro 868MHz.

```
#include <FrameXBee.h>
```

Métodos públicos

- int8_t addDataToPayload (packetXBee &packet, const uint8_t *data, uint8_t dataLength)

  Agrega datos contenidos en "data" al campo "payload" de la trama "packet".
- int8_t addDataToPayload (packetXBee &packet, const char *string)
- int8_t addDataToPayload (packetXBee &packet, uint8_t data)
- int8_t addDataToPayload (packetXBee &packet, uint16_t data)
- int8_t addDataToPayload (packetXBee &packet, uint32_t data)
- int8_t addDataToPayload (packetXBee &packet, float data)
- int8_t checkFrameType (uint8_t frameType)

  (inline) Comprueba si el byte pasado como argumento de la función es un tipo de trama válido o reconocido.
- FrameXBee ()

  Constructor por defecto de la clase FrameXBee.
- void getCommandFields (packetXBee &packet, uint8_t *bufferTX)

  Extrae los campos específicos de las tramas API para enviar comandos AT (tipos COMMAND_TX y LOCAL_CO↩ MMAND_TX) y los asigna en el lugar adecuado dentro del buffer de transmisión que se pasa por parámetro.
- void getDestinationAddress (packetXBee &packet, uint8_t *bufferTX)

  Extrae la dirección MAC (64-bits) del nodo destino de la trama.
- uint8_t getModeAPI ()

  (inline) Extrae el modo API usado en la lectura/transmisión de tramas.
- void getTxFields (packetXBee &packet, uint8_t *bufferTX)

  Extrae los campos específicos de las tramas API para enviar datos (tipos NORMAL_TX y EXPLICIT_TX) y los asigna en el lugar adecuado dentro del buffer de transmisión que se pasa por parámetro.
- int8_t readXBee (int fileD, uint8_t *buffer, uint32_t interval=10000)

  Realiza la lectura, dentro de un intervalo de tiempo (en milisegundos), de los bytes presentes en el buffer RX del módulo XBee.
- int8_t saveDataToFile (const char *fileName, const uint8_t *buffer, uint16_t length, uint16_t offset=0)

  Guarda en un fichero en disco los datos binarios que contiene 'buffer'.

- int8_t sendFrameNormal (packetXBee &packet, int fileD, const char *address, const uint8_t *data, uint16_t dataLength, uint8_t frameID=0x01)

  Envía una trama tipo 0x10 (NORMAL_TX) con las opciones por defecto, comprobando que la transmisión se realiza de forma correcta.
- int8_t sendFrameNormal (packetXBee &packet, int fileD, const char *address, const char *data, uint8_t frameID)
- int8_t sendXBee (packetXBee &packet, int fileD, uint8_t frameID=0x01)

  Extrae de la estructura 'packetXBee' los datos de la trama tipo 'frameID' y los transmite a través del puerto serie (descriptor de fichero 'fileD').
- int8_t setCommand (packetXBee &packet, uint8_t frameType, const char *commandAT, uint8_t options=0x02)

  Asigna, en los campos correspondientes de la estructura 'packetXBee', los campos específicos de las tramas API para enviar comandos AT sin parámetro (en las tramas tipo COMMAND_TX y tipo LOCAL_COMMAND_TX).
- int8_t setCommandParameter (packetXBee &packet, uint8_t frameType, const char *commandAT, const char *parameter, uint8_t options=0x02)

  Asigna, en los campos correspondientes de la estructura 'packetXBee', el comando AT y su parámetro, y calcula su longitud.
- int8_t setCommandParameter (packetXBee &packet, uint8_t frameType, const char *commandAT, uint8_t parameter, uint8_t options=0x02)
- int8_t setCommandParameter (packetXBee &packet, uint8_t frameType, const char *commandAT, uint16_t parameter, uint8_t options=0x02)
- int8_t setCommandParameter (packetXBee &packet, uint8_t frameType, const char *commandAT, uint32_t parameter, uint8_t options=0x02)
- int8_t setDestinationAddress (packetXBee &packet, const char *address)

  Asigna la dirección MAC "address" al campo "destinationMAC[]" de "packet".
- void setModeAPI (uint8_t mode=API_WITH_ESCAPES)

  (inline) Configura el modo API a usar en la lectura/transmisión de tramas.
- void setRetriesTx (uint8_t retries)

  (inline) Configura el máximo número de intentos que se realizarán para tratar de transmitir correctamente una trama con la función 'sendXBee()'.
- int8_t setTxFields (packetXBee &packet, uint8_t frameType, uint8_t radius=0x00, uint8_t txOptions=0x00, uint8_t sourceEP=0xE8, uint8_t destinationEP=0xE8, uint16_t CID=0x11, uint16_t PID=0xC105)

  Permite configurar los parámetros propios de las tramas XBee Tx de tipo NORMAL_TX (Transmit Request) y tipo EXPLICIT_TX (Explicit Addressing Command Frame).

Atributos privados

- uint8_t _modeAPI
- uint8_t _sendRetries

4.1.1. Descripción detallada

Clase: Métodos y variables para manejar y/o configurar las diferentes tramas contempladas en el protocolo de comunicación de los módulos XBee-Pro 868MHz.

Definición en la línea 468 del archivo FrameXBee.h.

4.1.2. Documentación del constructor y destructor

4.1.2.1. FrameXBee::FrameXBee ( )

Constructor por defecto de la clase FrameXBee.

Configura como valor por defecto el modo API con caracteres de escape (AP = 2) mediante "setModeAPI(API_↩ WITH_ESCAPES)", y estable el número máximo de reintentos que se realizarán para tratar de envíar una trama mediante "setRetriesTx(TX_RETRIES)".

Definición en la línea 49 del archivo FrameXBee.cpp.

4.1.3. Documentación de las funciones miembro

4.1.3.1. int8_t FrameXBee::addDataToPayload (packetXBee & *packet,* const uint8_t * *data,* uint8_t *dataLength*)

Agrega datos contenidos en "data" al campo "payload" de la trama "packet".

Es posible utilizar varias veces esta función para concatenar diferentes sets de datos dentro de la misma trama "packet": la función comprueba si existen datos previos en el campo "payload" de la trama, y en caso afirmativo, extrae la longitud (en bytes) de los mismos, asegurando que en ningún caso la suma de todos los datos concatenados supere MAX_PAYLOAD_LENGTH bytes.

En el caso de que "dataLength" supere el espacio (en bytes) que queda libre en "payload", la función concatenará datos, desde "data" a "payload", hasta alcanzar MAX_PAYLOAD_LENGTH, descartando los bytes restantes.

Nota

En todas las versiones sobrecargadas de esta función, se calcula internamente el valor del parámetro "data↩ Length".

Atención

Es responsabilidad del usuario asegurar que el tamaño indicado por MAX_PAYLOAD_LENGTH es el mismo en ambos nodos (origen/destino) del link RF de la comunicación, y que en cualquier caso, MAX_PAYLOA↩ D_LENGTH es inferior al tamaño especificado por MAX_FRAME_LENGTH e inferior al tamaño máximo del buffer Rx/Tx del módulo XBee.

Parámetros

out	*packet*	Referencia a la estructura "packetXBee" que almacena los diferentes campos de la trama.
in	*data*	Puntero al array que contiene los datos (carga útil) que se enviarán por RF al nodo destino. Pueden ser datos binarios o ASCII.
in	*dataLength*	Longitud/tamaño (en bytes) de la carga útil "data".

Valores devueltos

0	Éxito: todos los bytes de "data" han sido correctamente concatenados en el campo "payload" de "packet".
-1	Error: "dataLength" supera el espacio libre remanente en el campo"payload" de "packet".

Ver también

packetXBee, MAX_PAYLOAD_LENGTH, MAX_FRAME_LENGTH

Definición en la línea 64 del archivo FrameXBee.cpp.

4.1.3.2. int8_t FrameXBee::addDataToPayload (packetXBee & *packet,* const char * *string*)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

addDataToPayload()

Definición en la línea 124 del archivo FrameXBee.cpp.

4.1.3.3. int8_t FrameXBee::addDataToPayload (packetXBee & *packet,* uint8_t *data*)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

addDataToPayload()

Definición en la línea 136 del archivo FrameXBee.cpp.

4.1.3.4. int8_t FrameXBee::addDataToPayload (packetXBee & *packet,* uint16_t *data*)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

addDataToPayload()

Definición en la línea 151 del archivo FrameXBee.cpp.

4.1.3.5. int8_t FrameXBee::addDataToPayload (packetXBee & *packet,* uint32_t *data*)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

addDataToPayload()

Definición en la línea 166 del archivo FrameXBee.cpp.

4.1.3.6. int8_t FrameXBee::addDataToPayload (packetXBee & *packet,* float *data*)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Nota

En este caso, al ser el parámetro "data" de tipo float, la función comprueba internamente si el sistema sobre el que se ejecuta el código es "Little endian" o "Big endian" antes de agregar los datos a "payload".

Ver también

addDataToPayload(), UtilsXBee::systemLittleEndian()

Definición en la línea 182 del archivo FrameXBee.cpp.

4.1.3.7. int8_t FrameXBee::checkFrameType (uint8_t *frameType*) [inline]

(inline) Comprueba si el byte pasado como argumento de la función es un tipo de trama válido o reconocido.

Se proporciona esta función como tipo "inline" para permitir que el usuario pueda añadir o borrar a voluntad, y según sus necesidades, los tipos de tramas aceptadas y reconocidas como válidas. La función "FrameXBee↩ ::readXBee()" hace uso de esta característica para comprobar que los bytes que se están leyendo son de utilidad para el proposito del software desarrollado por el usuario con esta librería.

Parámetros

in	*frameType*	Byte cuyo valor (hexadecimal) se quiere comprobar si corresponde con algún tipo de trama reconocida o aceptada como válida.

Valores devueltos

frameType	Devuelve el byte pasado como parámetro de la función, lo que significa que ha sido reconocido/aceptado como válido.
-1	Error: el byte "frameType" no se acepta o reconoce.

Ver también

readXBee()

Definición en la línea 1161 del archivo FrameXBee.h.

4.1.3.8. void FrameXBee::getCommandFields (packetXBee & *packet,* uint8_t * *bufferTX*)

Extrae los campos específicos de las tramas API para enviar comandos AT (tipos COMMAND_TX y LOCAL_CO↩ MMAND_TX) y los asigna en el lugar adecuado dentro del buffer de transmisión que se pasa por parámetro.

Los datos de los campos se extraen de la estructura 'packet', y deben haber sido configurados previamente con las funciones setCommandParameter() y setCommand().

Finalmente, calcula el checksum de todos los datos contenidos en bufferTX y se asigna dicho valor en el byte correspondiente.

Parámetros

in,out	*packet*	Referencia a estructura "packetXBee" que almacena los diferentes campos de la trama.
out	*bufferTX*	Puntero al buffer de transmisión, que almacena de forma temporal todos los bytes de la trama XBee que se enviará al nodo destino.

Devuelve

void

Ver también

setCommand(), setCommandParameter(), getChecksum(), packetXBee

Definición en la línea 212 del archivo FrameXBee.cpp.

4.1.3.9. void FrameXBee::getDestinationAddress (packetXBee & *packet,* uint8_t * *bufferTX*)

Extrae la dirección MAC (64-bits) del nodo destino de la trama.

La dirección MAC debe asignarse previamente en el campo "destinationMAC[ ]" de la trama "packet" haciendo uso de la función 'getDestinationAddress()'. Se copian dichos bits en el lugar correspondiente dentro de "bufferTX".

Además, esta función rellena en "bufferTX" los bytes correspondientes a un campo reservado dentro de la trama que no puede ser modificado por el usuario, de valor fijo "0xFFFE".

Parámetros

in	*packet*	Referencia a estructura "packetXBee" que almacena los diferentes campos de la trama.
out	*bufferTX*	Puntero al buffer de transmisión, que almacena de forma temporal todos los bytes de la trama XBee que se enviará al nodo destino.

Devuelve

void

Ver también

setDestinationAddress(), sendXBee(), packetXBee

Definición en la línea 270 del archivo FrameXBee.cpp.

4.1.3.10. uint8_t FrameXBee::getModeAPI ( ) [inline]

(inline) Extrae el modo API usado en la lectura/transmisión de tramas.

Valores devueltos

0x01	Tramas API sin caracteres de escape (API_WITHOUT_ESCAPES).
0x02	Tramas API con caracteres de escape (API_WITH_ESCAPES).

Ver también

setModeAPI(), API_WITHOUT_ESCAPES, API_WITH_ESCAPES, UtilsXBee::convertFromScapedBytes(), UtilsXBee::convertToScapedBytes()

Definición en la línea 1194 del archivo FrameXBee.h.

4.1.3.11. void FrameXBee::getTxFields (packetXBee & *packet*, uint8_t * *bufferTX*)

Extrae los campos específicos de las tramas API para enviar datos (tipos NORMAL_TX y EXPLICIT_TX) y los asigna en el lugar adecuado dentro del buffer de transmisión que se pasa por parámetro.

Los datos de los campos se extraen de la estructura 'packet', y deben haber sido configurados previamente con la función 'setTxFields()'.

Finalmente, calcula el checksum de todos los datos contenidos en bufferTX y se asigna dicho valor en el byte correspondiente.

Parámetros

`in,out`	*packet*	Referencia a estructura "packetXBee" que almacena los diferentes campos de la trama.
`out`	*bufferTX*	Puntero al buffer de transmisión, que almacena de forma temporal todos los bytes de la trama XBee que se enviará al nodo destino.

Devuelve

void

Ver también

setTxFields(), getChecksum(), packetXBee

Definición en la línea 297 del archivo FrameXBee.cpp.

4.1.3.12. int8_t FrameXBee::readXBee (int *fileD*, uint8_t * *buffer*, uint32_t *interval* = `10000`)

Realiza la lectura, dentro de un intervalo de tiempo (en milisegundos), de los bytes presentes en el buffer RX del módulo XBee.

Se descarta: -> Cualquier byte leído antes de detectar un byte "Inicio de Trama" (0x7E). -> Tramas corruptas o con errores (checksum o longitud erróneas, etc).

Se leen datos (bytes) del buffer RX del módulo XBee hasta que se cumple alguna de las siguientes condiciones: -> El buffer de lectura alcanza 'MAX_FRAME_LENGTH' bytes leídos -> La longitud de trama (Length) supera el espacio libre en el buffer -> El timer desborda el tiempo 'interval' -> Se termina de leer completamente una trama correcta

Parámetros

in	*fileD*	Descriptor de fichero: puerto serie del que leer datos.
out	*buffer*	Puntero al buffer de recepción, que almacena de forma temporal todos los bytes de la trama XBee que se ha recibido.
in	*interval*	Intervalo de tiempo 'timeout' (en milisegundos) en el que se espera recibir una trama correcta.

Valores devueltos

frameType	Éxito: trama recibida correctamente. Devuelve el byte que indica el tipo de trama leída.
-1	Error: Tipo de Trama no reconocida/aceptada.
-2	Error: Longitud de la trama desborda el buffer de lectura.
-3	Error: Checksum incorrecto.

Definición en la línea 360 del archivo FrameXBee.cpp.

4.1.3.13. int8_t FrameXBee::saveDataToFile (const char * *fileName,* const uint8_t * *buffer,* uint16_t *length,* uint16_t *offset* = 0)

Guarda en un fichero en disco los datos binarios que contiene 'buffer'.

Parámetros

in	*fileName*	Cadena de caracteres con el nombre del fichero que se desea abrir para almacenar los datos.
in	*buffer*	Puntero al buffer de datos binarios que se desean almacenar en un fichero.
in	*length*	Longitud de los datos a copiar desde 'buffer' al fichero 'fileName'
in	*offset*	Offset de datos en 'buffer'. Se empezará a copiar bytes desde la posición buffer[offset] al fichero 'fileName'.

Valores devueltos

0	Éxito: Datos escritos correctamente en el fichero 'fileName'.
-1	Error: Error en la apertura del fichero, o durante la escritura de datos.

Definición en la línea 628 del archivo FrameXBee.cpp.

4.1.3.14. int8_t FrameXBee::sendFrameNormal (packetXBee & *packet,* int *fileD,* const char * *address,* const uint8_t * *data,* uint16_t *dataLength,* uint8_t *frameID* = 0x01)

Envía una trama tipo 0x10 (NORMAL_TX) con las opciones por defecto, comprobando que la transmisión se realiza de forma correcta.

Esta función realiza el envío haciendo internamente uso de las funciones: setTxFields(), setDestinationAddress(), addDataToPayload() y sendXBee().

Al finalizar la transmisión, se hace un borrado completo a cero de la estructura "packet".

Parámetros

in	*packet*	Referencia a la estructura "packetXBee" que almacena los diferentes campos de la trama.
in	*fileD*	Descriptor de fichero: puerto serie en el que escribir los datos que se transmitirán.
in	*address*	Dirección MAC unicast (64-bits) del nodo de destino. Admite la dirección de broadcast "FFFF" (16-bits). Es un puntero a array o a una cadena de caracteres.
in	*data*	Puntero al array que contiene los datos (carga útil) que se enviarán por RF al nodo destino. Pueden ser datos binarios o ASCII.
in	*dataLength*	Longitud/tamaño (en bytes) de la carga útil "data".
in	*frameID*	Identificador de Trama: permite identificar una trama enviada con su correspondiente ACK recibido. Si frameID = 0x00, no se recibirá ACK.

Valores devueltos

0	Éxito: Datos de la trama asignados correctamente.
-1	Error: Tipo de Trama no reconocida/aceptada.

Ver también

setTxFields(), setDestinationAddress(), addDataToPayload(), sendXBee()

Definición en la línea 673 del archivo FrameXBee.cpp.

4.1.3.15. FrameXBee::sendFrameNormal (packetXBee & *packet*, int *fileD*, const char * *address*, const char * *data*, uint8_t *frameID*)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

sendFrameNormal()

Definición en la línea 742 del archivo FrameXBee.cpp.

4.1.3.16. int8_t FrameXBee::sendXBee (packetXBee & *packet*, int *fileD*, uint8_t *frameID* = `0x01`)

Extrae de la estructura 'packetXBee' los datos de la trama tipo 'frameID' y los transmite a través del puerto serie (descriptor de fichero 'fileD').

Parámetros

in	*packet*	Referencia a la estructura "packetXBee" que almacena los diferentes campos de la trama.
in	*fileD*	Descriptor de fichero: puerto serie en el que escribir los datos que se transmitirán.
in	*frameID*	Identificador de Trama: permite identificar una trama enviada con su correspondiente ACK recibido. Si frameID = 0x00, no se recibirá ACK.

Valores devueltos

0	Éxito: Datos de la trama asignados correctamente.
-1	Error: Tipo de Trama no reconocida/aceptada.
-2	Error: Longitud de la trama desborda el buffer de lectura.
-3	Error: Checksum incorrecto.

Ver también

sendFrameNormal()

Definición en la línea 761 del archivo FrameXBee.cpp.

4.1.3.17. int8_t FrameXBee::setCommand (packetXBee & *packet*, uint8_t *frameType*, const char * *commandAT*, uint8_t *options* = 0x02)

Asigna, en los campos correspondientes de la estructura 'packetXBee', los campos específicos de las tramas API para enviar comandos AT sin parámetro (en las tramas tipo COMMAND_TX y tipo LOCAL_COMMAND_TX).

Parámetros

out	*packet*	Referencia a estructura "packetXBee" que almacena los diferentes campos de la trama.
in	*frameType*	Campo "frameType": tipo de trama que se está configurando. En esta función, sólo se aceptan los valores COMMAND_TX y LOCAL_COMMAND_TX.
in	*commandAT*	Nombre del comando (siempre dos caracteres ASCII que identifican el comando AT).
in	*options*	Corresponde al campo "Remote command options". Bitfield (valor por defecto = 0x02): bit0 = 1 (0x01): ACK desactivado. bit1 = 1 (0x02): Aplicar cambios inmediatamente en el dispositivo remoto (si no se indica este valor de 0x02, se deberá enviar posteriormente en otra trama el comando "AC" para hacer efectivas las nuevas configuraciones y valores). Los demás bits deben ser 0.

Valores devueltos

0	Éxito: datos de la trama asignados correctamente.
-1	Error: Tipo de Trama no reconocida/aceptada.

Ver también

setCommandParameter(), packetXBee

Definición en la línea 862 del archivo FrameXBee.cpp.

4.1.3.18. int8_t FrameXBee::setCommandParameter (packetXBee & *packet*, uint8_t *frameType*, const char * *commandAT*, const char * *parameter*, uint8_t *options* = 0x02)

Asigna, en los campos correspondientes de la estructura 'packetXBee', el comando AT y su parámetro, y calcula su longitud.

Parámetros

out	*packet*	Referencia a estructura "packetXBee" que almacena los diferentes campos de la trama.
in	*frameType*	Campo "frameType": tipo de trama que se está configurando. En esta función, sólo se aceptan los valores COMMAND_TX y LOCAL_COMMAND_TX.
in	*commandAT*	Nombre del comando (siempre dos caracteres ASCII que identifican el comando AT).
in	*parameter*	Si está presente (esto incluye el valor 0x00), este campo indica el valor que debe asignarse en el registro correspondiente al "commandAT".

Si no se envía el parámetro, indica que se está consultado el valor de dicho registro o parámetro, en cuyo caso se recomienda utilizar la función "setCommand()" en lugar de "setCommandParameter()".

Parámetros

in	*options*	Corresponde al campo "Remote command options". Bitfield (valor por defecto = 0x02): bit0 = 1 (0x01): ACK desactivado. bit1 = 1 (0x02): Aplicar cambios inmediatamente en el dispositivo remoto (si no se indica este valor de 0x02, se deberá enviar posteriormente en otra trama el comando "AC" para hacer efectivas las nuevas configuraciones y valores). Los demás bits deben ser 0.

Valores devueltos

0	Éxito: datos de la trama asignados correctamente.
-1	Error: Tipo de Trama no reconocida/aceptada.

Ver también

setCommand(), packetXBee

Definición en la línea 907 del archivo FrameXBee.cpp.

4.1.3.19. int8_t FrameXBee::setCommandParameter (packetXBee & *packet*, uint8_t *frameType*, const char * *commandAT*, uint8_t *parameter*, uint8_t *options* = 0x02)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

setCommandParameter(), setCommand()

Definición en la línea 924 del archivo FrameXBee.cpp.

4.1.3.20. int8_t FrameXBee::setCommandParameter (packetXBee & *packet*, uint8_t *frameType*, const char * *commandAT*, uint16_t *parameter*, uint8_t *options* = 0x02)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

setCommandParameter(), setCommand()

Definición en la línea 942 del archivo FrameXBee.cpp.

4.1.3.21. int8_t FrameXBee::setCommandParameter (packetXBee & *packet*, uint8_t *frameType*, const char * *commandAT*, uint32_t *parameter*, uint8_t *options* = 0x02)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

setCommandParameter(), setCommand()

Definición en la línea 960 del archivo FrameXBee.cpp.

4.1.3.22. int8_t FrameXBee::setDestinationAddress (packetXBee & *packet*, const char * *address*)

Asigna la dirección MAC "address" al campo "destinationMAC[ ]" de "packet".

La dirección debe proporcionarse en caracteres hexadecimales, sin espacios en blanco entre ellos.

Se admiten dos tipos de dirección: dirección unicast completa (64-bits), o dirección broadcast reducida "FFFF" (16-bits). Si se usa "FFFF", esta función rellenará los bits restantes con valores nulos (0x00) hasta alcanzar los 64-bits.

Ejemplo de uso de esta función:

```
packetXBee frame01;
char MACnodo[] = "0013A200403AB310";

frame.setDestinationAddress( frame01, MACnodo,
      NORMAL_TX );
```

Parámetros

out	*packet*	Referencia a la estructura "packetXBee" que almacena los diferentes campos de la trama.
in	*address*	Dirección MAC unicast (64-bits) del nodo de destino. Admite la dirección de broadcast "FFFF" (16-bits). Es un puntero a array o a una cadena de caracteres.

Valores devueltos

0	Éxito: dirección MAC asignada correctamente.
-1	Error: formato de la dirección MAC incorrecto.

Ver también

packetXBee, getDestinationAddress()

Definición en la línea 986 del archivo FrameXBee.cpp.

4.1.3.23. void FrameXBee::setModeAPI (uint8_t *mode* = API_WITH_ESCAPES) [inline]

(inline) Configura el modo API a usar en la lectura/transmisión de tramas.

Parámetros

in	*mode*	Modo API que se activa: Tramas API sin caracteres de escape (API_WITHOUT_ESCAPES). Tramas API con caracteres de escape (API_WITH_ESCAPES). (valor por defecto = API_WITH_ESCAPES):

Devuelve

void

Ver también

getModeAPI(), API_WITHOUT_ESCAPES, API_WITH_ESCAPES, UtilsXBee::convertFromScapedBytes(), UtilsXBee::convertToScapedBytes()

Definición en la línea 1206 del archivo FrameXBee.h.

4.1.3.24. void FrameXBee::setRetriesTx (uint8_t *retries*) [inline]

(inline) Configura el máximo número de intentos que se realizarán para tratar de transmitir correctamente una trama con la función 'sendXBee()'.

Parámetros

in	*retries*	Número máximo de intentos para una transmisión.

Devuelve

void

Ver también

sendXBee()

Definición en la línea 1219 del archivo FrameXBee.h.

4.1.3.25. int8_t FrameXBee::setTxFields (packetXBee & *packet*, uint8_t *frameType*, uint8_t *radius* = 0x00, uint8_t *txOptions* = 0x00, uint8_t *sourceEP* = 0xE8, uint8_t *destinationEP* = 0xE8, uint16_t *CID* = 0x11, uint16_t *PID* = 0xC105)

Permite configurar los parámetros propios de las tramas XBee Tx de tipo NORMAL_TX (Transmit Request) y tipo EXPLICIT_TX (Explicit Addressing Command Frame).

Todos los parámetros (excepto "frameType") son opcionales: en aquellos que no sean configurados por el usuario, se asignarán valores por defecto de forma automática.

Los parámetros "frameType", "radius", y "txOptions" son comunes a ambos tipos de tramas,siendo el resto parámetros propios de la trama EXPLICIT_TX. El parámetro "frameType" se evalua primero, por tanto, si se intenta configurar en una trama NORMAL_TX los valores de alguno de los parámetros propios de EXPLICIT_TX, la función no devolverá error y se ejecutará correctamente, pero dichos valores no serán tenidos en cuenta.

En la descripción de los parámetros, se indica el nombre del campo dentro de la estructura "packetXBee" al que hacen referencia.

Nota

En las tramas EXPLICIT_TX:

- Los valores por defecto que se asignan a los distintos parámetros activan el modo transparente de transmisión de datos ("transparent data mode"), con clusterID = 0x11, endpoints origen y destino = 0xE8, y profileID = 0xC105.
- Si se desea activar el modo "link-testing", se debe configurar clusterID en modo "loop-back" (0x12), dejando el resto de parámetros con sus valores por defecto.

Parámetros

out	*packet*	Estructura "packetXBee" que almacena los diferentes campos de la trama.
in	*frameType*	Campo "frameType": tipo de trama que se está configurando. En esta función, sólo se aceptan los valores NORMAL_TX y EXPLICIT_TX.
in	*radius*	Campo "hopsRadius": indica el número máximo de saltos permitidos en transmisiones tipo broadcast.
in	*txOptions*	Campo "txOptions": opciones de transmisión.
in	*sourceEP*	Campo "sourceEndpoint": nodo endpoint origen de la transmisión.
in	*destinationEP*	Campo "destinationEndpoint": nodo endpoint destino de la transmisión.
in	*CID*	Campo "CID[]": Cluster ID usado en la transmisión.
in	*PID*	Campo "PID[]": Profile ID usado en la transmisión.

Valores devueltos

0	Éxito: parámetros asignados correctamente.
-1	Error: esta función no admite el tipo de trama especificado en "frameType".

Ver también

packetXBee, sendXBee()

Definición en la línea 1052 del archivo FrameXBee.cpp.

4.1.4. Documentación de los datos miembro

4.1.4.1. uint8_t FrameXBee::_modeAPI [private]

Variable privada: Especifica el modo de funcionamiento (parámentro AP) AP = 0: modo Transparente (API off) AP = 1: modo API, sin caracteres de escape (API-mode 1) AP = 2: modo API, con caracteres de escape (API-mode 2)

Definición en la línea 477 del archivo FrameXBee.h.

4.1.4.2. uint8_t FrameXBee::_sendRetries [private]

Variable privada: Especifica el máximo número de intentos que se realizarán para tratar de transmitir correctamente una trama.

Definición en la línea 483 del archivo FrameXBee.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- FrameXBee.h
- FrameXBee.cpp

4.2. Referencia de la Clase SerialXBee

Clase: Métodos para controlar/configurar conexiones con un módulo XBee a través de una interfaz serie estándar.

`#include <SerialXBee.h>`

Métodos públicos

- int8_t closePort (serial_port &serial)

  Cierra/desconecta la conexión del módulo XBee con el puerto/interfaz serie.
- int8_t openPort (serial_port &serial, uint32_t baudrate, const char *device)

  Establece las nuevas opciones de configuración de la interfaz serie y abre una conexión con el puerto "device" a la velocidad "baudrate".
- SerialXBee ()

  Constructor por defecto de la clase SerialXBee.
- int8_t setBaudrate (serial_port &serial, uint32_t baudrate)

  Configura las velocidades (baudrate) de entrada/salida del puerto, y las opciones por defecto de la interfaz serie.
- int8_t setSerialOptions (serial_port &serial, termios &options)

  Establece las nuevas opciones de configuración de la interfaz serie.

4.2.1. Descripción detallada

Clase: Métodos para controlar/configurar conexiones con un módulo XBee a través de una interfaz serie estándar.

Definición en la línea 78 del archivo SerialXBee.h.

4.2.2. Documentación del constructor y destructor

4.2.2.1. SerialXBee::SerialXBee ( )

Constructor por defecto de la clase SerialXBee.

Definición en la línea 45 del archivo SerialXBee.cpp.

4.2.3. Documentación de las funciones miembro

4.2.3.1. int8_t SerialXBee::closePort (serial_port & *serial*)

Cierra/desconecta la conexión del módulo XBee con el puerto/interfaz serie.

Parámetros

in	*serial*	Referencia a estructura "serial_port" que almacena los datos mínimos necesarios para abrir el puerto serie.

Valores devueltos

0	Éxito: Cierre correcto del puerto serie.
-1	Error: El puerto serie no se ha podido cerrar correctamente.

Ver también

setCommand(), setCommandParameter(), getChecksum(), packetXBee

Definición en la línea 57 del archivo SerialXBee.cpp.

4.2.3.2. int8_t SerialXBee::openPort (serial_port & *serial*, uint32_t *baudrate*, const char * *device*)

Establece las nuevas opciones de configuración de la interfaz serie y abre una conexión con el puerto "device" a la velocidad "baudrate".

Los baudrates admitidos son: 0, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, y 230400.

Parámetros

in	*serial*	Referencia a estructura "serial_port" que almacena los datos mínimos necesarios para abrir el puerto serie.
in	*baudrate*	Velocidad (data rate) en bits-por-segundo a la que se configura el puerto.
in	*device*	Nombre que recibe la interfaz serie en el sistema. Ej.: "/dev/ttyUSBx", "/dev/ttySx".

Valores devueltos

0	Éxito: Apertura/conexión correcta con el puerto serie.
-1	Error: No se ha podido abrir/configurar el puerto serie. Este error puede ser debido a cualquiera de los pasos internos en la configuración de la interfaz serie (baudrate, struct termios).

Ver también

setCommand(), setCommandParameter(), getChecksum(), packetXBee

Definición en la línea 70 del archivo SerialXBee.cpp.

4.2.3.3. int8_t SerialXBee::setBaudrate (serial_port & *serial*, uint32_t *baudrate*)

Configura las velocidades (baudrate) de entrada/salida del puerto, y las opciones por defecto de la interfaz serie.

Los baudrates admitidos son: 0, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, y 230400.

Parámetros

in	*serial*	Referencia a estructura "serial_port" que almacena los datos mínimos necesarios para abrir el puerto serie.
in	*baudrate*	Velocidad (data rate) en bits-por-segundo a la que se configura el puerto.

Valores devueltos

0	Éxito: Baudrate configurado correctamente.
-1	Error: No se ha podido configurar el baudrate, o el valor introducido no es correcto.

Ver también

setCommand(), setCommandParameter(), getChecksum(), packetXBee

Definición en la línea 113 del archivo SerialXBee.cpp.

4.2.3.4. int8_t SerialXBee::setSerialOptions (serial_port & *serial*, termios & *options*)

Establece las nuevas opciones de configuración de la interfaz serie.

Parámetros

in	*serial*	Referencia a estructura "serial_port" que almacena los datos mínimos necesarios para abrir el puerto serie.
in	*options*	Referencia a estructura tipo "termios" que almacena diferentes banderas y máscara de bits que controlan las opciones de configuración/control de la interfaz serie.

Valores devueltos

0	Éxito: Opciones establecidas correctamente en "termios".
-1	Error: No se ha podido configurar la interfaz.

Ver también

setCommand(), setCommandParameter(), getChecksum(), packetXBee

Definición en la línea 194 del archivo SerialXBee.cpp.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- SerialXBee.h
- SerialXBee.cpp

4.3. Referencia de la Clase UtilsXBee

Clase: Métodos auxiliares para tramas XBee, para realizar tareas concretas de forma especializada y optimizada.

`#include <UtilsXBee.h>`

Métodos públicos

- uint16_t bytesToB16 (uint8_t MSB, uint8_t LSB)

  Devuelve, en una variable de 16-bits (sin signo), el valor de la suma de 2 bytes (sin signo) independientes.
- uint32_t bytesToB32 (uint8_t MSB, uint8_t byte2, uint8_t byte3, uint8_t LSB)

  Devuelve, en una variable de 32-bits (sin signo), el valor de la suma de 4 bytes (sin signo) independientes.
- void convertFromEscapedBytes (uint8_t *frame, uint16_t lengthField, uint8_t escapedBytes=0, uint16_t start=0)

  Genera una trama API sin caracteres de escape (AP = 1) a partir de una trama que puede contener caracteres de escape (AP = 2).
- void convertToEscapedBytes (uint8_t *frame, uint16_t &packetLength)

  Genera una trama API sin caracteres de escape (AP = 1) a partir de una trama que puede contener caracteres de escape (AP = 2).
- void copyArray (const uint8_t *dataSource, uint8_t *dataDest, uint16_t maxBytes, uint16_t offsetSource=0, uint16_t offsetDest=0)

  Copia hasta un máximo de "maxBytes" datos de 8-bits, desde el array "dataSource[offsetSource]" al array "dataDest[offsetDest]".
- uint8_t getChecksum (uint8_t *frame)

  Calcula el checksum de la trama "frame".
- uint8_t getChecksum (char *frame)
- uint16_t getLengthFrame (const uint8_t *frame)

  Calcula la longitud total de una trama (incluyendo los bytes de Ïnicio de Trama, campo lengthFrame, y checksum).
- uint8_t getMSB (uint32_t data)

  Obtiene el MSB (Most Significant Byte) de cualquier entero sin signo de tamaño igual o menor a 32bits (uint8_t, uint16_t, uint32_t).
- void printFrame (const char *headString, const char *lineString, uint8_t *frame, uint16_t length, uint16_t offset=0)

  Imprime (en hexadecimal) por std::cout los valores de los diferentes bytes que componen una trama. Se debe incluir un mensaje de cabecera y una línea de texto que acompañará cada byte impreso.
- void printFrame (const char *headString, const char *lineString, const char *frame, uint16_t length, uint16_t offset=0)
- void printVerbose (const char *msg...)

  Función que imprime información de depuración (modo "verbose" activado) por std::cout, seguido de un número variable de argumentos, que pueden ser de tipo entero (%d), hexadecimal (%X), decimal (%f), o string (%s). No admite modificadores de formato (banderas, longitud, precisión, etc.) del tipo %.2X.
- void resetFormatFlags ()

  (inline) Limpia las banderas de formato modificadas por setOutputFlagsHex() Para ello utiliza el manipulador resetiosflags() (librería iomanip).
- void setDebugVar (bool basic=false, bool verbose=false)

  (inline) Activa (valor uno) o desactiva (valor cero) las banderas que controlan la impresión de información de depuración en modo "básico" o en modo "verbose" (detallado).
- void setOutputFlagsHex ()

  (inline) Modifica las banderas de formato de los flujos de salida std::cout y std::cerr para configurar la impresión de números hexadecimales con el formato 0x##, donde ## es cualquier valor hexadecimal valido (0-9, A-F)
- uint8_t str2hex (const char *str)

  Convierte un carácter en el valor literal (hexadecimal) correspondiente. Ejemplo! El carácter: char ID[] = "7E"; Se convierte en: uint8_t array8b[1] = {0x7E};.

- bool systemLittleEndian ()

  Informa si el sistema sobre el que se ejecuta la función es "Little Endian" o "Big Endian".
- uint32_t timer ()

  Devuelve el intervalo de tiempo (en mili-segundos) transcurrido desde el inicio de la ejecución del programa.
- void uSecondsPause (uint32_t interval=500)

  Permite configurar un intervalo de tiempo (en mili-segundos) de pausa.
- UtilsXBee ()

  Constructor por defecto de la clase UtilsXBee.

Atributos públicos

- bool basicEnabled

Atributos privados

- bool _verboseEnabled
- timeval end_point
- timeval start_program

4.3.1. Descripción detallada

Clase: Métodos auxiliares para tramas XBee, para realizar tareas concretas de forma especializada y optimizada.

Definición en la línea 106 del archivo UtilsXBee.h.

4.3.2. Documentación del constructor y destructor

4.3.2.1. UtilsXBee::UtilsXBee ( )

Constructor por defecto de la clase UtilsXBee.

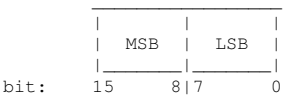
Configura el valor de inicio del timer mediante la llamada "gettimeofday(&start_program, NULL)" y configura las banderas que controlan la impresión de información de depuración mediante la función "setDebugVar(false, false)"

Definición en la línea 44 del archivo UtilsXBee.cpp.

4.3.3. Documentación de las funciones miembro

4.3.3.1. uint16_t UtilsXBee::bytesToB16 (uint8_t *MSB,* uint8_t *LSB*)

Devuelve, en una variable de 16-bits (sin signo), el valor de la suma de 2 bytes (sin signo) independientes.



Parámetros

in	*MSB*	Byte que ocupará el MSB del valor 16-bits devuelto.
in	*LSB*	Byte que ocupará el LSB del valor 16-bits devuelto.

Valores devueltos

Variable	16-bits uint16_t = MSB + LSB

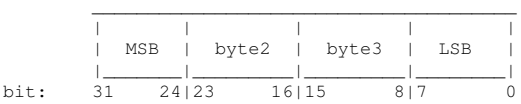
Ver también

bytesToB32(), getMSB()

Definición en la línea 60 del archivo UtilsXBee.cpp.

4.3.3.2. uint32_t UtilsXBee::bytesToB32 (uint8_t *MSB*, uint8_t *byte2*, uint8_t *byte3*, uint8_t *LSB*)

Devuelve, en una variable de 32-bits (sin signo), el valor de la suma de 4 bytes (sin signo) independientes.



Parámetros

in	*MSB*	Byte que ocupa los bits 31 a 24 del valor 32-bits.
in	*byte2*	Byte que ocupa los bits 23 a 16 del valor 32-bits.
in	*byte3*	Byte que ocupa los bits 15 a 8 del valor 32-bits.
in	*LSB*	Byte que ocupa los bits 7 a 0 del valor 32-bits.

Valores devueltos

Variable	32-bits uint32_t = MSB + byte2 + byte3 + LSB

Ver también

bytesToB16(), getMSB()

Definición en la línea 73 del archivo UtilsXBee.cpp.

4.3.3.3. void UtilsXBee::convertFromEscapedBytes (uint8_t * *frame*, uint16_t *lengthField*, uint8_t *escapedBytes* = 0, uint16_t *start* = 0)

Genera una trama API sin caracteres de escape (AP = 1) a partir de una trama que puede contener caracteres de escape (AP = 2).

Parámetros

in,out	*frame*	Puntero al array (trama) que contiene caracteres de escape.
in	*lengthField*	Valor (16-bits) del campo "lengthFrame" (bytes frame[1] y frame[2]) de la trama.
in	*escapedBytes*	Número de caracteres de escape que contiene la trama "frame". Nota: En una secuencia con caracteres de escape (por ejemplo, 0x11 = 0x7D + 0x31), "escapedBytes" solo tiene en cuenta los bytes 0x7D. Si no se conoce cuántos caracteres de escape contiene la trama, entonces "lengthFrame" debe contener la longitud completa de la trama (todos los bytes que contiene) y "escapedBytes = 0".
in	*start*	Offset desde dónde comenzar a buscar caracteres de escape (frame[start]).

Devuelve

void

Ver también

convertToEscapedBytes()

Definición en la línea 89 del archivo UtilsXBee.cpp.

4.3.3.4. void UtilsXBee::convertToEscapedBytes (uint8_t * *frame*, uint16_t & *packetLength*)

Genera una trama API sin caracteres de escape (AP = 1) a partir de una trama que puede contener caracteres de escape (AP = 2).

Parámetros

in,out	*frame*	Puntero al array (trama) dónde se buscarán bytes que deben convertirse en caracteres de escape. Al finalizar la función, "frame" será una trama con caracteres de escape (AP = 2).
in	*packetLength*	Longitud total (bytes) de la trama "frame". Nota: este valor se pasa por referencia, por lo que será actualizado fuera del ámbito de la función con la nueva longitud de la trama con caracteres de escape.

Devuelve

void

Ver también

convertFromEscapedBytes()

Definición en la línea 134 del archivo UtilsXBee.cpp.

4.3.3.5. void UtilsXBee::copyArray (const uint8_t * *dataSource,* uint8_t * *dataDest,* uint16_t *maxBytes,* uint16_t *offsetSource* = 0, uint16_t *offsetDest* = 0)

Copia hasta un máximo de "maxBytes" datos de 8-bits, desde el array "dataSource[offsetSource]" al array "data↩ Dest[offsetDest]".

Esta función no comprueba el tamaño de los arrays pasados como parámetros: copia exactamente "maxBytes" caracteres desde "dataSource" a "dataDest".

Para evitar desbordamientos, el usuario debe comprobar que el tamaño de los arrays origen/destino es, al menos, "dataSource[offsetSource + maxBytes]" y "dataDest[offsetDest + maxBytes]".

Parámetros

in	*dataSource*	Puntero al array origen que contiene los datos que se van a copiar.
out	*dataDest*	Puntero al array destino de los datos que serán copiados.
in	*maxBytes*	Número máximo de caracteres que serán copiados desde el array origen.
in	*offsetSource*	Offset de los datos en el array origen. Los datos serán copiados desde "dataSource[offsetSource]" en adelante, hasta alcanzar "dataSource[offsetSource + maxBytes]".
in	*offsetDest*	Offset de los datos en el array destino. Los datos serán copiados desde "dataDest[offsetDest]" en adelante, hasta alcanzar "dataDest[offsetDest + maxBytes]".

Devuelve

void

Definición en la línea 185 del archivo UtilsXBee.cpp.

4.3.3.6. uint8_t UtilsXBee::getChecksum (uint8_t * *frame*)

Calcula el checksum de la trama "frame".

El checksum se calcula sobre todos los carácteres "non-escaped" como [0xFF - (suma de 8-bits de todos los bytes desde frame[3] hasta el último byte de la trama sin incluir el propio byte de checksum].

Parámetros

in	*frame*	Puntero a la trama sobre la que se calcula el checksum.

Devuelve

Checksum calculado para la trama "frame".

Definición en la línea 201 del archivo UtilsXBee.cpp.

4.3.3.7. uint8_t UtilsXBee::getChecksum (char * *frame*)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

getChecksum()

Definición en la línea 224 del archivo UtilsXBee.cpp.

4.3.3.8. uint16_t UtilsXBee::getLengthFrame (const uint8_t * *frame*)

Calcula la longitud total de una trama (incluyendo los bytes de Ïnicio de Trama, campo lengthFrame, y checksum).

Parámetros

in	*frame*	Puntero al array que almacena todos los bytes de la tramaXBee de la que se obtendrá su longitud total.

Devuelve

Longitud total de la trama.

Definición en la línea 237 del archivo UtilsXBee.cpp.

4.3.3.9. uint8_t UtilsXBee::getMSB (uint32_t *data*)

Obtiene el MSB (Most Significant Byte) de cualquier entero sin signo de tamaño igual o menor a 32bits (uint8_t, uint16_t, uint32_t).

Parámetros

in	*data*	Variable de la que se desea obtener el MSB.

Valores devueltos

MSB	Byte Más Significativo (MSB)

Ver también

bytesToB16(), bytesToB32()

Definición en la línea 253 del archivo UtilsXBee.cpp.

4.3.3.10. void UtilsXBee::printFrame (const char * *headString*, const char * *lineString*, uint8_t * *frame*, uint16_t *length*, uint16_t *offset* = 0)

Imprime (en hexadecimal) por std::cout los valores de los diferentes bytes que componen una trama. Se debe incluir un mensaje de cabecera y una línea de texto que acompañará cada byte impreso.

Ejemplo de uso de esta función:

```
uint8_t payload[11] = {0x21, 0x7E, 0x65, 0x8b, 0x74, 0x11, 0x13, 0x41};

utils.printFrame( "Print Payload content!", "FrameByte", payload,
                  sizeof(payload) );
```

Al ejecutar el anterior código en un terminal de comandos, se visualizará:

```
*** Print Payload content! ***
--> FrameByte[  ]:  0x21
--> FrameByte[ 1]:  0x7E
--> FrameByte[ 2]:  0x65
--> FrameByte[ 3]:  0x8B
--> FrameByte[ 4]:  0x74
--> FrameByte[ 5]:  0x11
--> FrameByte[ 6]:  0x13
--> FrameByte[ 7]:  0x41
--> FrameByte[ 8]:  0x00
--> FrameByte[ 9]:  0x00
--> FrameByte[10]:  0x00
```

Parámetros

in	*headString*	Mensaje de cabecera.
in	*lineString*	Nombre que acompaña al índice del valor impreso.
in	*frame*	Puntero a la trama que se imprime.
in	*length*	Longitud (bytes) a imprimir de la trama "frame".
in	*offset*	Offset de los bytes a imprimir.

Devuelve

void

Definición en la línea 268 del archivo UtilsXBee.cpp.

4.3.3.11. void UtilsXBee::printFrame (const char * *headString*, const char * *lineString*, const char * *frame*, uint16_t *length*, uint16_t *offset* = 0)

Esta es una función miembro sobrecargada que se suministra por conveniencia. Difiere de la anterior función solamente en los argumentos que acepta.

Ver también

printFrame()

Definición en la línea 292 del archivo UtilsXBee.cpp.

4.3.3.12. void UtilsXBee::printVerbose (const char * *msg...*)

Función que imprime información de depuración (modo "verbose" activado) por std::cout, seguido de un número variable de argumentos, que pueden ser de tipo entero (%d), hexadecimal (%X), decimal (%f), o string (%s). No admite modificadores de formato (banderas, longitud, precisión, etc.) del tipo %.2X.

Ejemplo de uso de esta función:

```
uint8_t payload[] = {0x21, 0x7E, 0x65, 0x8b, 0x74, 0x11, 0x13, 0x41};
char mensaje[] = "Trama FrameByte!!!";
double time = 4.521;

utils.printVerbose("%s Recibida hace: %f [segundos]\nLongitud: %d,Checksum:
    0x%X\n", mensaje, time, sizeof(payload), payload[sizeof(payload)-1]);
```

Al ejecutar el anterior código en un terminal de comandos, se visualizará:

```
Trama FrameByte!!! Recibida hace: 4.521000 [segundos]
Longitud: 8, Checksum: 0x41
```

Parámetros

in	*msg*	Mensaje que se imprime por pantalla (std::cout).
in	...	Cualquier argumento extra que sigue a "msg".

Devuelve

void

Ver también

BASIC_INFO(), setDebugVar()

Definición en la línea 310 del archivo UtilsXBee.cpp.

4.3.3.13. void UtilsXBee::resetFormatFlags ( ) [inline]

(inline) Limpia las banderas de formato modificadas por setOutputFlagsHex() Para ello utiliza el manipulador resetiosflags() (librería iomanip).

Devuelve

void

Ver también

setOutputFlagsHex()

Definición en la línea 594 del archivo UtilsXBee.h.

4.3.3.14. void UtilsXBee::setDebugVar (bool *basic* = false, bool *verbose* = false) [inline]

(inline) Activa (valor uno) o desactiva (valor cero) las banderas que controlan la impresión de información de depuración en modo "básico" o en modo "verbose" (detallado).

Parámetros

in	*basic*	Bandera: controla la impresión de información básica de depuración mediante la macro BASIC_INFO(). Valor por defecto: false.
in	*verbose*	Bandera: controla la impresión de información detallada de depuración mediante la función printVerbose(). Valor por defecto: false.

Devuelve

void

Ver también

BASIC_INFO(), printVerbose()

Definición en la línea 619 del archivo UtilsXBee.h.

4.3.3.15. void UtilsXBee::setOutputFlagsHex ( ) [inline]

(inline) Modifica las banderas de formato de los flujos de salida std::cout y std::cerr para configurar la impresión de números hexadecimales con el formato 0x##, donde ## es cualquier valor hexadecimal valido (0-9, A-F)

Devuelve

void

Ver también

resetFormatFlags()

Definición en la línea 634 del archivo UtilsXBee.h.

4.3.3.16. uint8_t UtilsXBee::str2hex (const char * *str*)

Convierte un carácter en el valor literal (hexadecimal) correspondiente. Ejemplo! El carácter: char ID[] = "7E"; Se convierte en: uint8_t array8b[1] = {0x7E};.

Devuelve

Valor hexadecimal equivalente al carácter "str".

Definición en la línea 362 del archivo UtilsXBee.cpp.

4.3.3.17. bool UtilsXBee::systemLittleEndian ( )

Informa si el sistema sobre el que se ejecuta la función es "Little Endian" o "Big Endian".

Valores devueltos

true	El sistema es "Little Endian".
false	El sistema es "Big Endian".

Definición en la línea 397 del archivo UtilsXBee.cpp.

4.3.3.18. uint32_t UtilsXBee::timer ( )

Devuelve el intervalo de tiempo (en mili-segundos) transcurrido desde el inicio de la ejecución del programa.

Internamente, la función extrae los valores de la estructura "timeval start_program" inicializada al instanciar la clase Utils, ("timeval start_program" se puede re-iniciar en cualquier momento haciendo uso de "gettimeofday(&start_↩ program, NULL)"), y calcula la diferencia con el valor de tiempo en el momento de ejecutar esta función "timer()".

Devuelve

Intervalo de tiempo calculado (32-bits sin signo). Equivale a un máximo de 50 días aproximadamente.

Ver también

uSecondsPause()

Definición en la línea 439 del archivo UtilsXBee.cpp.

4.3.3.19. void UtilsXBee::uSecondsPause (uint32_t *interval* = `500`)

Permite configurar un intervalo de tiempo (en mili-segundos) de pausa.

Parámetros

`in`	*interval*	Intervalo de tiempo que esta función espera antes de retornar. Valor máximo configurable: 32-bits

Devuelve

void

Ver también

timer()

Definición en la línea 463 del archivo UtilsXBee.cpp.

4.3.4. Documentación de los datos miembro

4.3.4.1. bool UtilsXBee::_verboseEnabled `[private]`

Variable privada: Bandera que controla la impresión de información detallada de depuración mediante la función printVerbose(). Para modificar su valor, utilizar la función setDebugVar().

Definición en la línea 126 del archivo UtilsXBee.h.

4.3.4.2. bool UtilsXBee::basicEnabled

Variable pública: Bandera: controla la impresión de información básica de depuración mediante la macro BASIC↩ _INFO(). Para modificar su valor, utilizar la función setDebugVar().

Definición en la línea 572 del archivo UtilsXBee.h.

4.3.4.3. timeval UtilsXBee::end_point [private]

Definición en la línea 119 del archivo UtilsXBee.h.

4.3.4.4. timeval UtilsXBee::start_program [private]

Variables privadas: Estructura devuelta por una llamada a la función de sistema gettimeofday(), que es usada en la función timer() de esta librería.

struct timeval { time_t tv_sec; // segundos suseconds_t tv_usec; // y micro-segundos };

Definición en la línea 119 del archivo UtilsXBee.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- UtilsXBee.h
- UtilsXBee.cpp

4.4. Referencia de la Estructura packetXBee

Estructura: Contiene los campos para almacenar la información y datos necesarios para enviar un paquete/trama a un módulo XBee. Las variables de la estructura hacen referencia tanto a campos propios de las tramas XBee para transmisión de datos y comandos AT como a variables auxiliares de utilidad para ciertas funciones de esta librería. Esta estructura es usada por gran parte de las funciones de la clase 'FrameXBee'.

```
#include <FrameXBee.h>
```

Atributos públicos

- uint8_t CID [2]

  Variable: Cluster ID usado en la transmisión.
- uint8_t commandAT [2]

  Variable: Comando AT que se envía al nodo/dispositivo de destino.
- uint8_t commandParameter [MAX_COMMAND_LENGTH]

  Variable: Parámetro del Comando AT que se envía al nodo de destino.
- uint8_t destinationEndpoint

  Variable: Nodo extremo al que se envía el mensaje como destino final.
- uint8_t destinationMAC [8]

  Variable: Identifica la dirección MAC (64-bits) del dispositivo de destino.
- uint8_t frameID

  Variable: Identifica la trama de datos UART del host.
- uint16_t frameLength

  Variable Auxiliar: Longitud total (bytes) de la trama (campo 'length' + 4)
- uint8_t frameType

  Variable: Tipo de trama usada.
- uint8_t hopsRadius

  Variable: Número máximo de saltos permitidos en una transmisión broadcast.
- uint8_t parameterLength

  Variable Auxiliar: Longitud total (bytes) de datos en 'commandParameter[]'.
- uint8_t payload [MAX_PAYLOAD_LENGTH]

  Variable: Buffer de datos que se envían al nodo/dispositivo de destino.
- uint16_t payloadLength

  Variable Auxiliar: Longitud total (bytes) de datos en 'payload[]'.
- uint8_t PID [2]

  Variable: Profile ID usado en la transmisión.
- uint16_t previousLength

  Variable Auxiliar: Longitud (bytes) de datos contenidos en 'payload[]'.
- uint8_t remoteCommandOptions

  Variable: Opciones de transmisión para comandos AT (bitfield).
- uint8_t sourceEndpoint

  Variable: Nodo extremo que originó la transmisión del mensaje.
- uint8_t txOptions

  Variable: Opciones de transmisión (bitfield).

4.4.1. Descripción detallada

Estructura: Contiene los campos para almacenar la información y datos necesarios para enviar un paquete/trama a un módulo XBee. Las variables de la estructura hacen referencia tanto a campos propios de las tramas XBee para transmisión de datos y comandos AT como a variables auxiliares de utilidad para ciertas funciones de esta librería. Esta estructura es usada por gran parte de las funciones de la clase 'FrameXBee'.

Definición en la línea 336 del archivo FrameXBee.h.

4.4.2. Documentación de los datos miembro

4.4.2.1. uint8_t packetXBee::CID[2]

Variable: Cluster ID usado en la transmisión.

Definición en la línea 420 del archivo FrameXBee.h.

4.4.2.2. uint8_t packetXBee::commandAT[2]

Variable: Comando AT que se envía al nodo/dispositivo de destino.

Definición en la línea 446 del archivo FrameXBee.h.

4.4.2.3. uint8_t packetXBee::commandParameter[MAX_COMMAND_LENGTH]

Variable: Parámetro del Comando AT que se envía al nodo de destino.

Definición en la línea 450 del archivo FrameXBee.h.

4.4.2.4. uint8_t packetXBee::destinationEndpoint

Variable: Nodo extremo al que se envía el mensaje como destino final.

Definición en la línea 416 del archivo FrameXBee.h.

4.4.2.5. uint8_t packetXBee::destinationMAC[8]

Variable: Identifica la dirección MAC (64-bits) del dispositivo de destino.

destinationMAC[0] = Most Significant Byte destinationMAC[7] = Less Significant Byte

Definición en la línea 392 del archivo FrameXBee.h.

4.4.2.6. uint8_t packetXBee::frameID

Variable: Identifica la trama de datos UART del host.

frameID relaciona la trama enviada con el correspondiente ACK. Si frameID = 0x00, no se recibirá ACK.

Definición en la línea 384 del archivo FrameXBee.h.

4.4.2.7. uint16_t packetXBee::frameLength

Variable Auxiliar: Longitud total (bytes) de la trama (campo 'length' + 4)

frameLength = Campo 'Length' de la trama + Campo inicio de trama (1 byte = 0x7E) + Campo Length (2 bytes) + Campo checksum (1 byte).

Definición en la línea 347 del archivo FrameXBee.h.

4.4.2.8. uint8_t packetXBee::frameType

Variable: Tipo de trama usada.

0x08: AT Command Request (LOCAL_COMMAND_TX) 0x10: Transmit Request (NORMAL_TX mode) 0x11: Explicit Addressing Command Frame (EXPLICIT_TX mode) 0x17: Remote AT Command Request (COMMAND_TX mode)

Definición en la línea 376 del archivo FrameXBee.h.

4.4.2.9. uint8_t packetXBee::hopsRadius

Variable: Número máximo de saltos permitidos en una transmisión broadcast.

Si hopsRadius = 0x00 'broadcast radius' se configurará a su máximo valor.

Definición en la línea 399 del archivo FrameXBee.h.

4.4.2.10. uint8_t packetXBee::parameterLength

Variable Auxiliar: Longitud total (bytes) de datos en 'commandParameter[]'.

Definición en la línea 363 del archivo FrameXBee.h.

4.4.2.11. uint8_t packetXBee::payload[MAX_PAYLOAD_LENGTH]

Variable: Buffer de datos que se envían al nodo/dispositivo de destino.

Definición en la línea 428 del archivo FrameXBee.h.

4.4.2.12. uint16_t packetXBee::payloadLength

Variable Auxiliar: Longitud total (bytes) de datos en 'payload[]'.

Definición en la línea 359 del archivo FrameXBee.h.

4.4.2.13. uint8_t packetXBee::PID[2]

Variable: Profile ID usado en la transmisión.

Definición en la línea 424 del archivo FrameXBee.h.

4.4.2.14. uint16_t packetXBee::previousLength

Variable Auxiliar: Longitud (bytes) de datos contenidos en 'payload[]'.

Esta variable almacena la longitud previa (bytes) de los datos contenidos en el buffer 'payload[]' antes de concatenar nuevos datos en el buffer.

Definición en la línea 355 del archivo FrameXBee.h.

4.4.2.15. uint8_t packetXBee::remoteCommandOptions

Variable: Opciones de transmisión para comandos AT (bitfield).

bit0 = 1 (0x01): ACK desactivado. bit1 = 1 (0x02): Aplicar cambios inmediatamente en el dispositivo remoto. Los demás bits deben ser 0.

Definición en la línea 442 del archivo FrameXBee.h.

4.4.2.16. uint8_t packetXBee::sourceEndpoint

Variable: Nodo extremo que originó la transmisión del mensaje.

Definición en la línea 412 del archivo FrameXBee.h.

4.4.2.17. uint8_t packetXBee::txOptions

Variable: Opciones de transmisión (bitfield).

bit0 = 1 (0x01): ACK desactivado. bit1 = 1 (0x02): No realizar "route discovery". Los demás bits deben ser 0.

Definición en la línea 408 del archivo FrameXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- FrameXBee.h

4.5. Referencia de la Unión received

Unión: Facilita la extracción/interpretación de los datos recibidos/leídos.

`#include <FrameXBee.h>`

Atributos públicos

- uint8_t bytesRX [MAX_FRAME_LENGTH]

  Buffer de datos recibidos/leídos.
- rxPacket88 rx88

  Estructura: campos de trama 0x88(LOCAL_COMMAND_RX)
- rxPacket8B rx8A

  Estructura: campos de trama 0x8A (MODEM_STATUS)
- rxPacket8B rx8B

  Estructura: campos de trama 0x8B (TX_STATUS)
- rxPacket90 rx90

  Estructura: campos de trama 0x90 (NORMAL_RX)
- rxPacket91 rx91

  Estructura: campos de trama 0x91 (EXPLICIT_RX)
- rxPacket92 rx92

  Estructura: campos de trama 0x92 (SAMPLES_RX)
- rxPacket97 rx97

  Estructura: campos de trama 0x97 (COMMAND_RX)

4.5.1. Descripción detallada

Unión: Facilita la extracción/interpretación de los datos recibidos/leídos.

Esta unión permite extraer la información del campo concreto requerido por el usuario, de una trama tipo 0x88, 0x8B, 0x90, 0x91, 0x92, 0x97, a partir de un array (buffer bytesRX[]) de datos que han debido ser previamente leidos por la función 'FrameXBee::readXBee()'.

Definición en la línea 306 del archivo FrameXBee.h.

4.5.2. Documentación de los datos miembro

4.5.2.1. uint8_t received::bytesRX[MAX_FRAME_LENGTH]

Buffer de datos recibidos/leídos.

Definición en la línea 308 del archivo FrameXBee.h.

4.5.2.2. rxPacket88 received::rx88

Estructura: campos de trama 0x88(LOCAL_COMMAND_RX)

Definición en la línea 309 del archivo FrameXBee.h.

4.5.2.3. rxPacket8B received::rx8A

Estructura: campos de trama 0x8A (MODEM_STATUS)

Definición en la línea 310 del archivo FrameXBee.h.

4.5.2.4. rxPacket8B received::rx8B

Estructura: campos de trama 0x8B (TX_STATUS)

Definición en la línea 311 del archivo FrameXBee.h.

4.5.2.5. rxPacket90 received::rx90

Estructura: campos de trama 0x90 (NORMAL_RX)

Definición en la línea 312 del archivo FrameXBee.h.

4.5.2.6. rxPacket91 received::rx91

Estructura: campos de trama 0x91 (EXPLICIT_RX)

Definición en la línea 313 del archivo FrameXBee.h.

4.5.2.7. rxPacket92 received::rx92

Estructura: campos de trama 0x92 (SAMPLES_RX)

Definición en la línea 314 del archivo FrameXBee.h.

4.5.2.8. rxPacket97 received::rx97

Estructura: campos de trama 0x97 (COMMAND_RX)

Definición en la línea 315 del archivo FrameXBee.h.

La documentación para esta unión fue generada a partir del siguiente fichero:

- FrameXBee.h

4.6. Referencia de la Estructura rxPacket88

Estructuras 'rxPacket' - Define estructuras/uniones para almacenar, de forma ordenada, todos los bytes de cualquier trama XBee-Pro 868 que sea leída en recepción por la función 'FrameXBee::readXBee()'. Información adicional en el Manual de Usuario del Protocolo de los módulos XBee-Pro 868. Nota: En las tramas dónde el campo "checksum" viene precedido de un campo de longitud variable (payload, commandData, samplesData), se ha omitido el campo "checksum", porque su posición dentro de la trama dependerá de la longitud de dichos datos.

```
#include <FrameXBee.h>
```

Atributos públicos

- uint8_t commandAT [2]

  Comando AT enviado previamente en trama 0x08.
- uint8_t commandData [MAX_COMMAND_LENGTH]

  Datos/parámetro del comando AT.
- uint8_t commandStatus

  Información sobre el comando AT.
- uint8_t frameID

  Identificador de trama (para comprobar ACK)
- uint8_t frameType

  Tipo de trama (en este caso: siempre 0x88)
- uint8_t length [2]

  Longitud trama (excluyendo: start+length+checksum)
- uint8_t start

  Byte de Inicio de Trama (siempre 0x7E)

4.6.1. Descripción detallada

Estructuras 'rxPacket' - Define estructuras/uniones para almacenar, de forma ordenada, todos los bytes de cualquier trama XBee-Pro 868 que sea leída en recepción por la función 'FrameXBee::readXBee()'. Información adicional en el Manual de Usuario del Protocolo de los módulos XBee-Pro 868. Nota: En las tramas dónde el campo "checksum" viene precedido de un campo de longitud variable (payload, commandData, samplesData), se ha omitido el campo "checksum", porque su posición dentro de la trama dependerá de la longitud de dichos datos.

Estructura: Contiene los campos de la trama Rx tipo 0x88 (LOCAL_COMMAND_RX)

Esta trama sólo se recibe si previamente se ha enviado a través de la UART el módulo (conectado en local) una trama tipo 0x08 (LOCAL_COMMAND_TX).

Definición en la línea 150 del archivo FrameXBee.h.

4.6.2. Documentación de los datos miembro

4.6.2.1. uint8_t rxPacket88::commandAT[2]

Comando AT enviado previamente en trama 0x08.

Definición en la línea 156 del archivo FrameXBee.h.

4.6.2.2. uint8_t rxPacket88::commandData[MAX_COMMAND_LENGTH]

Datos/parámetro del comando AT.

Definición en la línea 158 del archivo FrameXBee.h.

4.6.2.3. uint8_t rxPacket88::commandStatus

Información sobre el comando AT.

Definición en la línea 157 del archivo FrameXBee.h.

4.6.2.4. uint8_t rxPacket88::frameID

Identificador de trama (para comprobar ACK)

Definición en la línea 155 del archivo FrameXBee.h.

4.6.2.5. uint8_t rxPacket88::frameType

Tipo de trama (en este caso: siempre 0x88)

Definición en la línea 154 del archivo FrameXBee.h.

4.6.2.6. uint8_t rxPacket88::length[2]

Longitud trama (excluyendo: start+length+checksum)

Definición en la línea 153 del archivo FrameXBee.h.

4.6.2.7. uint8_t rxPacket88::start

Byte de Inicio de Trama (siempre 0x7E)

Definición en la línea 152 del archivo FrameXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- FrameXBee.h

4.7. Referencia de la Estructura rxPacket8A

Estructura: Contiene los campos de la trama Rx tipo 0x8A (MODEM_STATUS).

```
#include <FrameXBee.h>
```

Atributos públicos

- uint8_t checksum
  Checksum de la trama.
- uint8_t frameType
  Tipo de trama (en este caso: siempre 0x8A)
- uint8_t length [2]
  Longitud trama (excluyendo: start+length+checksum)
- uint8_t start
  Byte de Inicio de Trama (siempre 0x7E)
- uint8_t status
  Información sobre el comando AT.

4.7.1. Descripción detallada

Estructura: Contiene los campos de la trama Rx tipo 0x8A (MODEM_STATUS).

Esta trama es recibida por RF desde un módulo remoto en respuesta a determinadas condiciones, indicadas en el campo "status".

Definición en la línea 170 del archivo FrameXBee.h.

4.7.2. Documentación de los datos miembro

4.7.2.1. uint8_t rxPacket8A::checksum

Checksum de la trama.

Definición en la línea 176 del archivo FrameXBee.h.

4.7.2.2. uint8_t rxPacket8A::frameType

Tipo de trama (en este caso: siempre 0x8A)

Definición en la línea 174 del archivo FrameXBee.h.

4.7.2.3. uint8_t rxPacket8A::length[2]

Longitud trama (excluyendo: start+length+checksum)

Definición en la línea 173 del archivo FrameXBee.h.

4.7.2.4. uint8_t rxPacket8A::start

Byte de Inicio de Trama (siempre 0x7E)

Definición en la línea 172 del archivo FrameXBee.h.

4.7.2.5. uint8_t rxPacket8A::status

Información sobre el comando AT.

Definición en la línea 175 del archivo FrameXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- FrameXBee.h

4.8. Referencia de la Estructura rxPacket8B

Estructura: Contiene los campos de la trama Rx tipo 0x8B (TX_STATUS).

`#include <FrameXBee.h>`

Atributos públicos

- uint8_t checksum

  Checksum de la trama.
- uint8_t deliveryStatus

  Estado de la transmisión.
- uint8_t discoveryStatus

  Estado del 'Descubrimiento de Ruta'.
- uint8_t frameID

  Identificador de trama (para comprobar ACK)
- uint8_t frameType

  Tipo de trama (en este caso: siempre 0x8B)
- uint8_t length [2]

  Longitud trama (excluyendo: start+length+checksum)
- uint8_t reserved [2]

  Campo reservado (siempre 0xFFFE)
- uint8_t retriesTx

  Retransmisiones realizadas.
- uint8_t start

  Byte de Inicio de Trama (siempre 0x7E)

4.8.1. Descripción detallada

Estructura: Contiene los campos de la trama Rx tipo 0x8B (TX_STATUS).

Esta trama sólo se recibe si previamente se ha enviado a través de la UART del módulo (conectado en local) una trama tipo 0x10 (NORMAL_TX) o 0x11 (EXPLICIT_TX) hacia un nodo en remoto. Informa del éxito/fracaso de dicha transmisión.

Definición en la línea 188 del archivo FrameXBee.h.

4.8.2. Documentación de los datos miembro

4.8.2.1. uint8_t rxPacket8B::checksum

Checksum de la trama.

Definición en la línea 198 del archivo FrameXBee.h.

4.8.2.2. uint8_t rxPacket8B::deliveryStatus

Estado de la transmisión.

Definición en la línea 196 del archivo FrameXBee.h.

4.8.2.3. uint8_t rxPacket8B::discoveryStatus

Estado del 'Descubrimiento de Ruta'.

Definición en la línea 197 del archivo FrameXBee.h.

4.8.2.4. uint8_t rxPacket8B::frameID

Identificador de trama (para comprobar ACK)

Definición en la línea 193 del archivo FrameXBee.h.

4.8.2.5. uint8_t rxPacket8B::frameType

Tipo de trama (en este caso: siempre 0x8B)

Definición en la línea 192 del archivo FrameXBee.h.

4.8.2.6. uint8_t rxPacket8B::length[2]

Longitud trama (excluyendo: start+length+checksum)

Definición en la línea 191 del archivo FrameXBee.h.

4.8.2.7. uint8_t rxPacket8B::reserved[2]

Campo reservado (siempre 0xFFFE)

Definición en la línea 194 del archivo FrameXBee.h.

4.8.2.8. uint8_t rxPacket8B::retriesTx

Retransmisiones realizadas.

Definición en la línea 195 del archivo FrameXBee.h.

4.8.2.9. uint8_t rxPacket8B::start

Byte de Inicio de Trama (siempre 0x7E)

Definición en la línea 190 del archivo FrameXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- FrameXBee.h

4.9. Referencia de la Estructura rxPacket90

Estructura: Contiene los campos de la trama Rx tipo 0x90 (NORMAL_RX).

`#include <FrameXBee.h>`

Atributos públicos

- uint8_t frameType
  Tipo de trama (en este caso: siempre 0x90)
- uint8_t length [2]
  Longitud trama (excluyendo: start+length+checksum)
- uint8_t options
  Opciones de transmisión de la trama recibida.
- uint8_t payload [MAX_PAYLOAD_LENGTH]
  Campo de datos(carga útil)
- uint8_t reserved [2]
  Campo reservado (siempre 0xFFFE)
- uint8_t sourceMAC [8]
  Dirección MAC (64-bits): nodo origen de la trama.
- uint8_t start
  Byte de Inicio de Trama (siempre 0x7E)

4.9.1. Descripción detallada

Estructura: Contiene los campos de la trama Rx tipo 0x90 (NORMAL_RX).

Esta trama sólo se recibe si AO = 0 (parámetro del módulo XBee) cuando el módulo recibe por RF una trama tipo 0x10 (NORMAL_TX) o 0x11 (EXPLICIT_TX).

Definición en la línea 209 del archivo FrameXBee.h.

4.9.2. Documentación de los datos miembro

4.9.2.1. uint8_t rxPacket90::frameType

Tipo de trama (en este caso: siempre 0x90)

Definición en la línea 213 del archivo FrameXBee.h.

4.9.2.2. uint8_t rxPacket90::length[2]

Longitud trama (excluyendo: start+length+checksum)

Definición en la línea 212 del archivo FrameXBee.h.

4.9.2.3. uint8_t rxPacket90::options

Opciones de transmisión de la trama recibida.

Definición en la línea 216 del archivo FrameXBee.h.

4.9.2.4. uint8_t rxPacket90::payload[MAX_PAYLOAD_LENGTH]

Campo de datos(carga útil)

Definición en la línea 217 del archivo FrameXBee.h.

4.9.2.5. uint8_t rxPacket90::reserved[2]

Campo reservado (siempre 0xFFFE)

Definición en la línea 215 del archivo FrameXBee.h.

4.9.2.6. uint8_t rxPacket90::sourceMAC[8]

Dirección MAC (64-bits): nodo origen de la trama.

Definición en la línea 214 del archivo FrameXBee.h.

4.9.2.7. uint8_t rxPacket90::start

Byte de Inicio de Trama (siempre 0x7E)

Definición en la línea 211 del archivo FrameXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- FrameXBee.h

4.10. Referencia de la Estructura rxPacket91

Estructura: Contiene los campos de la trama Rx tipo 0x91 (EXPLICIT_RX).

```
#include <FrameXBee.h>
```

Atributos públicos

- uint8_t clusterID [2]

  Identificador de Cluster.
- uint8_t dstEndpoint

  Nodo "endpoint" destino de la transmisión.
- uint8_t frameType

  Tipo de trama (en este caso: siempre 0x91)
- uint8_t length [2]

  Longitud trama (excluyendo: start+length+checksum)
- uint8_t options

  Opciones de transmisión de la trama recibida.
- uint8_t payload [MAX_PAYLOAD_LENGTH]

  Campo de datos (carga útil)
- uint8_t profileID [2]

  Identificador de Perfil.
- uint8_t reserved [2]

  Campo reservado (siempre 0xFFFE)
- uint8_t sourceMAC [8]

  Dirección MAC (64b): nodo origen de la trama.
- uint8_t srcEndpoint

  Nodo "endpoint" origen de la transmisión.
- uint8_t start

  Byte de Inicio de Trama (siempre 0x7E)

4.10.1. Descripción detallada

Estructura: Contiene los campos de la trama Rx tipo 0x91 (EXPLICIT_RX).

Esta trama sólo se recibe si AO = 1 (parámetro del módulo XBee) cuando el módulo recibe por RF una trama tipo 0x10 (NORMAL_TX) o 0x11 (EXPLICIT_TX).

Definición en la línea 228 del archivo FrameXBee.h.

4.10.2. Documentación de los datos miembro

4.10.2.1. uint8_t rxPacket91::clusterID[2]

Identificador de Cluster.

Definición en la línea 237 del archivo FrameXBee.h.

4.10.2.2. uint8_t rxPacket91::dstEndpoint

Nodo "endpoint" destino de la transmisión.

Definición en la línea 236 del archivo FrameXBee.h.

4.10.2.3. uint8_t rxPacket91::frameType

Tipo de trama (en este caso: siempre 0x91)

Definición en la línea 232 del archivo FrameXBee.h.

4.10.2.4. uint8_t rxPacket91::length[2]

Longitud trama (excluyendo: start+length+checksum)

Definición en la línea 231 del archivo FrameXBee.h.

4.10.2.5. uint8_t rxPacket91::options

Opciones de transmisión de la trama recibida.

Definición en la línea 239 del archivo FrameXBee.h.

4.10.2.6. uint8_t rxPacket91::payload[MAX_PAYLOAD_LENGTH]

Campo de datos (carga útil)

Definición en la línea 240 del archivo FrameXBee.h.

4.10.2.7. uint8_t rxPacket91::profileID[2]

Identificador de Perfil.

Definición en la línea 238 del archivo FrameXBee.h.

4.10.2.8. uint8_t rxPacket91::reserved[2]

Campo reservado (siempre 0xFFFE)

Definición en la línea 234 del archivo FrameXBee.h.

4.10.2.9. uint8_t rxPacket91::sourceMAC[8]

Dirección MAC (64b): nodo origen de la trama.

Definición en la línea 233 del archivo FrameXBee.h.

4.10.2.10. uint8_t rxPacket91::srcEndpoint

Nodo "endpoint" origen de la transmisión.

Definición en la línea 235 del archivo FrameXBee.h.

4.10.2.11. uint8_t rxPacket91::start

Byte de Inicio de Trama (siempre 0x7E)

Definición en la línea 230 del archivo FrameXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- FrameXBee.h

4.11. Referencia de la Estructura rxPacket92

Estructura: Contiene los campos de la trama Rx tipo 0x92 (SAMPLES_RX).

`#include <FrameXBee.h>`

Atributos públicos

- uint8_t analogMask
  Máscara de bits - Entradas ADC activas.
- uint8_t digitalMask [2]
  Máscara de bits - Entradas digitales activas.
- uint8_t frameType
  Tipo de trama (en este caso: siempre 0x92)
- uint8_t length [2]
  Longitud trama (excluyendo: start+length+checksum)
- uint8_t numberSamples
  Conjuntos de muestras incluidos (siempre 0x01)
- uint8_t options
  Opciones de transmisión de la trama recibida.
- uint8_t reserved [2]
  Campo reservado (siempre 0xFFFE)
- uint8_t samplesData [MAX_SAMPLES_LENGTH]
  Muestras Digitales y ADC.
- uint8_t sourceMAC [8]
  Dirección MAC (64b): nodo origen de la trama.
- uint8_t start
  Byte de Inicio de Trama (siempre 0x7E)

4.11.1. Descripción detallada

Estructura: Contiene los campos de la trama Rx tipo 0x92 (SAMPLES_RX).

Esta trama sólo se recibe si se ha configurado el módulo XBee para realizar de forma automática un muestreo analógico (ADC) o digital usando las opciones Sample Rate (IR) o Change Detect (IC). En caso de "forzar" la toma de muestras mediante el envío de un comando AT tipo "IS" (mediante la trama 0x17 - COMMAND_TX), los datos se recibiran en una trama tipo 0x97 (COMMAND_RX), en cuyo caso, los datos correspondientes a los campos "numberSamples", "digitalMask", "analogMask", y "samplesData", estarán contenidos en el campo "command↩ Data" de la trama 0x97.

Definición en la línea 257 del archivo FrameXBee.h.

4.11.2. Documentación de los datos miembro

4.11.2.1. uint8_t rxPacket92::analogMask

Máscara de bits - Entradas ADC activas.

Definición en la línea 267 del archivo FrameXBee.h.

4.11.2.2. uint8_t rxPacket92::digitalMask[2]

Máscara de bits - Entradas digitales activas.

Definición en la línea 266 del archivo FrameXBee.h.

4.11.2.3. uint8_t rxPacket92::frameType

Tipo de trama (en este caso: siempre 0x92)

Definición en la línea 261 del archivo FrameXBee.h.

4.11.2.4. uint8_t rxPacket92::length[2]

Longitud trama (excluyendo: start+length+checksum)

Definición en la línea 260 del archivo FrameXBee.h.

4.11.2.5. uint8_t rxPacket92::numberSamples

Conjuntos de muestras incluidos (siempre 0x01)

Definición en la línea 265 del archivo FrameXBee.h.

4.11.2.6. uint8_t rxPacket92::options

Opciones de transmisión de la trama recibida.

Definición en la línea 264 del archivo FrameXBee.h.

4.11.2.7. uint8_t rxPacket92::reserved[2]

Campo reservado (siempre 0xFFFE)

Definición en la línea 263 del archivo FrameXBee.h.

4.11.2.8. uint8_t rxPacket92::samplesData[MAX_SAMPLES_LENGTH]

Muestras Digitales y ADC.

Definición en la línea 268 del archivo FrameXBee.h.

4.11.2.9. uint8_t rxPacket92::sourceMAC[8]

Dirección MAC (64b): nodo origen de la trama.

Definición en la línea 262 del archivo FrameXBee.h.

4.11.2.10. uint8_t rxPacket92::start

Byte de Inicio de Trama (siempre 0x7E)

Definición en la línea 259 del archivo FrameXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- FrameXBee.h

4.12. Referencia de la Estructura rxPacket97

Estructura: Contiene los campos de la trama Rx tipo 0x97 (COMMAND_RX).

```
#include <FrameXBee.h>
```

Atributos públicos

- uint8_t commandAT [2]
  Comando AT enviado previamente en trama 0x17.
- uint8_t commandData [MAX_COMMAND_LENGTH]
  Datos/parámetro del comando AT.
- uint8_t commandStatus
  Información sobre el comando AT.
- uint8_t frameID
  Identificador de trama (para comprobar ACK)
- uint8_t frameType
  Tipo de trama (en este caso: siempre 0x97)
- uint8_t length [2]
  Longitud trama (excluyendo: start+length+checksum)
- uint8_t reserved [2]
  Campo reservado (siempre 0xFFFE)
- uint8_t sourceMAC [8]
  Dirección MAC (64b): nodo origen de la trama.
- uint8_t start
  Byte de Inicio de Trama (siempre 0x7E)

4.12.1. Descripción detallada

Estructura: Contiene los campos de la trama Rx tipo 0x97 (COMMAND_RX).

Esta trama sólo se recibe si previamente se ha enviado por RF una trama tipo 0x17 (COMMAND_TX) hacia un nodo remoto. En caso de recibir esta trama tras el envío de un comando "IS" (Inquiry/Force Sample), los datos contenidos en el campo "commandData" se corresponden con los que se recibirían en una trama 0x92 (SAMPL↩ ES_RX). Consultar la documentación de la trama SAMPLES_RX para más información.

Definición en la línea 283 del archivo FrameXBee.h.

4.12.2. Documentación de los datos miembro

4.12.2.1. uint8_t rxPacket97::commandAT[2]

Comando AT enviado previamente en trama 0x17.

Definición en la línea 291 del archivo FrameXBee.h.

4.12.2.2. uint8_t rxPacket97::commandData[MAX_COMMAND_LENGTH]

Datos/parámetro del comando AT.

Definición en la línea 293 del archivo FrameXBee.h.

4.12.2.3. uint8_t rxPacket97::commandStatus

Información sobre el comando AT.

Definición en la línea 292 del archivo FrameXBee.h.

4.12.2.4. uint8_t rxPacket97::frameID

Identificador de trama (para comprobar ACK)

Definición en la línea 288 del archivo FrameXBee.h.

4.12.2.5. uint8_t rxPacket97::frameType

Tipo de trama (en este caso: siempre 0x97)

Definición en la línea 287 del archivo FrameXBee.h.

4.12.2.6. uint8_t rxPacket97::length[2]

Longitud trama (excluyendo: start+length+checksum)

Definición en la línea 286 del archivo FrameXBee.h.

4.12.2.7. uint8_t rxPacket97::reserved[2]

Campo reservado (siempre 0xFFFE)

Definición en la línea 290 del archivo FrameXBee.h.

4.12.2.8. uint8_t rxPacket97::sourceMAC[8]

Dirección MAC (64b): nodo origen de la trama.

Definición en la línea 289 del archivo FrameXBee.h.

4.12.2.9. uint8_t rxPacket97::start

Byte de Inicio de Trama (siempre 0x7E)

Definición en la línea 285 del archivo FrameXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- FrameXBee.h

4.13. Referencia de la Estructura serial_port

Estructura: Contiene los elementos necesarios al configurar un puerto serie.

```
#include <SerialXBee.h>
```

Atributos públicos

- uint32_t baudrate
  Baudrate.
- char device [40]
  Nombre del puerto. Ej.: "/dev/ttySx".
- int fd
  File Descriptor (descriptor de fichero)

4.13.1. Descripción detallada

Estructura: Contiene los elementos necesarios al configurar un puerto serie.

Debe contener, como mínimo, el "descriptor de fichero" del puerto abierto. Por comodidad, se proporcionan también "baudrate" (velocidad in/out de la interfaz serie), y el nombre "device" que recibe la interfaz en el sistema. Otros miembros pueden ser agregados dependiendo de la plataforma/sistema sobre el que se ejecute el código.

Definición en la línea 60 del archivo SerialXBee.h.

4.13.2. Documentación de los datos miembro

4.13.2.1. uint32_t serial_port::baudrate

Baudrate.

Definición en la línea 62 del archivo SerialXBee.h.

4.13.2.2. char serial_port::device[40]

Nombre del puerto. Ej.: "/dev/ttySx".

Definición en la línea 63 del archivo SerialXBee.h.

4.13.2.3. int serial_port::fd

File Descriptor (descriptor de fichero)

Definición en la línea 61 del archivo SerialXBee.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- SerialXBee.h

Capítulo 5

Documentación de archivos

5.1. Referencia del Archivo README.md

5.2. Referencia del Archivo FrameXBee.cpp

Funciones, clases y variables para manejar tramas XBee-Pro 868.

```
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <iostream>
#include <fstream>
#include "FrameXBee.h"
#include "UtilsXBee.h"
```

Variables

- FrameXBee frame = FrameXBee()

5.2.1. Descripción detallada

Funciones, clases y variables para manejar tramas XBee-Pro 868.

Definiciones y/o declaraciones de clases, variables y funciones para manejar y/o configurar las diferentes tramas contempladas en el protocolo de comunicación de los módulos XBee-Pro 868MHz. XBee es "big-endian" en las tramas con datos multi-byte, y esta librería sigue el mismo criterio.

Autor

Antonio Navea Jiménez

Versión

1.1

Fecha

Septiembre 2016

Atención

Los módulos XBee-Pro 868 han sido descatalogados en 2016 por Digi International.

Copyright

5.2.2. Documentación de las variables

5.2.2.1. FrameXBee frame = FrameXBee()

Definición en la línea 1104 del archivo FrameXBee.cpp.

5.3. Referencia del Archivo FrameXBee.h

Módulos, clases y variables para manejar tramas XBee-Pro 868.

```
#include <cstdint>
```

Clases

- class FrameXBee

  Clase: Métodos y variables para manejar y/o configurar las diferentes tramas contempladas en el protocolo de comunicación de los módulos XBee-Pro 868MHz.
- struct packetXBee

  Estructura: Contiene los campos para almacenar la información y datos necesarios para enviar un paquete/trama a un módulo XBee. Las variables de la estructura hacen referencia tanto a campos propios de las tramas XBee para transmisión de datos y comandos AT como a variables auxiliares de utilidad para ciertas funciones de esta librería. Esta estructura es usada por gran parte de las funciones de la clase 'FrameXBee'.
- union received

  Unión: Facilita la extracción/interpretación de los datos recibidos/leídos.
- struct rxPacket88

  Estructuras 'rxPacket' - Define estructuras/uniones para almacenar, de forma ordenada, todos los bytes de cualquier trama XBee-Pro 868 que sea leída en recepción por la función 'FrameXBee::readXBee()'. Información adicional en el Manual de Usuario del Protocolo de los módulos XBee-Pro 868. Nota: En las tramas dónde el campo "checksum" viene precedido de un campo de longitud variable (payload, commandData, samplesData), se ha omitido el campo "checksum", porque su posición dentro de la trama dependerá de la longitud de dichos datos.
- struct rxPacket8A

  Estructura: Contiene los campos de la trama Rx tipo 0x8A (MODEM_STATUS).
- struct rxPacket8B

  Estructura: Contiene los campos de la trama Rx tipo 0x8B (TX_STATUS).
- struct rxPacket90

  Estructura: Contiene los campos de la trama Rx tipo 0x90 (NORMAL_RX).
- struct rxPacket91

  Estructura: Contiene los campos de la trama Rx tipo 0x91 (EXPLICIT_RX).
- struct rxPacket92

  Estructura: Contiene los campos de la trama Rx tipo 0x92 (SAMPLES_RX).
- struct rxPacket97

  Estructura: Contiene los campos de la trama Rx tipo 0x97 (COMMAND_RX).

Variables

- const uint8_t API_WITH_ESCAPES = 0x02

  API-mode = 2 (con caracteres de escape)
- const uint8_t API_WITHOUT_ESCAPES = 0x01

  API-mode = 1 (sin caracteres de escape)
- const uint8_t COMMAND_RX = 0x97

  Trama Rx "Remote AT Command Response".
- const uint8_t COMMAND_TX = 0x17

  Trama Tx "Remote AT Command Request".
- const uint8_t EXPLICIT_RX = 0x91

  Trama Rx "Explicit RX Indicator"(AO=1)
- const uint8_t EXPLICIT_TX =0x11

  Trama Tx "Explicit Addressing Command Frame".
- FrameXBee frame
- const uint8_t LOCAL_COMMAND_RX = 0x88

  Trama Rx "AT Command Response".
- const uint8_t LOCAL_COMMAND_TX = 0x08

  Trama Tx "AT Command Request".
- const uint8_t MAX_COMMAND_LENGTH = 30

  Longitud máxima (bytes) del campo "commandParameter" (trama "packetXBee").

- const uint16_t MAX_FRAME_LENGTH = (MAX_PAYLOAD_LENGTH + 24)

  Longitud máxima total (bytes) de una trama "struct packetXBee".
- const uint16_t MAX_PAYLOAD_LENGTH = 90

  Longitud máxima (bytes) del campo "payload" (trama "packetXBee").
- const uint8_t MAX_SAMPLES_LENGTH = 14

  Longitud máxima (bytes) del campo "samplesData" (trama 0x92 = SAMPLES_RX).
- const uint8_t MODEM_STATUS = 0x8A

  Trama Rx "Modem Status".
- const uint8_t NORMAL_RX = 0x90

  Trama Rx "Receive packet" (AO=0)
- const uint8_t NORMAL_TX = 0x10

  Trama Tx "Transmit Request".
- const uint8_t SAMPLES_RX = 0x92

  Trama Rx "Data Sample Rx Indicator".
- const uint8_t TX_RETRIES = 3

  Máximo número de intentos para transmitir una trama.
- const uint8_t TX_STATUS = 0x8B

  Trama Rx "Transmit Status".

5.3.1. Descripción detallada

Módulos, clases y variables para manejar tramas XBee-Pro 868.

Definiciones y/o declaraciones de clases, variables y funciones para manejar y/o configurar las diferentes tramas contempladas en el protocolo de comunicación de los módulos XBee-Pro 868MHz. XBee es "big-endian" en las tramas con datos multi-byte, y esta librería sigue el mismo criterio.

Autor

Antonio Navea Jiménez

Versión

1.1

Fecha

Septiembre 2016

Atención

Los módulos XBee-Pro 868 han sido descatalogados en 2016 por Digi International.

Copyright

5.3.2. Documentación de las variables

5.3.2.1. const uint8_t API_WITH_ESCAPES = 0x02

API-mode = 2 (con caracteres de escape)

Definición en la línea 115 del archivo FrameXBee.h.

5.3.2.2. const uint8_t API_WITHOUT_ESCAPES = 0x01

API-mode = 1 (sin caracteres de escape)

Definición en la línea 112 del archivo FrameXBee.h.

5.3.2.3. const uint8_t COMMAND_RX = 0x97

Trama Rx "Remote AT Command Response".

Definición en la línea 56 del archivo FrameXBee.h.

5.3.2.4. const uint8_t COMMAND_TX = 0x17

Trama Tx "Remote AT Command Request".

Definición en la línea 50 del archivo FrameXBee.h.

5.3.2.5. const uint8_t EXPLICIT_RX = 0x91

Trama Rx "Explicit RX Indicator"(AO=1)

Definición en la línea 54 del archivo FrameXBee.h.

5.3.2.6. const uint8_t EXPLICIT_TX =0x11

Trama Tx "Explicit Addressing Command Frame".

Definición en la línea 49 del archivo FrameXBee.h.

5.3.2.7. FrameXBee frame

Definición en la línea 1104 del archivo FrameXBee.cpp.

5.3.2.8. const uint8_t LOCAL_COMMAND_RX = 0x88

Trama Rx "AT Command Response".

Definición en la línea 66 del archivo FrameXBee.h.

5.3.2.9. const uint8_t LOCAL_COMMAND_TX = 0x08

Trama Tx "AT Command Request".

Definición en la línea 65 del archivo FrameXBee.h.

5.3.2.10. const uint8_t MAX_COMMAND_LENGTH = 30

Longitud máxima (bytes) del campo "commandParameter" (trama "packetXBee").

No debe exceder el valor: 256 (0x100).

Definición en la línea 79 del archivo FrameXBee.h.

5.3.2.11. const uint16_t MAX_FRAME_LENGTH = (MAX_PAYLOAD_LENGTH + 24)

Longitud máxima total (bytes) de una trama "struct packetXBee".

No debe exceder el valor: 300 (0x12C). Valor mínimo = MAX_PAYLOAD_LENGTH + 24 (bytes).

Definición en la línea 100 del archivo FrameXBee.h.

5.3.2.12. const uint16_t MAX_PAYLOAD_LENGTH = 90

Longitud máxima (bytes) del campo "payload" (trama "packetXBee").

No debe exceder el valor: 256 (0x100).

Definición en la línea 84 del archivo FrameXBee.h.

5.3.2.13. const uint8_t MAX_SAMPLES_LENGTH = 14

Longitud máxima (bytes) del campo "samplesData" (trama 0x92 = SAMPLES_RX).

Su longitud viene determinada por el número total de muestras que contiene la trama, que nunca será superior a 14 bytes (2-bytes de muestras digitales + 6 muestras ADC de 2-bytes cada una) para módulos XBee-PRO 868.

Definición en la línea 93 del archivo FrameXBee.h.

5.3.2.14. const uint8_t MODEM_STATUS = 0x8A

Trama Rx "Modem Status".

Definición en la línea 60 del archivo FrameXBee.h.

5.3.2.15. const uint8_t NORMAL_RX = 0x90

Trama Rx "Receive packet" (AO=0)

Definición en la línea 53 del archivo FrameXBee.h.

5.3.2.16. const uint8_t NORMAL_TX = 0x10

Trama Tx "Transmit Request".

Definición en la línea 48 del archivo FrameXBee.h.

5.3.2.17. const uint8_t SAMPLES_RX = 0x92

Trama Rx "Data Sample Rx Indicator".

Definición en la línea 55 del archivo FrameXBee.h.

5.3.2.18. const uint8_t TX_RETRIES = 3

Máximo número de intentos para transmitir una trama.

Definición en la línea 122 del archivo FrameXBee.h.

5.3.2.19. const uint8_t TX_STATUS = 0x8B

Trama Rx "Transmit Status".

Definición en la línea 61 del archivo FrameXBee.h.

5.4. Referencia del Archivo SerialXBee.cpp

Funciones para controlar/configurar conexiones con un módulo XBee a través de una interfaz serie estándar.

```
#include <cerrno>
#include <cstring>
#include <fcntl.h>
#include <unistd.h>
#include "SerialXBee.h"
#include "UtilsXBee.h"
```

Variables

- SerialXBee serial = SerialXBee()

5.4.1. Descripción detallada

Funciones para controlar/configurar conexiones con un módulo XBee a través de una interfaz serie estándar.

Esta librería permite abrir/cerrar un puerto serie, configurar el baudrate (valores estándar soportados por los módulos, desde 1200 a 230400 bps) de la conexión serie, así como establece correctamente las opciones por defecto para el puerto: 8-bits, sin paridad, 1-bit de stop).

Autor

Antonio Navea Jiménez

Versión

1.1

Fecha

Septiembre 2016

Copyright

5.4.2. Documentación de las variables

5.4.2.1. SerialXBee serial = SerialXBee()

Definición en la línea 236 del archivo SerialXBee.cpp.

5.5. Referencia del Archivo SerialXBee.h

Métodos para controlar/configurar conexiones con un módulo XBee a través de una interfaz serie estándar.

```
#include <cstdint>
#include <termios.h>
```

Clases

- struct serial_port

  Estructura: Contiene los elementos necesarios al configurar un puerto serie.
- class SerialXBee

  Clase: Métodos para controlar/configurar conexiones con un módulo XBee a través de una interfaz serie estándar.

Variables

- SerialXBee serial

5.5.1. Descripción detallada

Métodos para controlar/configurar conexiones con un módulo XBee a través de una interfaz serie estándar.

Esta librería permite abrir/cerrar un puerto serie, configurar el baudrate (valores estándar soportados por los módulos, desde 1200 a 230400 bps) de la conexión serie, así como establecer correctamente las opciones por defecto para el puerto: 8-bits, sin paridad, 1-bit de stop.

Autor

Antonio Navea Jiménez

Versión

1.1

Fecha

Septiembre 2016

Copyright

5.5.2. Documentación de las variables

5.5.2.1. SerialXBee serial

Definición en la línea 236 del archivo SerialXBee.cpp.

5.6. Referencia del Archivo UtilsXBee.cpp

Utilidades auxiliares para tramas XBee, para realizar tareas concretas de forma especializada y optimizada.

```
#include <cstdint>
#include <cstdio>
#include <cstring>
#include <sys/time.h>
#include <cstdarg>
#include "UtilsXBee.h"
```

Variables

- UtilsXBee utils = UtilsXBee()

5.6.1. Descripción detallada

Utilidades auxiliares para tramas XBee, para realizar tareas concretas de forma especializada y optimizada.

Esta librería está pensada para usarse de forma completamente independiente al tipo de trama XBee utilizada.

Autor

Antonio Navea Jiménez

Versión

1.1

Fecha

Septiembre 2016

Copyright

{ Copyright (C) 2016, Antonio R. Navea Jimenez. All rights reserved.

This file is part of UniversalXBee.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http↩://www.gnu.org/licenses/. }

5.6.2. Documentación de las variables

5.6.2.1. UtilsXBee utils = UtilsXBee()

Definición en la línea 473 del archivo UtilsXBee.cpp.

5.7. Referencia del Archivo UtilsXBee.h

Utilidades auxiliares para tramas XBee, para realizar tareas concretas de forma especializada y optimizada.

```
#include <cstdint>
#include <sys/time.h>
#include <iomanip>
#include <iostream>
```

Clases

- class UtilsXBee

  Clase: Métodos auxiliares para tramas XBee, para realizar tareas concretas de forma especializada y optimizada.

'defines'

- #define BASIC_INFO()
- #define VERBOSE_ENABLE

Variables

- const uint8_t FAILURE = -1

  Ejecución de una función con "Fracaso" o "Error".
- const uint8_t SUCCESS = 0

  Ejecución de una función con "Éxito".
- UtilsXBee utils

5.7.1. Descripción detallada

Utilidades auxiliares para tramas XBee, para realizar tareas concretas de forma especializada y optimizada.

Esta librería está pensada para usarse de forma completamente independiente al tipo de trama XBee utilizada.

Autor

Antonio Navea Jiménez

Versión

1.1

Fecha

Septiembre 2016

Copyright

5.7.2. Documentación de los 'defines'

5.7.2.1. #define BASIC_INFO( )

Valor:

```
if (utils.basicEnabled)                                  \
    do {                                                                \
        std::cout « "\nFile: " « __FILE__ « ", Line: " «  __LINE__      \
        « ", Function: " « std::dec « __FUNCTION__ « "()" « std::endl; \
    } while (false)
```

Macro: Imprime información básica (nombre del Archivo, número de Línea, y nombre de Función) del ámbito en el código donde se ejecuta la macro.

Esta macro informa del ámbito del código dónde se han generado los mensajes de información/error mostrados al definir las macros "SHOW_CUSTOM_INFO" o "VERBOSE_ENABLE". Es posible desactivar la impresión de esta información si "basicEnabled = 0" (ver función: setDebugVar()).

Definición en la línea 80 del archivo UtilsXBee.h.

5.7.2.2. #define VERBOSE_ENABLE

Activa la impresión de información por std::cout mostrando mensajes de información ("INFO:") sobre ciertas acciones realizadas internamente por una función, o informando de los errores ("ERROR:") más comunes que se pueden producir al ejecutar una función, ayudando al usuario a corregir la acción que dio lugar al error.

Definición en la línea 49 del archivo UtilsXBee.h.

5.7.3. Documentación de las variables

5.7.3.1. const uint8_t FAILURE = -1

Ejecución de una función con "Fracaso" o "Error".

Definición en la línea 92 del archivo UtilsXBee.h.

5.7.3.2. const uint8_t SUCCESS = 0

Ejecución de una función con "Éxito".

Definición en la línea 91 del archivo UtilsXBee.h.

5.7.3.3. UtilsXBee utils

Definición en la línea 473 del archivo UtilsXBee.cpp.

5.8. Referencia del Archivo COPYING.txt

Funciones

- GNU GENERAL PUBLIC LICENSE June Copyright (C) 2007 Free Software Foundation

Variables

- GNU GENERAL PUBLIC LICENSE Version

5.8.1. Documentación de las funciones

5.8.1.1. GNU GENERAL PUBLIC LICENSE June Copyright (C)

5.8.2. Documentación de las variables

5.8.2.1. GNU GENERAL PUBLIC LICENSE Version

Definición en la línea 2 del archivo COPYING.txt.

Índice alfabético

6 Conclusiones

En este proyecto se ha visto la versatilidad de los módulos *XBee®* para realizar comunicaciones inalámbricas por radio-frecuencia, así como se ha explotado su capacidad de comunicación a través de la UART con un dispositivo "host", estableciendo una comunicación serie entre nodo y host. El protocolo que estos módulos soportan tiene como principal virtud la sencillez de aprendizaje, y la facilidad para trabajar con los módulos en casos prácticos y reales en muy poco tiempo. Todo ello ha derivado en la realización de una librería programada en C/C++ que permite usar los módulos *XBee-PRO® 868* en plataformas con soporte para sistemas operativos POSIX (Windows/Cygwin y Linux con GCC), tales como PC's o dispositivos Raspberry Pi, prescindiendo de este modo de la dependencia con otras plataformas como *Waspmote*. Además, se ha mostrado como documentar, de forma ordenada y atractiva visualmente, el código de un proyecto software, mediante el uso de la herramienta *Open-Source* denominada *Doxygen*.

Por otra parte, como todo proyecto de desarrollo software, esta librería es susceptible de ser ampliada y/o mejorada en el futuro, añadiendo nuevas posibilidades, como pueden ser la encriptación y/o cifrado de los datos enviados, la posibilidad de poder configurar la interfaz serie más allá de los valores por defecto de uso general, o incorporando nuevas funcionalidades, como podrían ser el envío y recepción de multi-tramas, o la división de tramas que superen el tamaño máximo permitido en los buffer de Rx/Tx mediante una división en tramas menores que se recompongan en el nodo destino, o una adaptación dinámica de la potencia de transmisión del módulo en función del ratio de tramas enviadas correctamente y/o la calidad de la señal recibida desde los nodos adyacentes, minimizando así el gasto de energía.

Toda ampliación o mejora de características de la librería, evidentemente, estará supeditada a la capacidad de computación y autonomía de la batería del nodo sobre el que se ejecute el código (en caso de ser aplicable). Asimismo, esta librería es fácilmente adaptable a otros módulos *XBee®*, ya que todos ellos comparten un set básico de tramas API, independiente del protocolo sobre el que trabaja cada módulo individualmente (ZigBee, 802.15.4, etc), lo que la convierte en una potente herramienta de trabajo para aprovechar las características de los módulos *XBee®* de Digi International.

Parte III.

Apéndices

Apéndice A

Compiladores GNU: GCC y G++

Toda la información de este Apéndice hace referencia a la versión (en fase de desarrollo) del compilador GCC v7.0 [15], siendo la *release* publicada más reciente la versión GCC v6.2 (agosto 2016).

GCC significa "GNU Compiler Collection", y es una distribución integrada de compiladores con soporte para varios de los principales lenguajes de programación, que actualmente incluyen: C, C++, Objective-C, Objective-C++, Java, Fortran, Ada, y Go.

Para cada lenguaje compilado por GCC para el que existe un estándar, GCC intenta seguir una o más versiones de esa norma, en unas ocasiones con ciertas excepciones, y en otras permitiendo algunas extensiones.

Existe un componente de GCC, independiente del lenguaje de programación usado, que incluye la mayoría de los optimizadores, así como los *back-ends* que generan el código máquina para gran multitud de procesadores.

La parte del compilador que es específica de un lenguaje de programación en particular se llama *front-end*. Además de los *front-ends* que son componentes integrados de GCC, hay otros *front-ends* que son mantenidos por separado como lenguajes de apoyo (como Pascal, Mercury, y COBOL).

La mayoría de compiladores para lenguajes diferentes a C tienen sus propios nombres. Por ejemplo, el compilador de C++ se denomina G++, y el compilador de Ada es el GNAT. Cuando se habla de la compilación en cualquiera de dichos lenguajes, se puede hacer referencia al compilador en concreto por su propio nombre (G++, GNAT, etc), o de forma genérica como GCC.

Históricamente, los compiladores de muchos lenguajes, incluyendo C++ y Fortran, se han implementado como "pre-procesadores" que generan otro lenguaje de alto nivel como C. Ninguno de los compiladores incluidos en GCC están implementados de este modo; todos ellos generan código máquina directamente. Este tipo de pre-procesador no debe confundirse con el *pre-procesador* de C, que es una característica inherente de los lenguajes de programación C, C++, Objective-C y Objective-C++.

A.1 Lenguaje C

El estándar original **ANSI-C**[1] (X3.159-1989) fue ratificado en 1989, y se hizo publico en 1990, fecha en la que este estándar fue ratificado como estándar ISO (ISO/IEC 9899:1990)[2].

No hubo diferencias técnicas entre estas publicaciones, aunque las secciones del estándar ANSI se volvieron a numerar y se convirtieron en cláusulas de la norma ISO. Esta norma, en sus dos formas, es comúnmente conocida como **C89**, o en ocasiones como **C90** (en referencia a las fechas de ratificación). Para seleccionar esta norma en GCC, se ha de utilizar una de las siguientes opciones (todas ellas son equivalentes entre sí): *-ansi*, *-std=c90* o *-std=iso9899:1990*. Para obtener todos los diagnósticos requeridos por el estándar, se debe especificar además la opción *-pedantic*, o si se desea que sean considerados como errores en lugar de advertencias (*warnings*), la opción *-pedantic-errors*.

Los errores encontrados en la norma ISO 1990 C se solventaron en sendas "Correcciones Técnicas" (fe de erratas) publicadas en 1994 y 1996. GCC no es compatible con la versión sin corregir.

Una nueva versión de la norma ISO C se publicó en 1999 como ISO/IEC 9899:1999, y se conoce comúnmente como **C99** (durante el desarrollo de este estándar, los borradores se conocían como *C9X*). GCC da soporte sustancialmente completo a esta versión del estándar. Para seleccionar esta norma, se debe utilizar las opciones de compilación *-std=c99* o *-std=iso9899:1999*.

Los errores encontrados en la norma ISO 1999 C se solventaron en tres "Correcciones Técnicas", publicadas en 2001, 2004 y 2007. GCC no es compatible con la versión sin corregir.

Una cuarta versión del estándar C, conocido como **C11**, se publicó en 2011 como ISO/IEC 9899:2011 (durante el desarrollo de este estándar, los borradores se conocían como *C1X*). GCC da soporte sustancialmente completo a esta versión del estándar, el cual se habilita con *-std=c11* o *-std=iso9899:2011*.

De forma predeterminada, GCC proporciona algunas extensiones al lenguaje C que sólo en raras ocasiones entran en conflicto con el estándar C. Algunas de las características que forman parte del estándar C99 se aceptan como extensiones en el modo C90, y algunas características que son parte de la norma C11 se aceptan como extensiones en los modos C90 y C99. El uso de cualquiera de las opciones *-std* (denominadas *"dialectos"*) mencionadas anteriormente desactiva dichas extensiones cuando entran en conflicto con la versión del estándar C seleccionado. También se puede seleccionar una versión extendida del lenguaje C de forma explícita con *-std=gnu90* (para C90 con extensiones GNU), *-std=gnu99* (para C99 con extensiones GNU) o *-std=gnu11* (para C11 con extensiones GNU).

El **valor por defecto**, cuando ningún dialecto de C se ha seleccionado con la opción *-std*, es: **-std=gnu11**.

En la Tabla A.1 se hace un resumen de los diferentes dialectos y opciones que se han mencionado en esta sección:

[1] El "Instituto Nacional Estadounidense de Estándares", conocido por sus siglas en inglés como *American National Standards Institute* (ANSI), es una organización sin fines de lucro que supervisa el desarrollo de estándares en EE.UU., miembro de las organización internacionales ISO e IEC. Más información en *https://www.ansi.org/*

[2] *International Organization for Standardization* (ISO) u "Organización Internacional de Normalización", es una organización compuesta por diversas organizaciones nacionales de estandarización, con sede en Ginebra (Suiza), que se encarga de la creación de estándares internacionales; el representante español es la *Asociación Española de Normalización y Certificación* (AENOR). ISO ha formado varios comités conjuntos con la "Comisión Electrotécnica Internacional" (*International Electrotechnical Commission* (IEC) por sus siglas en inglés), para desarrollar los estándares y la terminología relacionados con áreas de tecnología eléctrica y electrónica, incluyendo el comité JTC 1 para las "Tecnologías de la Información". Más información en *http://www.iso.org/* y *http://www.aenor.es/*

Tabla A.1 Dialectos de GCC v7.0 y su correspondencia con los estándares del lenguaje C.

Estándar C	Dialecto
C89 o C90	-ansi -std=c90 -std=iso9899:1990
C99	-std=c99 -std=iso9899:1999
C11	-std=c11 -std=iso9899:2011
C90 con extensiones GNU	-std=gnu90
C99 con extensiones GNU	-std=gnu99
C11 con extensiones GNU	**-std=gnu11** (valor por defecto)

A.2 Lenguaje C++

GCC es compatible tanto con el estándar original **ISO C++**, publicado en 1998, como con las revisiones publicadas en 2011 y 2014. Es posible utilizar **G++** como nombre del compilador GCC cuando se hace referencia al lenguaje C++.

El estándar original ISO C++ fue publicado como estándar ISO (ISO/IEC 14882:1998) y modificado en una "Corrección Técnica" publicada en 2003 (ISO/IEC 14882:2003). Estos estándares son conocidos como **C++98** y **C++03** respectivamente. GCC implementa la mayor parte del estándar C++98 (*export* es una notable excepción) y la mayoría de los cambios de C++03. Para seleccionar esta norma en GCC, se debe utilizar una de las siguientes opciones: *-ansi*, *-std=c++98*, o *-std=c++03*.

Un estándar ISO C++ revisado fue publicado en 2011 como ISO/IEC 14882:2011, y se conoce como **C++11** (durante el desarrollo de este estándar, los borradores se conocían como C++0X). C++11 implementa varios cambios en el lenguaje C++, los cuales han sido todos incluidos en GCC. Para seleccionar este estándar en GCC, se debe utilizar la opción *-std=c++11*.

Otro estándar revisado de ISO C++ fue publicado en 2014 como ISO/IEC 14882:2014, y se conoce como **C++14** (antes de su publicación se conocía como C++1y). C++14 contiene numerosos e importantes cambios en el lenguaje C++, todos incluidos en GCC. Para seleccionar esta norma en GCC, se debe utilizar la opción *-std=c++14*.

GCC también admite el *"C++ Concepts Technical Specification"*, ISO/IEC TS 19217:2015, que permite que se definan restricciones para las plantillas o *templates*, permitiendo que los argumentos de la plantilla sean comprobados, sobrecargados o especializados según las restricciones que se apliquen. El soporte para *"C++ Concepts Technical Specification"* se incluye en el modo experimental **C++1z**, que corresponde a la próxima revisión de la norma ISO C++, cuya fecha estimada de publicación es 2017. Para habilitar C++1z en GCC, se debe utilizar la opción *-std=c++17* o *-std=c++1z*.

Para obtener todos los diagnósticos requeridos por cualquiera de las versiones del estándar ISO C++ descritos anteriormente, se debe especificar la opción *-pedantic*, o *-pedantic-errors* si se desea que sean considerados como errores en lugar de *warnings*, de lo contrario GCC admitirá algunas de las características que no cumplen el estándar ISO C++, tratándolas como extensiones del lenguaje.

Por defecto, el GCC también proporciona algunas extensiones adicionales para el lenguaje C++, que en ocasiones poco frecuentes entran en conflicto con el estándar C++. El uso de cualquiera de las opciones *-std* (denominadas *"dialectos"*) mencionadas anteriormente desactiva dichas extensiones cuando entran en

conflicto con la versión del estándar C++ seleccionado.

También se puede seleccionar una versión extendida del lenguaje C++ de forma explícita con *-std=gnu++98* (para C++98 con extensiones GNU), *-std=gnu++11* (para C++11 con extensiones GNU) o *-std=gnu++14* (para C++14 con extensiones GNU), o *-std=gnu++1z* (para C++1z con extensiones de GNU).

El **valor por defecto**, si ningún dialecto de C++ se ha seleccionado con la opción *-std*, es: **-std=gnu++14**.

En la Tabla A.2 se hace un resumen de los diferentes dialectos y opciones que se han mencionado en esta sección:

Tabla A.2 Dialectos de GCC v7.0 y su correspondencia con los estándares del lenguaje C++.

Estándar C++	Dialecto
C++98 o C++03	-ansi -std=c++98 -std=c++03
C++11	-std=c++11
C++14	-std=c++14
C++17 o C++1z (modo experimental)	-std=c++17 -std=c++1z
C++98 con extensiones GNU	-std=gnu++98
C++11 con extensiones GNU	-std=gnu++11
C++14 con extensiones GNU	**-std=gnu++14** (valor por defecto)
C++17 o C++1z con extensiones GNU (modo experimental)	-std=gnu++1z

Apéndice B

Doxygen - Generación de documentación para C++

Doxygen [10] es la herramienta *de-facto* para generar documentación a partir de código fuente C++ comentado, que soporta además muchos otros de los lenguajes de programación más conocidos y utilizados, como son C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, y UNO/OpenOffice), Fortran, VHDL y Tcl. La mayor parte del código de *Doxygen* está escrita por *Dimitri van Heesch*, bajo la licencia *GNU General Public License*.

Doxygen es capaz de generar documentación HTML para su visualización on-line en navegadores web, así como genera también documentación para su consulta *off-line* en forma de "Manual de Referencia" en LaTeX (a partir del cuál es posible generar un documento en formato PDF), siempre a partir de un conjunto de archivos fuente que contienen comentarios formateados para que Doxygen los interprete. Existe también soporte para generar documentación en RTF (MS-Word), PostScript, PDF con hipervínculos, HTML comprimido, y "*man pages*" (páginas de manual en línea para sistemas Unix).

La documentación es directamente extraída de los archivos fuente que han sido previamente comentados, lo que hace mucho más sencillo mantener la coherencia entre la documentación y el código fuente. Además, se puede configurar *Doxygen* para extraer la estructura del código a partir de código fuente sin comentar, lo que resulta de gran utilidad para inferir la estructura del mismo en grandes proyectos. *Doxygen* puede también mostrar las relaciones entre diferentes elementos incluyendo gráficos de dependencias y diagramas de herencias entre clases e interfaces que son generados automáticamente. En todos los casos, se debe tener muy en cuenta que se debe respetar un ancho máximo de 80 caracteres para los comentarios, especialmente en aquellos de una sola línea.

Doxygen ha sido desarrollado bajo Mac OS X y Linux, siendo altamente portable. Por otra parte, hay disponibles diferentes ejecutables para *Windows*, así como ciertos plugins que integran las funciones de *Doxygen* en algunos entornos de desarrollo, como el *IDE Eclipse*.

Para indicar a *Doxygen* que debe procesar un comentario como parte de la documentación, se deben usar ciertas palabras clave o comentarios con un formato concreto, ya que *Doxygen* "no reconoce" como parte de la documentación que debe generar los comentarios "ordinarios" presentes en el código, típicos en los lenguajes afines a C/C++, tales como // (comentario de una línea) o /* */ (comentario de varias líneas, o bloque de comentario). Los comentarios de estilo C se pueden utilizar para agregar anotaciones al código que no se desean que se visualicen en la documentación.

Al comentar el código se pueden utilizar descripciones breves o detalladas. Hay diferentes formas de indicar a *Doxygen* que un comentario es una descripción detallada:

- Se puede usar el estilo *JavaDoc*, que consiste en utilizar un bloque de comentarios de estilo C, pero empezando con dos asteriscos (*), como en el siguiente ejemplo:

```
/** Doxygen interpretará este texto como una
 *  descripción detallada.
 */
```

- Se puede usar el estilo *Qt*, agregando un signo de exclamación (!) tras la llave de apertura de un bloque de comentarios de estilo C, como en el siguiente ejemplo:

```
/*! Doxygen interpretará este texto como una
 *  descripción detallada.
 */
```

- En los dos casos anteriores, los asteriscos intermedios son opcionales, y el texto de la descripción puede comenzar en cualquier línea del bloque de comentarios. Ejemplo:

```
/*!
   Doxygen también interpretará este texto como una descripción
       detallada.
 */
```

- Una tercera alternativa es usar un bloque de comentarios formado por, al menos, dos líneas de comentarios de estilo C++, pero agregando a cada línea un barra adicional (/) o un signo de exclamación (!). A continuación, un ejemplo (en ambos casos, el final del bloque de comentarios se indica con una línea de comentario en blanco):

```
/// Doxygen también interpretará este texto
/// como una descripción detallada.
///

//! Esto también constituirá una descripción
//! detallada para Doxygen.
//!
```

Para las descripciones breves existen también varias posibilidades:

- Una opción es usar el comando `\brief` dentro de alguno de los bloques de comentario explicados anteriormente para descripciones detalladas. Este comando termina cuando finaliza un párrafo, por lo que la descripción detallada comienza tras una línea en blanco. Ejemplo:

```
/*! \brief Descripción breve.
 *         Sigue la descripción breve.
 *
 *  Tras línea en blanco, comienza esta descripción detallada.
 */
```

Todos los comandos admiten dos formas con equivalente función: con barra invertida \ (ejemplo: \brief) y con arroba @ (ejemplo: @brief).

- Si la opción JAVADOC_AUTOBRIEF está configurada en "YES" en el archivo de configuración (generalmente llamado *Doxyfile*), al utilizar el estilo *JavaDoc* en bloques de comentarios comenzará automáticamente una descripción breve, que terminará en el primer punto que aparezca, seguido de un espacio o de nueva línea, siendo el texto posterior una descripción detallada. A continuación, un ejemplo:

```
/** Descripción breve que acaba con este punto. La descripción
 *  detallada es todo este texto que sigue hasta el final de párrafo.
 */

 /// En comentarios multi-línea de estilo C++ también se cumple
 /// lo anterior. Esta frase ya es una descripción detallada.
```

- Una tercera opción es usar un comentario de estilo C++ que no ocupe más de una línea. Aquí un ejemplo:

```
/// Descripción breve.
/** Descripción detallada. */


... o también...


//! Descripción breve.

//! La descripción detallada
//! comienza aqui.
```

En el segundo ejemplo del anterior código, es indispensable separar la descripción breve del bloque que contiene la descripción detallada, usando una línea en blanco, para lo cual es necesario que se haya configurado JAVADOC_AUTOBRIEF = NO.

Como se ha podido observar, *Doxygen* es bastante flexible. Todas las descripciones explicadas anteriormente, deben **preceder** a la función, método, variable, etc. a la que describen. Para introducir una descripción **después** de los miembros (métodos, variables) de un archivo, struct, union, class o enum, se debe utilizar el símbolo (<), tal y como se muestra en los siguientes ejemplos:

```
int var1; /*!< Descripción detallada después del miembro */

int var2; /**< Descripción detallada después del miembro */

int var3; //!< Descripción detallada después del miembro
     //!<

int var4;  ///< Descripción detallada después del miembro
     ///<

int var5; //!< Descripción breve después del miembro

int var6; ///< Descripción breve después del miembro
```

En la librería *UniversalXBee* se ha utilizado siempre el estilo *Qt*, tanto para las descripciones breves como para las detalladas. A continuación, se describen y muestra un ejemplo de otros comandos especiales que se han usado en dicha librería.

```
   /*!
  * @brief
  * Descripción breve.
  *
  * \parblock
  * Este es el primer párrafo de una descripción detallada.
  *
  * Aunque haya una línea en blanco previa, al estar dentro de un bloque
  * "parblock", este párrafo también pertenece a la descripción detallada.
  * \endparblock
  *
  * @code
  *   fprintf(stdout, "Ejemplo de código dentro de la documentación\n");
  * @endcode
  *
  * @param[in,out] number1  Descripción del parámetro "number1".
  *
  * @param[in] number2  Descripción del parámetro "number2".
  *
  * @param[out] number3  Descripción del parámetro "number3".
  *
  * @retval  0 Descripción de este valor que devuelve la función.
  * @retval  -1 Descripción de este otro valor que devuelve la función.
  *
  * @return  Descripción de un valor devuelto por la función no conocido
  *       a priori.
  */
int8_t functionExample( uint8_t* number1, unit8_t number2, uint16_t number3 );
```

Para crear un bloque con varios párrafos que formen una misma unidad descriptiva, se deben usar los comandos `@parblock` (apertura del bloque) y `@endparblock` (cierre del bloque), como en el anterior ejemplo, dónde se han utilizado dichos comandos para formar una descripción detallada de dos párrafos. Dicho comando puede usarse para incorporar varios párrafos a otro tipo de descripciones (como por ejemplo, @param o @retval, que se explicarán más adelante).

`@code` (apertura de bloque) y `@endcode` (cierre de bloque) encierran un bloque de código, que es tratado de forma diferente al texto ordinario, ya que es interpretado como código fuente. Los nombres de clases y miembros, y otras entidades documentadas, son automáticamente reemplazadas por links a la documentación correspondiente.

`@param[(dirección)]` comienza la descripción del parámetro/argumento de la función con el nombre que acompaña (en los ejemplos, "numberx"), seguido de la descripción de dicho parámetro. La existencia del parámetro es chequeada, y si no existe el parámetro, o algún parámetro de la definición o declaración de la función no ha sido documentado, se avisará con un "warning". @param[(dirección)] tiene un atributo opcional (dirección), que especifica si el atributo es de entrada [in], salida [out], o ambos [in,out] (caso típico cuando un parámetro es pasado por referencia para coger su valor y actualizarlo posteriormente desde el interior de la propia función).

`@retval` describe el valor devuelto por una función, indicando dicho valor. En caso de no poder indicarse un valor en concreto conocido previamente (por ejemplo, si el valor devuelto es la suma de dos números pasados por argumentos), se puede utilizar el comando `@return` seguido de la descripción del valor o set de

valores.

Existen muchas otras opciones y comandos para completar la documentación, que se pueden encontrar en la sección "*Documentation*" de la web de descarga de *Doxygen* [10].

A continuación, se muestra una captura de pantalla (Figura B.1) de la versión HTML que *Doxygen* genera del código documentado de la librería *UniversalXBee*. Para acceder a ella, se debe abrir el archivo *index.html* que hay dentro de la carpeta /doc/html que acompaña a la librería.

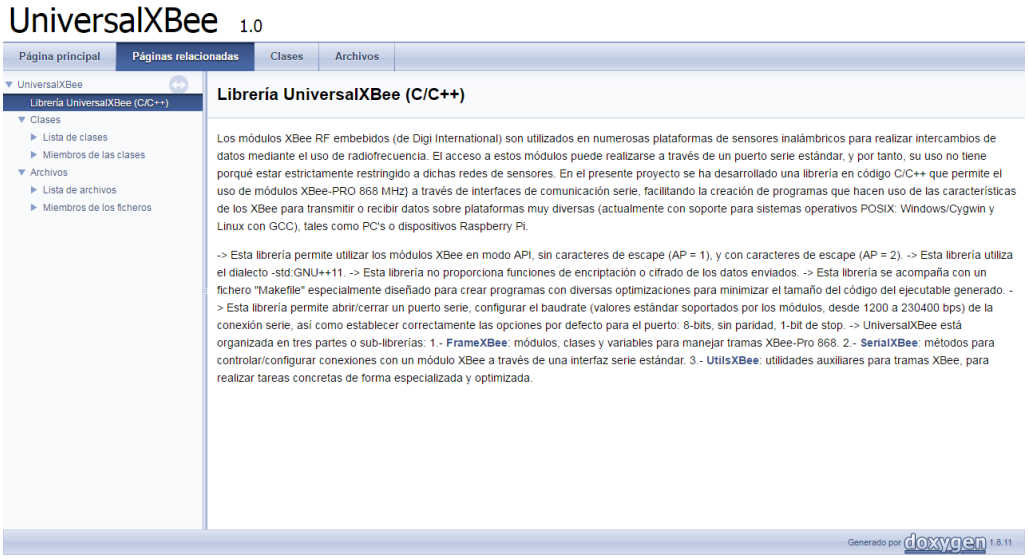


Figura B.1 Aspecto gráfico de la documentación en versión HTML de *Doxygen*.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] *Standars, Contributed, and others libraries for Arduino*, https://www.arduino.cc/en/Reference/Libraries/.

[2] *Arduino library for communicating with XBee radios in API mode*, https://github.com/andrewrapp/xbee-arduino/.

[3] *AVR Libc - Standard C library for Atmel AVR microcontrollers (avr-binutils, avr-gcc, avr-libc)*, http://www.nongnu.org/avr-libc/.

[4] Eduardo Cañete, Jaime Chen, Manuel Díaz, Luis Llopis, Ana Reyna, and Bartolomé Rubio, *Using Wireless Sensor Networks and Trains as Data Mules to Monitor Slab Track Infrastructures*, vol. 15, Physical Sensors, no. 7, ISSN: 1424-8220, Multidisciplinary Digital Publishing Institute, http://www.mdpi.com/1424-8220/15/7/15101/, January 2015 (English).

[5] *Tutorials, Reference, Information, Articles, Forum, and more resources to learn the C++ language*, http://www.cplusplus.com/.

[6] *ISO C++ Programming Language FAQs*, https://isocpp.org/faq.

[7] *Learning C++ (complete expert tutorials)*, http://www.learncpp.com.

[8] *C++ Programming Style Guidelines*, http://geosoft.no/development/cppstyle.html.

[9] *Portable ANSI-C code for communicating with Digi International's XBee wireless radio modules in API-mode*, https://github.com/digidotcom/xbee_ansic_library/.

[10] *Doxygen, de-facto standard tool for generating documentation from annotated C++ sources*, http://www.stack.nl/~dimitri/doxygen/.

[11] *Eclipse - Integrated Development Environment and open-source community for developers*, https://eclipse.org/mars/.

[12] *Git - Free and open-source distributed version control system for software development projects*, https://git-scm.com/.

[13] *GCC and Make: Compiling, Linking and Building C/C++ Applications*, https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html.

[14] *Free-Software GNU licenses*, https://www.gnu.org/licenses/.

[15] *Manual for GNU compilers: GCC for C and C++*, https://gcc.gnu.org/onlinedocs/gcc/index.html.

[16] *Manual for GNU compilers: GNU make*, http://www.gnu.org/software/make/manual/make.html.

[17] *ISO/IEC JTC 1 (Information Technology) - Internet of Things (IoT), Preliminary Report, 2014*, http://www.iso.org/iso/internet_of_things_report-jtc1.pdf/.

[18] Herb Sutter and Andrei Alexandrescu, *C++ Coding Standards - 101 Rules, Guidelines, and Best Practices*, ISBN: 0-321-11358-6, Pearson Education, Inc., 2005 (English).

[19] *Waspmote (Libelium) - Open Source Sensor Node for the Internet of Things*, http://www.libelium.com/products/waspmote/.

[20] *Waspmote Pro API Online Documentation*, https://www.libelium.com/api/waspmote/.

[21] *Waspmote SDK (Software Development Kit) - Waspmote Pro IDE and Waspmote Pro API*, http://www.libelium.com/development/waspmote/sdk_applications/.

[22] *XBee-PRO® 868 Product Manual*, http://ftp1.digi.com/support/documentation/90001020_F.pdf/.

[23] *Knowledge Base articles - Digi support*, http://knowledge.digi.com/.

[24] *Knowledge Base: Analog and Digital Sampling Using XBee® Radios*, http://knowledge.digi.com/articles/Knowledge_Base_Article/Digital-and-analog-sampling-using-XBee-radios/.

[25] *XBee® RF embedded modules*, http://www.digi.com/products/xbee-rf-solutions/modules/.

[26] *XCTU - Next Generation Configuration Platform for XBee/RF Solutions*, http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu/.

Índice alfabético

Glosario

AENOR *Asociación Española de Normalización y Certificación*. 132

ANSI *American National Standards Institute*. 132

CoM *Computer-on-Module*. 4

DIY *Do-It-Yourself*. 4

GPL *GNU General Public License*. 4

IEC *International Electrotechnical Commission*. 132

IoT *Internet-of-Things*. 3

ISM *Industrial, Scientific and Medical*. 6

ISO *International Organization for Standardization*. 132

ITU *International Telecommunication Union*. 4

LGPL *GNU Lesser General Public License*. 4

LOS *Line-Of-Sight*. 6

M2M *Machine-To-Machine*. 4

MIT *Massachusetts Institute of Technology*. 3

OSH *Open-Source Hardware*. 4

RTTI *Run-Time Type Information*. 34

SoC *System-on-Chip*. 4

SRD *Show Range Devices*. 6