Digi XBee® 3 DigiMesh 2.4

RF Module

User Guide

Revision history—90002277

Revision	Date	Description
E	April 2020	Added BLE commands, **R?**, **US**. Updated OTA firmware/file system upgrades.
F	May 2020	Revised all API frame descriptions.
G	August 2020	Added **Sleep immediate**. Added the **SI** command. Added a note to **D8**. Updated **OTA firmware/file system upgrades**.
H	July 2021	Added safety instructions and UKCA labeling requirements.
J	March 2022	Added translated safety instructions.

Trademarks and copyright

Disclaimers

Warranty

To view product warranty information, go to the following website:

www.digi.com/howtobuy/terms

Customer support

Gather support information: Before contacting Digi technical support for help, gather the following information:

Product name and model

Product serial number (s)

Firmware version

Operating system/browser (if applicable)

- Logs (from time of reported issue)
- Trace (if possible)
- Description of issue
- Steps to reproduce

Contact Digi technical support: Digi offers multiple technical support plans and service packages. Contact us at +1 952.912.3444 or visit us at www.digi.com/support.

Feedback

To provide feedback on this document, email your comments to

techcomm@digi.com

Include the document title and part number (Digi XBee® 3 DigiMesh 2.4 RF Module User Guide, 90002277 J) in the subject line of your email.

Contents

Get started

Get started with MicroPython

Secure access

File system

Configure the XBee 3 DigiMesh RF Module

Modes

Serial communication

SPI operation

I/O support

Networking

Network commissioning and diagnostics

Sleep support

AT commands

Operate in API mode

Frame descriptions

OTA firmware/file system upgrades

OTA file system upgrades

About the XBee 3 DigiMesh RF Module

The XBee 3 DigiMesh RF Module consists of DigiMesh 2.4 firmware loaded on the XBee 3 hardware. This user guide covers the firmware. For information about XBee 3 hardware, see the *XBee 3 RF Module Hardware Reference Manual*.

Digi XBee 3 devices offer the flexibility to switch between multiple frequencies and wireless protocols as needed. These devices use the DigiMesh networking protocol using a globally deployable 2.4 GHz transceiver. This peer-to-peer mesh network offers users added network stability through self-healing, dense network operation, extending the operational life of battery dependent networks and provides an upgrade path to IEEE 802.15.4 or ZigBee mesh protocols, if desired.

The XBee 3 DigiMesh RF Module is a next generation long range radio. This comes in all three XBee form factors with multiple options for frequency, output power and XBee 3 features (with and without).

Applicable firmware and hardware

This user guide supports the following firmware:

- v.30xx DigiMesh

It supports the following hardware:

- XBee 3

Safety instructions

Safety instructions

XBee modules

- The XBee radio module cannot be guaranteed operation due to the radio link and so should not be used for interlocks in safety critical devices such as machines or automotive applications.
- The XBee radio module have not been approved for use in (this list is not exhaustive):
  - medical devices
  - nuclear applications
  - explosive or flammable atmospheres
- There are no user serviceable components inside the XBee radio module. Do not remove the shield or modify the XBee in any way. Modifications may exclude the module from any warranty and can cause the XBee radio to operate outside of regulatory compliance for a given country, leading to the possible illegal operation of the radio.
- Use industry standard ESD protection when handling the XBee module.
- Take care while handling to avoid electrical damage to the PCB and components.
- Do not expose XBee radio modules to water or moisture.
- Use this product with the antennas specified in the XBee module user guides.
- The end user must be told how to remove power from the XBee radio module or to locate the antennas 20 cm from humans or animals.

Инструкции за безопасност

XBee модули

- Радио модулът XBee не може да бъде гарантиран за работа поради радиовръзката и затова не трябва да се използва за блокировки в критични за безопасността устройства като машини или автомобилни приложения.
- Радио модулът XBee не е одобрен за използване в (този списък не е изчерпателен):
  - медицински изделия
  - ядрени приложения
  - експлозивна или запалима атмосфера
- В радиомодула XBee няма компоненти, които могат да се обслужват от потребителя. Не премахвайте щита и не модифицирайте XBee по никакъв начин. Модификациите могат да изключат модула от всякаква гаранция и да накарат радиото XBee да работи извън регулаторното съответствие за дадена държава, което води до възможна незаконна работа на радиото.
- Използвайте стандартна ESD защита при работа с XBee модула.
- Внимавайте, докато боравите, за да избегнете електрически повреди на печатната платка и компонентите.
- Не излагайте радиомодулите XBee на вода или влага.

- Използвайте този продукт с антените, посочени в ръководствата за потребителя на модула XBee.
- Крайният потребител трябва да бъде казано как да премахне захранването от радиомодула XBee или да разположи антените на 20 см от хора или животни.

Sigurnosne upute

XBee moduli

- Radio modulu XBee ne može se jamčiti rad zbog radio veze i stoga se ne smije koristiti za blokade u sigurnosnim kritičnim uređajima kao što su strojevi ili automobilske aplikacije.
- XBee radio modul nije odobren za upotrebu u (ovaj popis nije konačan):
  - medicinskih uređaja
  - nuklearne primjene
  - eksplozivne ili zapaljive atmosfere
- Unutar XBee radio modula nema komponenti koje može servisirati korisnik. Nemojte uklanjati štit i ni na koji način modificirati XBee. Izmjene mogu isključiti modul iz bilo kakvog jamstva i mogu uzrokovati rad XBee radija izvan usklađenosti s propisima za određenu zemlju, što može dovesti do mogućeg nezakonitog rada radija.
- Koristite standardnu ESD zaštitu pri rukovanju XBee modulom.
- Budite oprezni tijekom rukovanja kako biste izbjegli električna oštećenja PCB-a i komponenti.
- Ne izlažite XBee radio module vodi ili vlazi.
- Koristite ovaj proizvod s antenama navedenim u korisničkim vodičima za XBee modul.
- Krajnjem korisniku se mora reći kako da isključi napajanje iz XBee radio modula ili da locira antene 20 cm od ljudi ili životinja.

Bezpečnostní instrukce

moduly XBee

- Rádiový modul XBee nemůže zaručit provoz kvůli rádiovému spojení, a proto by neměl být používán pro blokování v zařízeních kritických z hlediska bezpečnosti, jako jsou stroje nebo automobilové aplikace.
- Rádiový modul XBee nebyl schválen pro použití v (tento seznam není vyčerpávající):
  - zdravotnické prostředky
  - jaderné aplikace
  - výbušné nebo hořlavé atmosféry
- Uvnitř rádiového modulu XBee nejsou žádné uživatelsky opravitelné součásti. Neodstraňujte štít ani nijak neupravujte XBee. Úpravy mohou vyjmout modul z jakékoli záruky a mohou způsobit, že rádio XBee bude fungovat mimo zákonnou shodu pro danou zemi, což povede k možnému nezákonnému provozu rádia.
- Při manipulaci s modulem XBee používejte standardní ochranu ESD.

- Při manipulaci buďte opatrní, aby nedošlo k elektrickému poškození desky plošných spojů a součástí.
- Nevystavujte rádiové moduly XBee vodě nebo vlhkosti.
- Používejte tento produkt s anténami uvedenými v uživatelských příručkách modulu XBee.
- Koncový uživatel musí být informován, jak odpojit napájení rádiového modulu XBee nebo jak umístit antény 20 cm od lidí nebo zvířat.

Sikkerhedsinstruktioner

XBee moduler

- XBee-radiomodulet kan ikke garanteres drift på grund af radioforbindelsen og bør derfor ikke bruges til aflåsninger i sikkerhedskritiske enheder såsom maskiner eller bilapplikationer.
- XBee-radiomodulet er ikke godkendt til brug i (denne liste er ikke udtømmende):
  - medicinsk udstyr
  - nukleare applikationer
  - eksplosive eller brandfarlige atmosfærer
- Der er ingen komponenter, der kan repareres af brugeren, inde i XBee-radiomodulet. Fjern ikke skjoldet eller modificer XBee på nogen måde. Ændringer kan udelukke modulet fra enhver garanti og kan få XBee-radioen til at fungere uden for lovgivningsoverholdelse for et givet land, hvilket kan føre til den mulige ulovlige drift af radioen.
- Brug industristandard ESD-beskyttelse, når du håndterer XBee-modulet.
- Vær forsigtig under håndteringen for at undgå elektrisk beskadigelse af printet og komponenterne.
- Udsæt ikke XBee-radiomoduler for vand eller fugt.
- Brug dette produkt med de antenner, der er specificeret i XBee-modulets brugervejledninger.
- Slutbrugeren skal fortælles, hvordan man fjerner strømmen fra XBee-radiomodulet eller placerer antennerne 20 cm fra mennesker eller dyr.

Veiligheidsinstructies

XBee-modules

- De werking van de XBee-radiomodule kan niet worden gegarandeerd vanwege de radioverbinding en mag daarom niet worden gebruikt voor vergrendelingen in veiligheidskritieke apparaten zoals machines of autotoepassingen.
- De XBee-radiomodule is niet goedgekeurd voor gebruik in (deze lijst is niet uitputtend):
  - o medische apparaten
  - o nucleaire toepassingen
  - o explosieve of ontvlambare atmosferen

- Er zijn geen door de gebruiker te onderhouden componenten in de XBee-radiomodule. Verwijder het schild niet en wijzig de XBee op geen enkele manier. Modificaties kunnen de module uitsluiten van enige garantie en kunnen ertoe leiden dat de XBee-radio werkt buiten de regelgeving voor een bepaald land, wat kan leiden tot de mogelijke illegale werking van de radio.
- Gebruik industriestandaard ESD-bescherming bij het hanteren van de XBee-module.
- Wees voorzichtig bij het hanteren om elektrische schade aan de printplaat en componenten te voorkomen.
- Stel XBee-radiomodules niet bloot aan water of vocht.
- Gebruik dit product met de antennes die zijn gespecificeerd in de gebruikershandleidingen van de XBee-module.
- De eindgebruiker moet worden verteld hoe de voeding van de XBee-radiomodule moet worden losgekoppeld of hoe de antennes op 20 cm van mensen of dieren moeten worden geplaatst.

Ohutusjuhised

XBee moodulid

- XBee raadiomooduli tööd ei saa raadiolingi tõttu garanteerida ja seetõttu ei tohiks seda kasutada ohutuse seisukohalt oluliste seadmete (nt masinad või autorakendused) blokeerimiseks.
- XBee raadiomoodulit ei ole heaks kiidetud kasutamiseks (see loetelu ei ole ammendav):
  - meditsiiniseadmed
  - tuumarakendused
  - plahvatusohtlik või tuleohtlik keskkond
- XBee raadiomoodulis ei ole kasutaja poolt hooldatavaid komponente. Ärge eemaldage kaitset ega muutke XBee mingil viisil. Muudatused võivad mooduli garantiist välja jätta ja XBee raadio töötab väljaspool antud riigi regulatiivseid vastavusi, põhjustades raadio võimaliku ebaseadusliku kasutamise.
- Kasutage XBee mooduli käsitsemisel tööstusharu standardset ESD-kaitset.
- Olge käsitsemisel ettevaatlik, et vältida PCB ja komponentide elektrikahjustusi.
- Ärge jätke XBee raadiomooduleid vee või niiskuse kätte.
- Kasutage seda toodet XBee mooduli kasutusjuhendis kirjeldatud antennidega.
- Lõppkasutajale tuleb öelda, kuidas XBee raadiomoodulilt toide eemaldada või antennid inimestest või loomadest 20 cm kaugusele paigutada.

Turvallisuusohjeet

XBee moduulit

- XBee-radiomoduulin toimintaa ei voida taata radiolinkin vuoksi, joten sitä ei tule käyttää turvallisuuden kannalta kriittisten laitteiden, kuten koneiden tai autosovellusten, lukitsemiseen.

- XBee-radiomoduulia ei ole hyväksytty käytettäväksi (tämä luettelo ei ole tyhjentävä):
  - lääketieteelliset laitteet
  - ydinvoimasovellukset
  - räjähdysvaarallisiin tai syttyviin tiloihin
- XBee-radiomoduulin sisällä ei ole käyttäjän huollettavia osia. Älä poista suojusta tai muokkaa XBeetä millään tavalla. Muutokset voivat sulkea moduulin takuun ulkopuolelle ja aiheuttaa sen, että XBee-radio toimii tietyn maan säädöstenmukaisuuden ulkopuolella, mikä johtaa radion mahdolliseen laittomaan käyttöön.
- Käytä alan standardia ESD-suojausta käsitellessäsi XBee-moduulia.
- Ole varovainen käsitellessäsi, jotta vältät piirilevyn ja komponenttien sähkövauriot.
- Älä altista XBee-radiomoduuleja vedelle tai kosteudelle.
- Käytä tätä tuotetta XBee-moduulin käyttöoppaissa määriteltyjen antennien kanssa.
- Loppukäyttäjälle on kerrottava, kuinka XBee-radiomoduulin virta katkaistaan tai antennit sijoitetaan 20 cm:n etäisyydelle ihmisistä tai eläimistä.

Consignes de sécurité

Modules XBee

- Le fonctionnement du module radio XBee ne peut pas être garanti en raison de la liaison radio et ne doit donc pas être utilisé pour les verrouillages dans des dispositifs critiques pour la sécurité tels que des machines ou des applications automobiles.
- Le module radio XBee n'a pas été approuvé pour une utilisation dans (cette liste n'est pas exhaustive) :
  - dispositifs médicaux
  - applications nucléaires
  - atmosphères explosives ou inflammables
- Il n'y a aucun composant réparable par l'utilisateur à l'intérieur du module radio XBee. Ne retirez pas la protection et ne modifiez en aucune façon le XBee. Les modifications peuvent exclure le module de toute garantie et peuvent entraîner le fonctionnement de la radio XBee en dehors de la conformité réglementaire pour un pays donné, ce qui peut entraîner un fonctionnement illégal de la radio.
- Utilisez la protection ESD standard de l'industrie lors de la manipulation du module XBee.
- Soyez prudent lors de la manipulation afin d'éviter des dommages électriques au circuit imprimé et aux composants.
- N'exposez pas les modules radio XBee à l'eau ou à l'humidité.
- Utilisez ce produit avec les antennes spécifiées dans les guides d'utilisation du module XBee.
- L'utilisateur final doit savoir comment couper l'alimentation du module radio XBee ou placer les antennes à 20 cm des humains ou des animaux.

Sicherheitshinweise

XBee-Module

- Der Betrieb des XBee-Funkmoduls kann aufgrund der Funkverbindung nicht garantiert werden und sollte daher nicht für Verriegelungen in sicherheitskritischen Geräten wie Maschinen oder Automobilanwendungen verwendet werden.
- Das XBee-Funkmodul ist nicht zugelassen für den Einsatz in (diese Liste ist nicht vollständig):
  - Medizinprodukte
  - nukleare Anwendungen
  - explosive oder brennbare Atmosphären
- Das XBee-Funkmodul enthält keine vom Benutzer zu wartenden Komponenten. Entfernen Sie nicht die Abschirmung oder modifizieren Sie das XBee in irgendeiner Weise. Modifikationen können das Modul von jeglicher Garantie ausschließen und dazu führen, dass das XBee-Funkgerät außerhalb der gesetzlichen Vorschriften für ein bestimmtes Land betrieben wird, was zu einem möglichen illegalen Betrieb des Funkgeräts führen kann.
- Verwenden Sie beim Umgang mit dem XBee-Modul ESD-Schutz nach Industriestandard.
- Seien Sie vorsichtig bei der Handhabung, um elektrische Schäden an der Leiterplatte und den Komponenten zu vermeiden.
- XBee-Funkmodule nicht Wasser oder Feuchtigkeit aussetzen.
- Verwenden Sie dieses Produkt mit den in den Benutzerhandbüchern des XBee-Moduls angegebenen Antennen.
- Dem Endbenutzer muss mitgeteilt werden, wie er das XBee-Funkmodul von der Stromversorgung trennt oder die Antennen 20 cm von Menschen oder Tieren entfernt aufstellt.

Οδηγίες ασφαλείας

Μονάδες XBee

- Η μονάδα ραδιοφώνου XBee δεν μπορεί να εγγυηθεί τη λειτουργία της λόγω της ραδιοζεύξης και επομένως δεν πρέπει να χρησιμοποιείται για ασφάλειες σε κρίσιμες για την ασφάλεια συσκευές, όπως μηχανήματα ή εφαρμογές αυτοκινήτου.
- Η μονάδα ραδιοφώνου XBee δεν έχει εγκριθεί για χρήση σε (αυτή η λίστα δεν είναι εξαντλητική):
  - ιατροτεχνολογικά προϊόντα
  - πυρηνικές εφαρμογές
  - εκρηκτικές ή εύφλεκτες ατμόσφαιρες
- Δεν υπάρχουν εξαρτήματα που να μπορούν να επισκευαστούν από το χρήστη μέσα στη μονάδα ραδιοφώνου XBee. Μην αφαιρείτε την ασπίδα και μην τροποποιείτε το XBee με κανέναν τρόπο. Οι τροποποιήσεις ενδέχεται να αποκλείουν τη μονάδα από οποιαδήποτε εγγύηση και μπορεί να προκαλέσουν τη λειτουργία του ραδιοφώνου XBee εκτός της συμμόρφωσης με τους κανονισμούς για μια δεδομένη χώρα, οδηγώντας σε πιθανή παράνομη λειτουργία του ραδιοφώνου.
- Χρησιμοποιήστε βιομηχανική προστασία ESD κατά το χειρισμό της μονάδας XBee.

- Προσέχετε κατά το χειρισμό για να αποφύγετε ηλεκτρική βλάβη στο PCB και στα εξαρτήματα.
- Μην εκθέτετε τις μονάδες ραδιοφώνου XBee σε νερό ή υγρασία.
- Χρησιμοποιήστε αυτό το προϊόν με τις κεραίες που καθορίζονται στους οδηγούς χρήσης της μονάδας XBee.
- Πρέπει να ενημερωθεί ο τελικός χρήστης πώς να αφαιρέσει την τροφοδοσία από τη μονάδα ραδιοφώνου XBee ή να εντοπίσει τις κεραίες σε απόσταση 20 cm από ανθρώπους ή ζώα.

Biztonsági utasítások

XBee modulok

• Az XBee rádiómodul működése nem garantálható a rádiókapcsolat miatt, ezért nem használható biztonsági szempontból kritikus eszközök, például gépek vagy autóipari alkalmazások reteszelésére.

• Az XBee rádiómodul nem engedélyezett a következő területeken való használatra (ez a lista nem teljes):

o orvosi eszközök

o nukleáris alkalmazások

o robbanásveszélyes vagy gyúlékony légkör

• Az XBee rádiómodulban nincsenek felhasználó által javítható alkatrészek. Ne távolítsa el a pajzsot, és semmilyen módon ne módosítsa az XBee-t. A módosítások kizárhatják a modult a jótállásból, és az XBee rádió működését az adott ország jogszabályi előírásaitól eltérően okozhatják, ami a rádió esetleges illegális működéséhez vezethet.

• Az XBee modul kezelésekor használjon ipari szabványos ESD védelmet.

• A kezelés során ügyeljen arra, hogy elkerülje a PCB és az alkatrészek elektromos károsodását.

• Ne tegye ki az XBee rádiómodulokat víznek vagy nedvességnek.

• Használja ezt a terméket az XBee modul használati útmutatójában meghatározott antennákkal.

• A végfelhasználót tájékoztatni kell arról, hogyan távolítsa el az XBee rádiómodul áramellátását, vagy hogyan helyezze el az antennákat az emberektől vagy állatoktól 20 cm-re.

Istruzioni di sicurezza

Moduli XBee

- Il funzionamento del modulo radio XBee non può essere garantito a causa del collegamento radio e quindi non deve essere utilizzato per gli interblocchi in dispositivi critici per la sicurezza come macchine o applicazioni automobilistiche.
- Il modulo radio XBee non è stato approvato per l'uso in (questo elenco non è esaustivo):
  - dispositivi medici
  - applicazioni nucleari
  - atmosfere esplosive o infiammabili
- Non ci sono componenti riparabili dall'utente all'interno del modulo radio XBee. Non rimuovere lo scudo o modificare in alcun modo l'XBee. Le modifiche possono escludere il modulo da qualsiasi garanzia e possono causare il funzionamento della radio XBee al di fuori della conformità normativa per un determinato paese, portando al possibile funzionamento

illegale della radio.

- Utilizzare la protezione ESD standard del settore durante la manipolazione del modulo XBee.
- Prestare attenzione durante la manipolazione per evitare danni elettrici al PCB e ai componenti.
- Non esporre i moduli radio XBee all'acqua o all'umidità.
- Utilizzare questo prodotto con le antenne specificate nelle guide per l'utente del modulo XBee.
- L'utente finale deve sapere come togliere l'alimentazione al modulo radio XBee o come posizionare le antenne a 20 cm da persone o animali.

Drošības instrukcijas

XBee moduļi

- Radio moduļa XBee darbība nevar tikt garantēta radio savienojuma dēļ, tāpēc to nevajadzētu izmantot bloķēšanai drošības ziņā kritiskās ierīcēs, piemēram, mašīnās vai automobiļos.
- XBee radio modulis nav apstiprināts lietošanai (šis saraksts nav pilnīgs):
  - medicīniskās ierīces
  - kodolprogrammas
  - sprādzienbīstamā vai uzliesmojošā vidē
- XBee radio moduļa iekšpusē nav neviena komponenta, ko lietotājs varētu apkopt. Nenoņemiet vairogu un nekādā veidā nepārveidojiet XBee. Modifikācijas rezultātā modulis var tikt izslēgts no jebkādas garantijas un var izraisīt XBee radio darbību, kas neatbilst noteiktās valsts normatīvajiem aktiem, izraisot iespējamu nelegālu radio darbību.
- Strādājot ar XBee moduli, izmantojiet nozares standarta ESD aizsardzību.
- Rīkojoties, rīkojieties uzmanīgi, lai izvairītos no PCB un komponentu elektriskiem bojājumiem.
- Nepakļaujiet XBee radio moduļus ūdens vai mitruma iedarbībai.
- Izmantojiet šo izstrādājumu ar antenām, kas norādītas XBee moduļa lietotāja rokasgrāmatās.
- Galalietotājam ir jāpaskaidro, kā atvienot XBee radio moduļa strāvu vai novietot antenas 20 cm attālumā no cilvēkiem vai dzīvniekiem.

Saugos instrukcijos

XBee moduliai

- Negalima garantuoti, kad „XBee“ radijo modulis veiks dėl radijo ryšio, todėl jo neturėtų būti naudojamas blokuoti saugai svarbiuose įrenginiuose, pvz., mašinose ar automobiliuose.
- XBee radijo modulis nebuvo patvirtintas naudoti (šis sąrašas nėra baigtinis):
  - medicinos prietaisai
  - branduolinės programos
  - sprogioje ar degioje aplinkoje

- XBee radijo modulio viduje nėra komponentų, kuriuos vartotojas galėtų prižiūrėti. Jokiu būdu nenuimkite skydo ir nekeiskite XBee. Dėl modifikacijų moduliui gali būti netaikoma jokia garantija, o „XBee“ radijas gali veikti ne pagal tam tikros šalies norminius reikalavimus, o tai gali sukelti neteisėtą radijo naudojimą.
- Dirbdami su XBee moduliu naudokite pramonės standartinę ESD apsaugą.
- Dirbdami būkite atsargūs, kad nepažeistumėte PCB ir komponentų.
- Saugokite XBee radijo modulius nuo vandens ar drėgmės.
- Naudokite šį gaminį su antenomis, nurodytomis XBee modulio vartotojo vadove.
- Galutiniam vartotojui turi būti paaiškinta, kaip atjungti XBee radijo modulio maitinimą arba nustatyti antenas 20 cm atstumu nuo žmonių ar gyvūnų.

Sikkerhetsinstruksjoner

XBee-moduler

- XBee-radiomodulen kan ikke garanteres drift på grunn av radiolinken, og bør derfor ikke brukes til forriglinger i sikkerhetskritiske enheter som maskiner eller bilapplikasjoner.
- XBee-radiomodulen er ikke godkjent for bruk i (denne listen er ikke uttømmende):
  - medisinsk utstyr
  - kjernefysiske applikasjoner
  - eksplosive eller brennbare atmosfærer
- Det er ingen komponenter som kan repareres av brukeren inne i XBee-radiomodulen. Ikke fjern skjoldet eller modifiser XBee på noen måte. Endringer kan ekskludere modulen fra enhver garanti og kan føre til at XBee-radioen fungerer utenfor regelverket for et gitt land, noe som kan føre til ulovlig drift av radioen.
- Bruk industristandard ESD-beskyttelse når du håndterer XBee-modulen.
- Vær forsiktig ved håndtering for å unngå elektrisk skade på PCB og komponenter.
- Ikke utsett XBee radiomoduler for vann eller fuktighet.
- Bruk dette produktet med antennene spesifisert i XBee-modulens brukerveiledninger.
- Sluttbrukeren må bli fortalt hvordan man fjerner strømmen fra XBee-radiomodulen eller plasserer antennene 20 cm fra mennesker eller dyr.

Instrukcje bezpieczeństwa

Moduły XBee

- Moduł radiowy XBee nie może zagwarantować działania ze względu na łącze radiowe, dlatego nie należy go używać do blokad w urządzeniach o krytycznym znaczeniu dla bezpieczeństwa, takich jak maszyny lub aplikacje motoryzacyjne.
- Moduł radiowy XBee nie został dopuszczony do użytku w (lista ta nie jest wyczerpująca):

- wyroby medyczne
- zastosowania nuklearne
- atmosferach wybuchowych lub łatwopalnych

- Wewnątrz modułu radiowego XBee nie ma żadnych elementów, które mogłyby być serwisowane przez użytkownika. Nie zdejmuj osłony ani nie modyfikuj XBee w żaden sposób. Modyfikacje mogą wykluczyć moduł z jakiejkolwiek gwarancji i spowodować, że radio XBee będzie działać niezgodnie z przepisami obowiązującymi w danym kraju, co może prowadzić do nielegalnego działania radia.
- Podczas obsługi modułu XBee należy stosować standardową ochronę ESD.
- Podczas obsługi należy zachować ostrożność, aby uniknąć uszkodzeń elektrycznych PCB i komponentów.
- Nie wystawiaj modułów radiowych XBee na działanie wody lub wilgoci.
- Używaj tego produktu z antenami określonymi w podręcznikach użytkownika modułu XBee.
- Użytkownik końcowy musi zostać poinformowany, jak odłączyć zasilanie modułu radiowego XBee lub zlokalizować anteny w odległości 20 cm od ludzi lub zwierząt.

Instruções de segurança

Módulos XBee

- • O módulo de rádio XBee não pode ter operação garantida devido ao link de rádio e, portanto, não deve ser usado para intertravamentos em dispositivos críticos de segurança, como máquinas ou aplicações automotivas.
- • O módulo de rádio XBee não foi aprovado para uso em (esta lista não é exaustiva):
  - o dispositivos médicos
  - o aplicações nucleares
  - o atmosferas explosivas ou inflamáveis
- • Não há componentes que possam ser reparados pelo usuário dentro do módulo de rádio XBee. Não remova a blindagem nem modifique o XBee de forma alguma. As modificações podem excluir o módulo de qualquer garantia e fazer com que o rádio XBee opere fora da conformidade regulatória de um determinado país, levando à possível operação ilegal do rádio.
- • Use proteção ESD padrão da indústria ao manusear o módulo XBee.
- • Tome cuidado ao manusear para evitar danos elétricos à PCB e aos componentes.
- • Não exponha os módulos de rádio XBee à água ou umidade.
- • Use este produto com as antenas especificadas nos guias do usuário do módulo XBee.
- • O usuário final deve ser informado sobre como remover a energia do módulo de rádio XBee ou localizar as antenas a 20 cm de humanos ou animais.

Instructiuni de siguranta

module XBee

- Nu se poate garanta funcţionarea modulului radio XBee din cauza conexiunii radio şi, prin urmare, nu trebuie utilizat pentru interblocări în dispozitive critice pentru siguranţă, cum ar fi maşini sau aplicaţii auto.
- Modulul radio XBee nu a fost aprobat pentru utilizare în (această listă nu este exhaustivă):
  - dispozitive medicale
  - aplicaţii nucleare
  - atmosfere explozive sau inflamabile
- Nu există componente care să poată fi reparate de utilizator în interiorul modulului radio XBee. Nu îndepărtaţi scutul şi nu modificaţi XBee în niciun fel. Modificările pot exclude modulul din orice garanţie şi pot face ca radioul XBee să funcţioneze în afara conformităţii cu reglementările pentru o anumită ţară, ceea ce duce la o posibilă funcţionare ilegală a radioului.
- Folosiţi protecţia ESD standard în industrie când manipulaţi modulul XBee.
- Aveţi grijă în timpul manipulării pentru a evita deteriorarea electrică a PCB-ului şi a componentelor.
- Nu expuneţi modulele radio XBee la apă sau umezeală.
- Utilizaţi acest produs cu antenele specificate în ghidurile utilizatorului modulului XBee.
- Utilizatorului final trebuie să i se spună cum să scoată alimentarea de la modulul radio XBee sau să găsească antenele la 20 cm de oameni sau animale.

Bezpečnostné inštrukcie

moduly XBee

- Rádiový modul XBee nemôže byť zaručený kvôli rádiovému spojeniu, a preto by sa nemal používať na blokovanie v zariadeniach kritických z hľadiska bezpečnosti, ako sú stroje alebo automobilové aplikácie.
- Rádiový modul XBee nebol schválený na použitie v (tento zoznam nie je úplný):
  - zdravotnícke pomôcky
  - jadrové aplikácie
  - výbušné alebo horľavé atmosféry
- Vo vnútri rádiového modulu XBee sa nenachádzajú žiadne používateľsky opraviteľné komponenty. Neodstraňujte štít ani žiadnym spôsobom neupravujte XBee. Úpravy môžu vyňať modul zo záruky a môžu spôsobiť, že rádio XBee bude fungovať mimo zhody s predpismi pre danú krajinu, čo vedie k možnej nezákonnej prevádzke rádia.
- Pri manipulácii s modulom XBee používajte štandardnú ochranu pred ESD.
- Pri manipulácii buďte opatrní, aby ste predišli elektrickému poškodeniu dosky plošných spojov a komponentov.
- Rádiové moduly XBee nevystavujte vode ani vlhkosti.

- Tento produkt používajte s anténami špecifikovanými v používateľských príručkách modulu XBee.
- Koncový používateľ musí byť informovaný o tom, ako odpojiť napájanie rádiového modulu XBee alebo ako umiestniť antény 20 cm od ľudí alebo zvierat.

Varnostna navodila

XBee moduli

- Radijskega modula XBee ni mogoče zagotoviti delovanja zaradi radijske povezave in ga zato ne smete uporabljati za zaklepanje v varnostno kritičnih napravah, kot so stroji ali avtomobilske aplikacije.
- Radijski modul XBee ni bil odobren za uporabo v (ta seznam ni izčrpen):
  - medicinskih pripomočkov
  - jedrske aplikacije
  - eksplozivne ali vnetljive atmosfere
- V radijskem modulu XBee ni komponent, ki bi jih lahko popravil uporabnik. Ne odstranjujte ščita in na noben način ne spreminjajte XBee. Spremembe lahko modul izključijo iz kakršne koli garancije in lahko povzročijo, da radio XBee deluje zunaj zakonske skladnosti za dano državo, kar vodi do možnega nezakonitega delovanja radia.
- Pri ravnanju z modulom XBee uporabite standardno industrijsko zaščito pred ESD.
- Pri rokovanju pazite, da se izognete električnim poškodbam tiskanega vezja in komponent.
- Radijskih modulov XBee ne izpostavljajte vodi ali vlagi.
- Ta izdelek uporabljajte z antenami, navedenimi v uporabniških priročnikih modula XBee.
- Končnemu uporabniku je treba povedati, kako odstraniti napajanje z radijskega modula XBee ali naj locira antene 20 cm od ljudi ali živali.

Módulos XBee

- No se puede garantizar el funcionamiento del módulo de radio XBee debido al enlace de radio y, por lo tanto, no debe usarse para enclavamientos en dispositivos críticos para la seguridad, como máquinas o aplicaciones automotrices.
- El módulo de radio XBee no ha sido aprobado para su uso en (esta lista no es exhaustiva):
  - dispositivos médicos
  - aplicaciones nucleares
  - atmósferas explosivas o inflamables
- No hay componentes reparables por el usuario dentro del módulo de radio XBee. No quite el escudo ni modifique el XBee de ninguna manera. Las modificaciones pueden excluir el módulo de cualquier garantía y pueden hacer que la radio XBee funcione fuera del cumplimiento normativo de un país determinado, lo que puede provocar una operación ilegal de la radio.
- Utilice la protección ESD estándar de la industria al manipular el módulo XBee.
- Tenga cuidado al manipularlo para evitar daños eléctricos en la PCB y los componentes.

- No exponga los módulos de radio XBee al agua ni a la humedad.
- Utilice este producto con las antenas especificadas en las guías de usuario del módulo XBee.
- Se debe indicar al usuario final cómo desconectar la alimentación del módulo de radio XBee o ubicar las antenas a 20 cm de personas o animales.

Säkerhets instruktioner

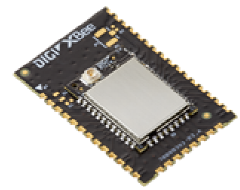
XBee-moduler

- XBee-radiomodulen kan inte garanteras funktion på grund av radiolänken och bör därför inte användas för förreglingar i säkerhetskritiska enheter som maskiner eller biltillämpningar.
- XBee-radiomodulen har inte godkänts för användning i (denna lista är inte uttömmande):
  - medicinsk utrustning
  - kärnkraftstillämpningar
  - explosiv eller brandfarlig atmosfär
- Det finns inga komponenter som användaren kan reparera inuti XBee-radiomodulen. Ta inte bort skölden eller modifiera XBee på något sätt. Ändringar kan utesluta modulen från alla garantier och kan göra att XBee-radion fungerar utanför bestämmelserna för ett visst land, vilket kan leda till att radion kan användas olagligt.
- Använd industristandard ESD-skydd när du hanterar XBee-modulen.
- Var försiktig vid hanteringen för att undvika elektriska skador på kretskortet och komponenterna.
- Utsätt inte XBee radiomoduler för vatten eller fukt.
- Använd den här produkten med antennerna som specificeras i XBee-modulens användarguider.
- Slutanvändaren måste informeras om hur man kopplar bort strömmen från XBee-radiomodulen eller för att placera antennerna 20 cm från människor eller djur.

Get started

Verify kit contents

The XBee 3 DigiMesh RF Module development kit contains the following components:

Part	
XBee 3 Zigbee SMT module (3)	
XBee Grove development board (3)	
Micro USB cable (3)	
Antenna - 2.4 GHz, half-wave dipole, 2.1 dBi, U.FL female, articulating (3)	
XBee stickers	

Assemble the hardware

This guide walks you through the steps required to assemble and disassemble the hardware components of your kit.

- Plug in the XBee 3 DigiMesh RF Module
- Unplug an XBee 3 DigiMesh RF Module

The kit includes several XBee Grove Development Boards. For more information about this hardware, see the XBee Grove Development Board documentation.

Plug in the XBee 3 DigiMesh RF Module

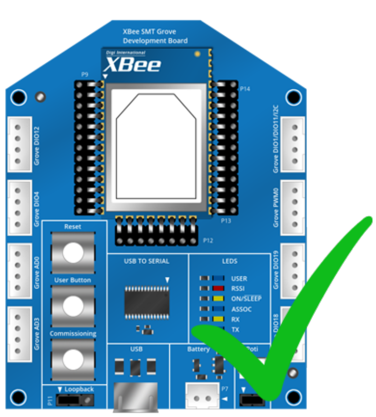
Follow these steps to connect the XBee devices to the boards included in the kit:

1. Plug one XBee 3 DigiMesh RF Module into the XBee Grove Development Board. When you connect the development board to a PC for the first time, the PC automatically installs drivers, which may take a few minutes to complete.

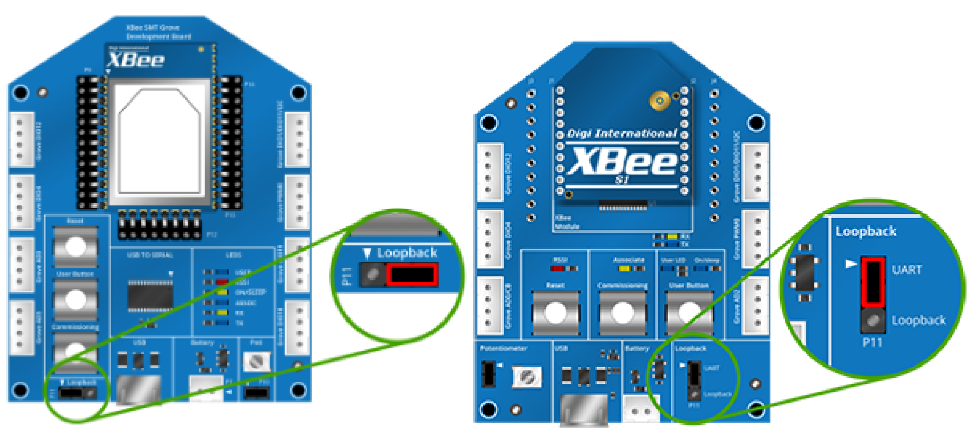
   Make sure the board is NOT powered (either by the micro USB or a battery) when you plug in the XBee module.

   For XBee SMT modules, align all XBee pins with the spring header and carefully push the module until it is hooked to the board.

   WARNING! Never insert or remove the XBee device while the power is on!



2. Once the XBee module is plugged into the board (and not before), connect the board to your computer using the micro USB cables provided.
3. Ensure the loopback jumper is in the UART position.



Unplug an XBee 3 DigiMesh RF Module

To disconnect a device from the XBee Grove Development Board:

1. Disconnect the micro USB cable from the board so it is not powered.
2. Remove the device from the board socket, taking care not to bend any of the pins. The surface mount device uses spring pins rather than a socket and has a rectangular board cutout designed to help in removing the XBee 3 DigiMesh RF Module.

CAUTION! Make sure the board is **not** powered when you remove the XBee 3 DigiMesh RF Module.

Configure the device using XCTU

XBee Configuration and Test Utility (XCTU) is a multi-platform program that enables users to interact with Digi radio frequency (RF) devices through a graphical interface. The application includes built-in tools that make it easy to set up, configure, and test Digi RF devices.

For instructions on downloading and using XCTU, see the *XCTU User Guide*.

Configure remote devices

You can communicate with remote devices over the air through a corresponding local device.

Note Using API mode on the local device allows you to send remote API commands.

These instructions show you how to configure a remote device parameter on a remote device.

1. Add two XBee devices to XCTU.
2. Load XBee 3 DigiMesh 2.4 firmware onto each device if it is not already loaded. See How to update the firmware of your modules in the *XCTU User Guide* for more information.
3. Configure the first device in API mode and name it **XBEE_A** by configuring the following parameters:

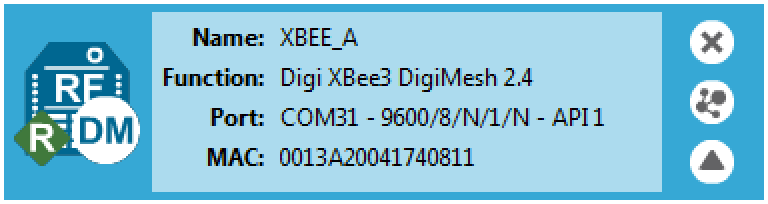
- **ID**: 2018
- **NI**: XBEE_A
- **AP**: API enabled [1]

4. Configure the second device in either API or Transparent mode, and name it **XBEE_B** by configuring the following parameters:

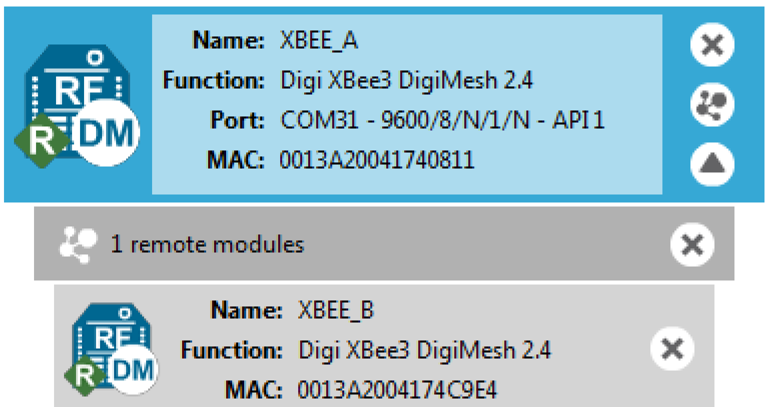
- **ID**: 2018
- **NI**: XBEE_B
- **AP**: 0 or 1

5. Disconnect XBEE_B from your computer and remove it from XCTU.
6. Connect XBEE_B to a power supply (or laptop or portable battery).

   The **Radio Modules** area should look something like this.



7. Select **XBEE_A** and click the **Discover radio nodes in the same network** button .
8. Click **Add selected devices** in the **Discovering remote devices** dialog. The discovered remote device appears below XBEE_A.



9. Select the remote device **XBEE_B** to display its current configuration settings. If you want to modify a command parameter, use the radio configuration pane.
10. Click the **Write radio settings** button to apply any changes and write it to the remote device.

Configure the devices for a range test

1. Add two devices to XCTU.
2. Select the first module and click the **Load default firmware settings** button.
3. Configure the following parameters:

   ID: 2018
   NI: LOCAL_DEVICE
   AP: API Mode Enabled [1]
4. Click the **Write radio settings** button.
5. Select the other module and click the **Default firmware settings** button.
6. Configure the following parameters:

   ID: 2018

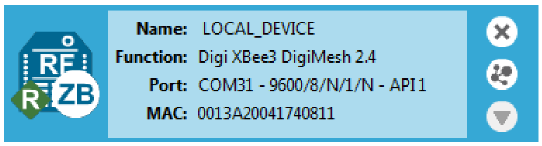
   NI: REMOTE_DEVICE
7. Click the **Write radio settings** button.

   After you write the radio settings for each device, their names appear in the **Radio Modules** area. The Port indicates that the LOCAL_DEVICE is in API mode.

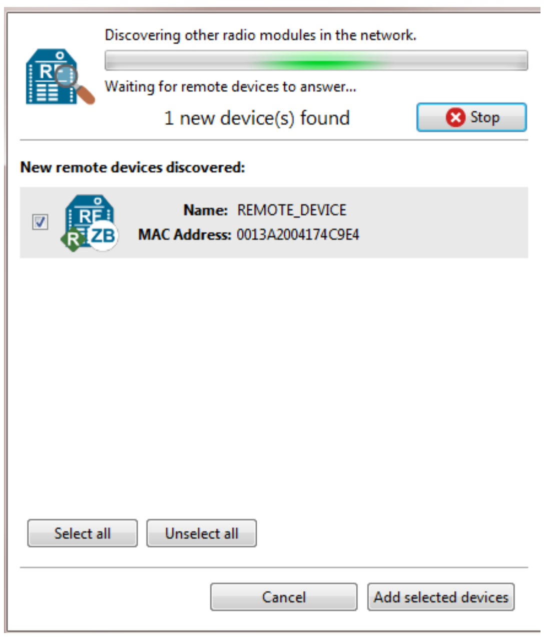
8. Disconnect REMOTE_DEVICE from the computer, remove it from XCTU, and connect it to a power supply, laptop, or portable battery.
9. Leave LOCAL_DEVICE connected to the computer.

Perform a range test

1. Go to the XCTU display for radio 1.

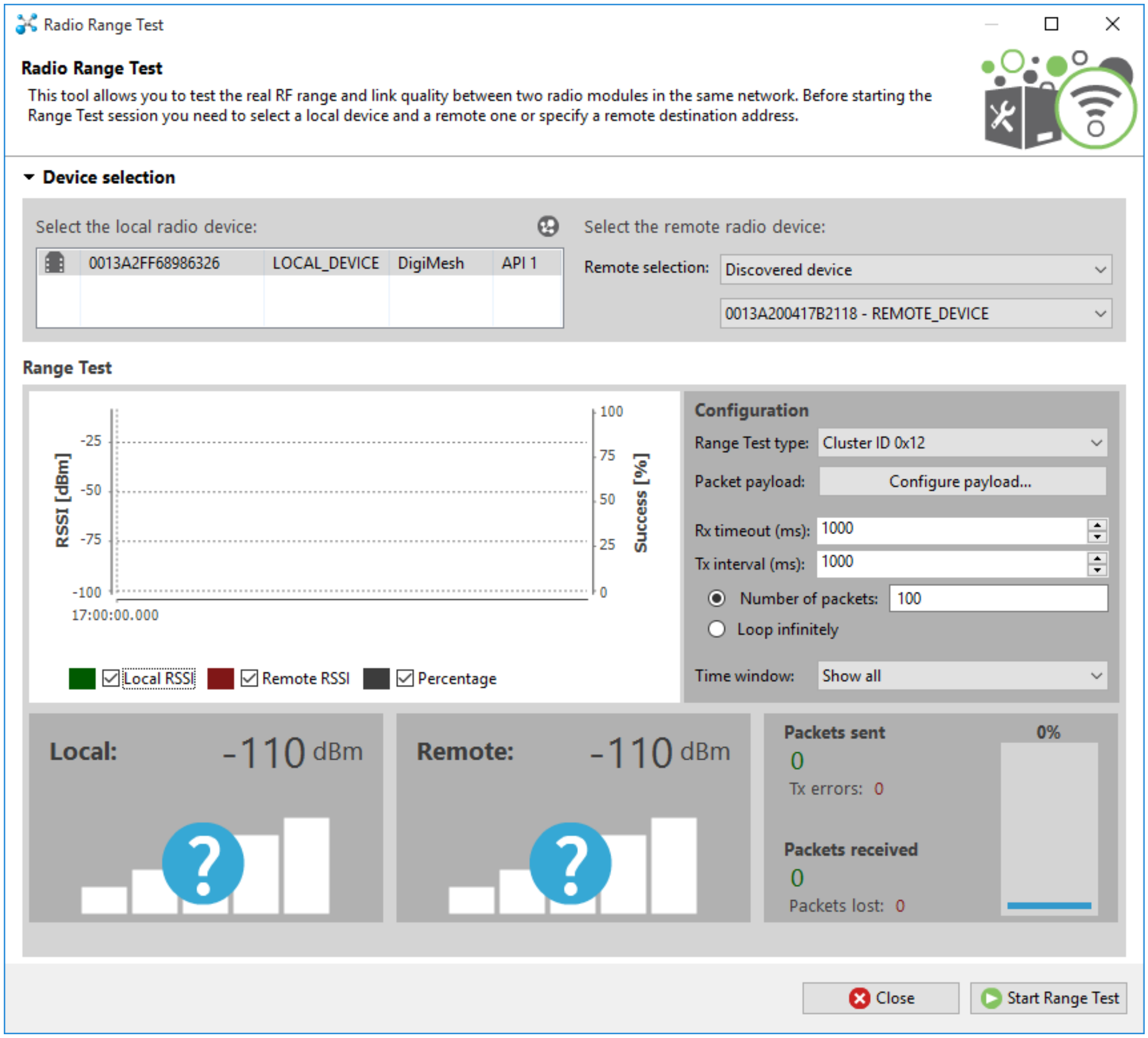


2. Click  to discover remote devices within the same network. The **Discover remote devices** dialog appears.



3. Click **Add selected devices**.

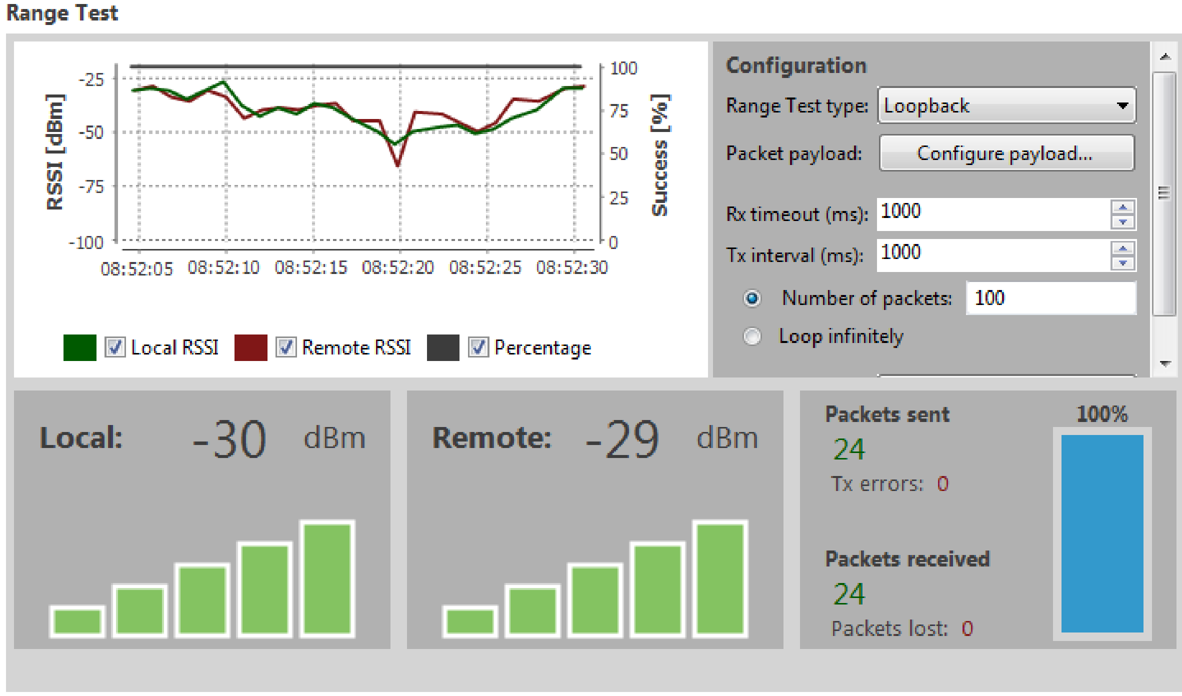
4. Click and select **Range test**. The **Radio Range Test** dialog appears.



5. In the **Select the local radio device** area, select radio 1. XCTU automatically selects the **Discovered device** option, and the **Start Range Test** button is active.

6. Click Start Range Test to begin the range test.

   If the test is running properly, the packets sent should match the packets received. You will also see the received signal strength indicator (RSSI) update for each radio after each reception.

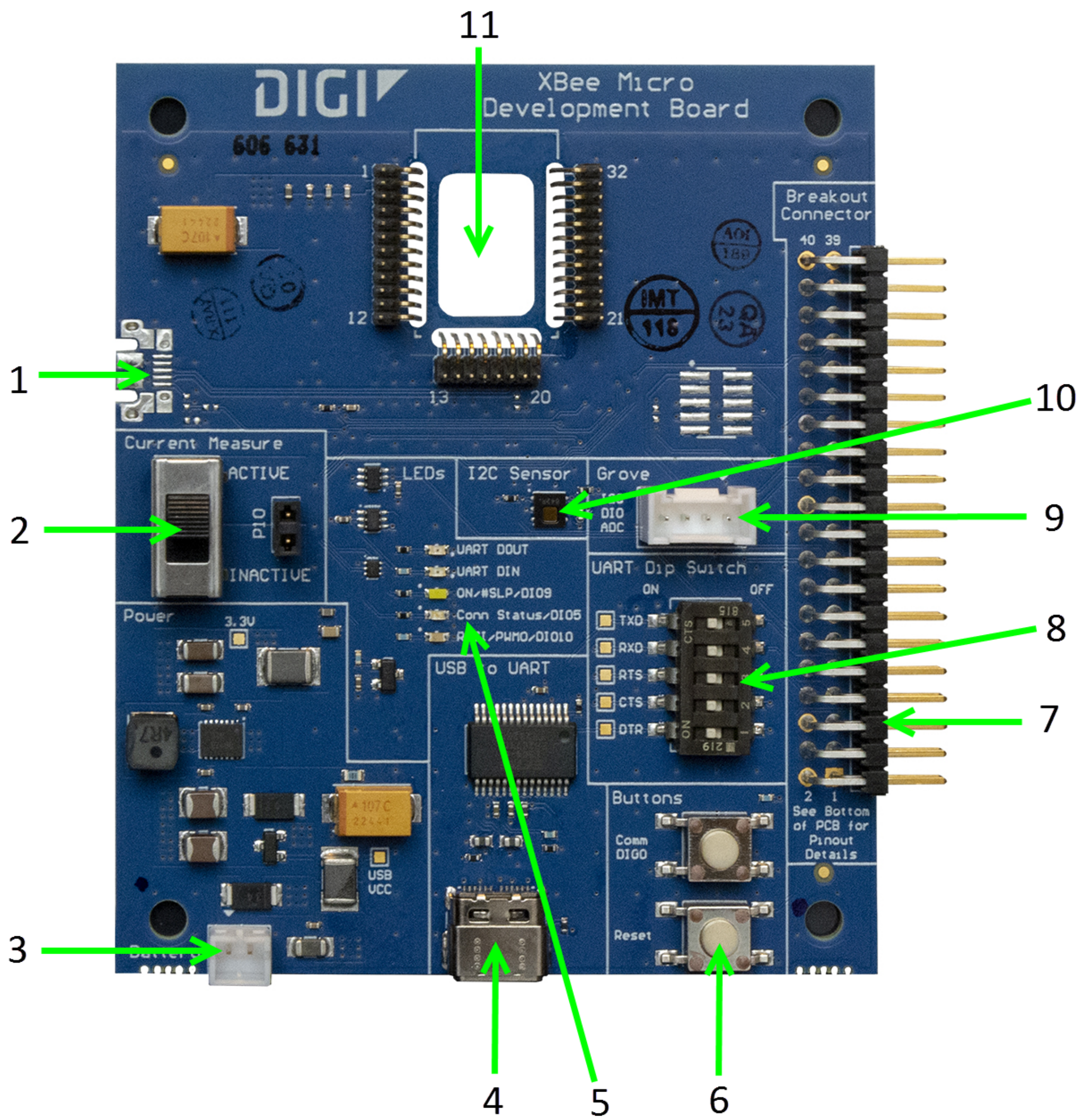


7. Move Radio 1 around to see the resulting signal strength at different distances. You can also test different power levels by reconfiguring the PL (TX Power Level) parameter on both devices.

XBIB-C Micro Mount reference

This picture shows the XBee-C Micro Mount development board and the table that follows explains the callouts in the picture.

Note This board is sold separately.

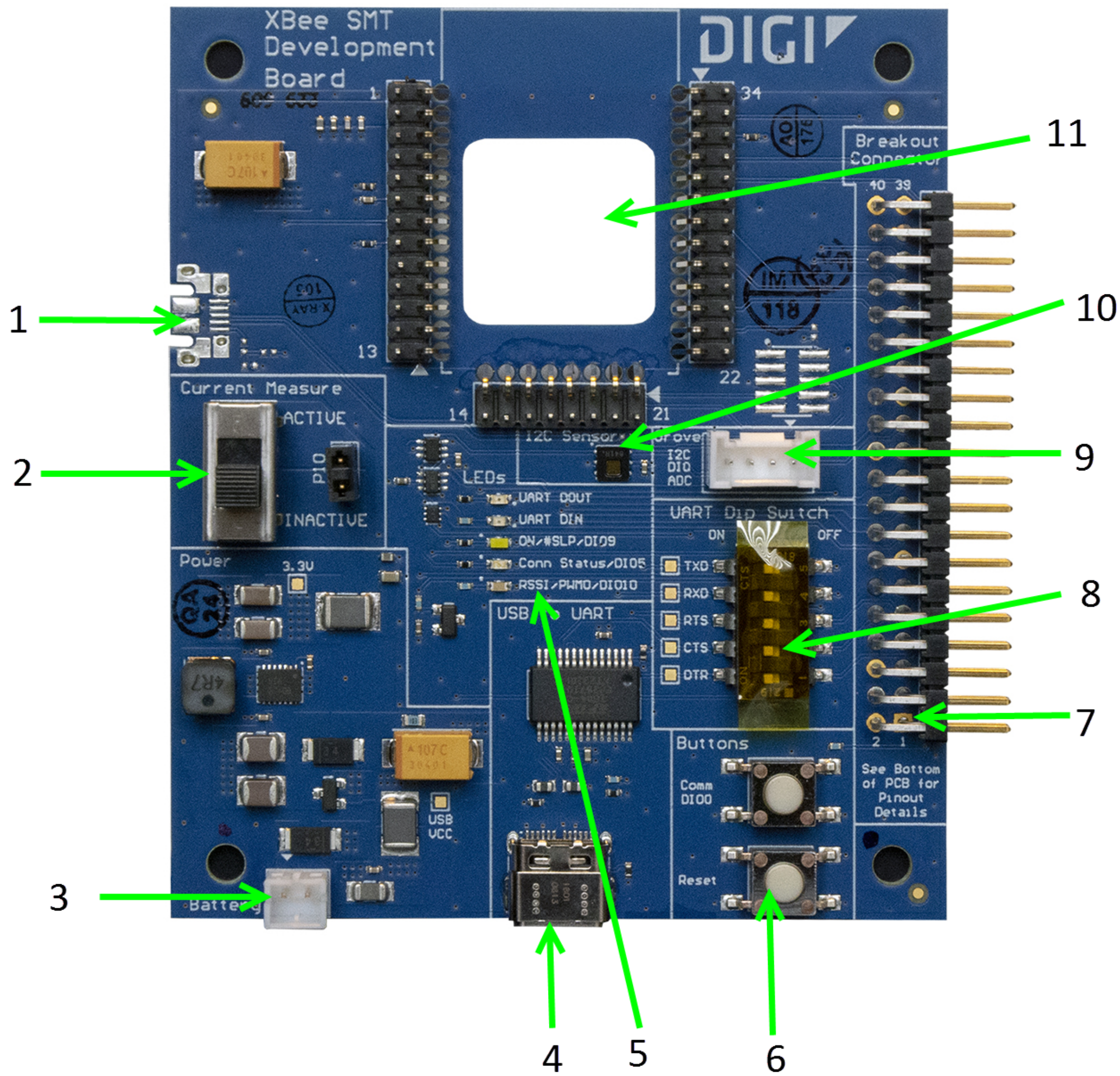


Number	Item	Description
1	Secondary USB (USB MICRO B)	Secondary USB Connector for possible future use. Not used.
2	Current Measure	Large switch controls whether current measure mode is active or inactive. When inactive, current can freely flow to the VCC pin of the XBee. When active, the VCC pin of the XBee is disconnected from the 3.3 V line on the development board. This allows current measurement to be conducted by attaching a current meter across the jumper P10.
3	Battery Connector	If desired, you can attach a battery to provide power to the development board. The voltage can range from 2 V to 5 V. The positive terminal is on the left.
4	USB-C Connector	Connects to your computer. This is connected to a USB to UART conversion chip that has the five UART lines passed to the XBee device. The UART Dip Switch can be used to disconnect these UART lines from the XBee.
5	LED indicator	Red: UART DOUT (modem sending serial/UART data to host) Green: UART DIN (modem receiving serial/UART data from host) White: ON/SLP/DIO9 Blue: Connection Status/DIO5 Yellow: RSSI/PWM0/DIO10
6	User Buttons	Comm DIO0 Button connects the Commissioning/DIO0 pin on the XBee Connector through to a 10 Ω resistor to GND when pressed. $\overline{\text{RESET}}$ Button Connects to the $\overline{\text{RESET}}$ pin on the XBee Connector to GND when pressed.
7	Breakout Connector	This 40-pin connector can be used to connect to various XBee pins as shown on the silkscreen on the bottom of the board.
8	UART Dip Switch	This dip switch allows the user to disconnect any of the primary UART lines on the XBee from the USB to UART conversion chip. This allows for testing on the primary UART lines without the USB to UART conversion chip interfering. Push Dip switches to the right to disconnect the USB to UART conversion chip from the XBee.
9	Grove Connector	This connector can be used to attach I2C enabled devices to the development board. Note that I2C needs to be available on the XBee in the board to use this functionality. Pin 1: I2C_CLK/XBee DIO1 Pin2: I2C_SDA/XBee DIO11 Pin3: VCC Pin4: GND
10	Temp/Humidity Sensor	This as a Texas Instruments HDC1080 temperature and humidity sensor. This part is accessible through I2C. Be sure that the XBee that is inserted into the development board has I2C if access to this sensor is desired.
11	XBee Socket	This is the socket for the XBee (Micro form factor).

XBIB-C SMT reference

This picture shows the XBee-C SMT development board and the table that follows explains the callouts in the picture.

Note This board is sold separately.

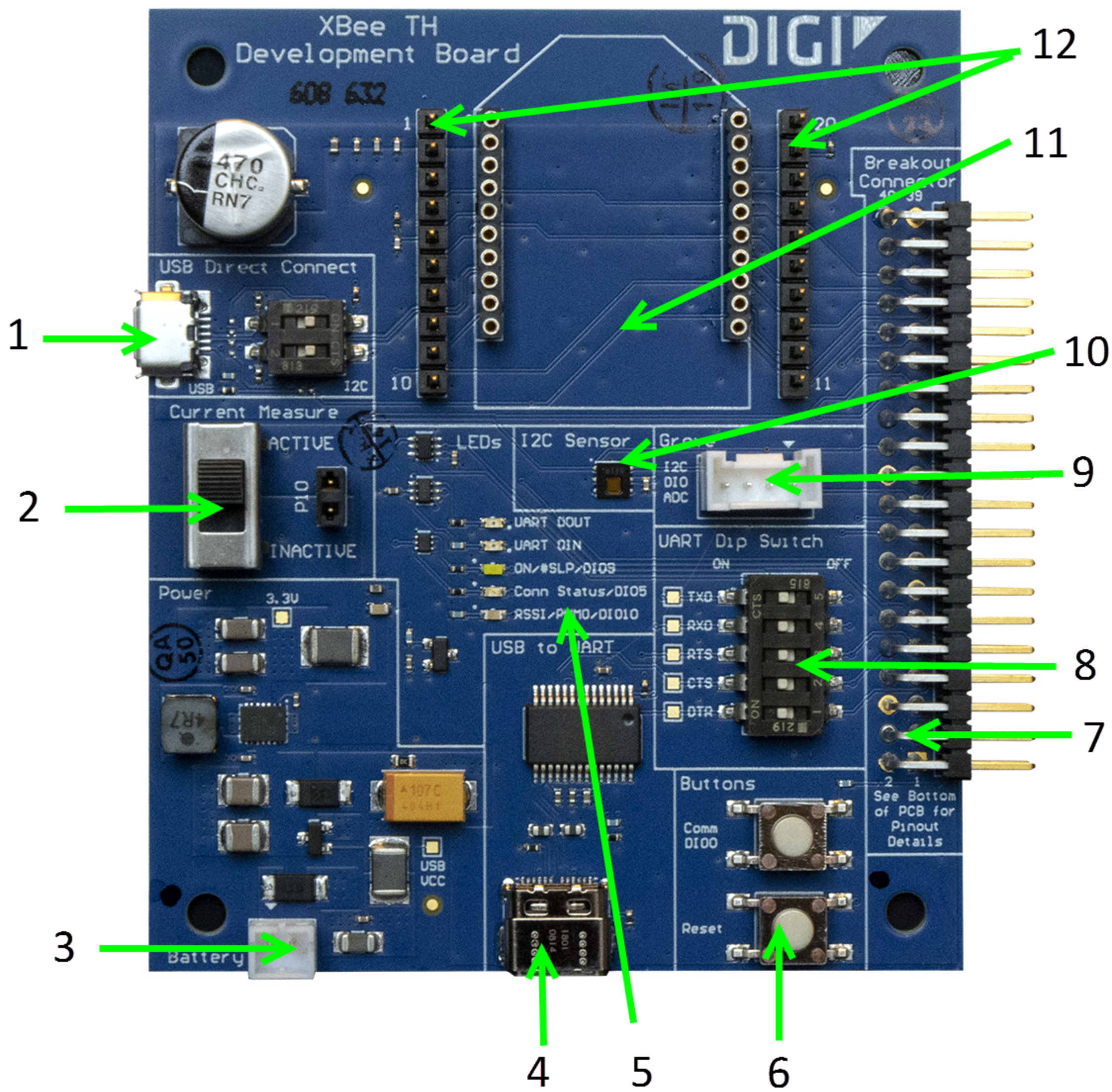


Number	Item	Description
1	Secondary USB (USB MICRO B)	Secondary USB Connector for possible future use. Not used.
2	Current Measure	Large switch controls whether current measure mode is active or inactive. When inactive, current can freely flow to the VCC pin of the XBee. When active, the VCC pin of the XBee is disconnected from the 3.3 V line on the dev board. This allows current measurement to be conducted by attaching a current meter across the jumper P10.
3	Battery Connector	If desired, you can attach a battery to provide power to the development board. The voltage can range from 2 V to 5 V. The positive terminal is on the left.
4	USB-C Connector	Connects to your computer. This is connected to a USB to UART conversion chip that has the five UART lines passed to the XBee. The UART Dip Switch can be used to disconnect these UART lines from the XBee.
5	LED indicator	Red: UART DOUT (modem sending serial/UART data to host) Green: UART DIN (modem receiving serial/UART data from host) White: ON/SLP/DIO9 Blue: Connection Status/DIO5 Yellow: RSSI/PWM0/DIO10
6	User Buttons	Comm DIO0 Button connects the Commissioning/DIO0 pin on the XBee Connector through to a 10 Ω resistor to GND when pressed. $\overline{\text{RESET}}$ Button Connects to the $\overline{\text{RESET}}$ pin on the XBee Connector to GND when pressed.
7	Breakout Connector	This 40-pin connector can be used to connect to various XBee pins as shown on the silkscreen on the bottom of the board.
8	UART Dip Switch	This dip switch allows the user to disconnect any of the primary UART lines on the XBee from the USB to UART conversion chip. This allows for testing on the primary UART lines without the USB to UART conversion chip interfering. Push Dip switches to the right to disconnect the USB to UART conversion chip from the XBee.
9	Grove Connector	This connector can be used to attach I2C enabled devices to the development board. Note that I2C needs to be available on the XBee in the board to use this functionality. Pin 1: I2C_CLK/XBee DIO1 Pin2: I2C_SDA/XBee DIO11 Pin3: VCC Pin4: GND
10	Temp/Humidity Sensor	This as a Texas Instruments HDC1080 temperature and humidity sensor. This part is accessible through I2C. Be sure that the XBee that is inserted into the Dev Board has I2C if access to this sensor is desired.
11	XBee Socket	This is the socket for the XBee (SMT form factor)

XBIB-CU TH reference

This picture shows the XBee-CU TH development board and the table that follows explains the callouts in the picture.

Note This board is sold separately.



Number	Item	Description
1	Secondary USB (USB MICRO B) and DIP Switch	Secondary USB Connector for direct programming of modules on some XBee units. Flip the Dip switches to the right for I2C access to the board; flip Dip switches to the left to disable I2C access to the board. The USB_P and USB_N lines are always connected to the XBee, regardless of Dip switch setting. This USB port is not designed to power the module or the board. Do not plug in a USB cable here unless the board is already being powered through the main USB-C connector. Do not attach a USB cable here if the Dip switches are pushed to the right. **WARNING!** Direct input of USB lines into XBee units or I2C lines not designed to handle 5V can result in the destruction of the XBee or I2C components. Could cause fire or serious injury. Do not plug in a USB cable here if the XBee device is not designed for it and do not plug in a USB cable here if the Dip switches are pushed to the right.
2	Current Measure	Large switch controls whether current measure mode is active or inactive. When inactive, current can freely flow to the VCC pin of the XBee. When active, the VCC pin of the XBee is disconnected from the 3.3 V line on the development board. This allows current measurement to be conducted by attaching a current meter across the jumper P10.
3	Battery Connector	If desired, a battery can be attached to provide power to the development board. The voltage can range from 2 V to 5 V. The positive terminal is on the left. If the USB-C connector is connected to a computer, the power will be provided through the USB-C connector and not the battery connector.
4	USB-C Connector	Connects to your computer and provides the power for the development board. This is connected to a USB to UART conversion chip that has the five UART lines passed to the XBee. The UART Dip Switch can be used to disconnect these UART lines from the XBee.
5	LED indicator	Red: UART DOUT (modem sending serial/UART data to host) Green: UART DIN (modem receiving serial/UART data from host) White: ON/SLP/DIO9 Blue: Connection Status/DIO5 Yellow: RSSI/PWM0/DIO10
6	User Buttons	Comm DIO0 Button connects the Commissioning/DIO0 pin on the XBee Connector through to a 10 Ω resistor to GND when pressed. $\overline{\text{RESET}}$ Button Connects to the $\overline{\text{RESET}}$ pin on the XBee Connector to GND when pressed.

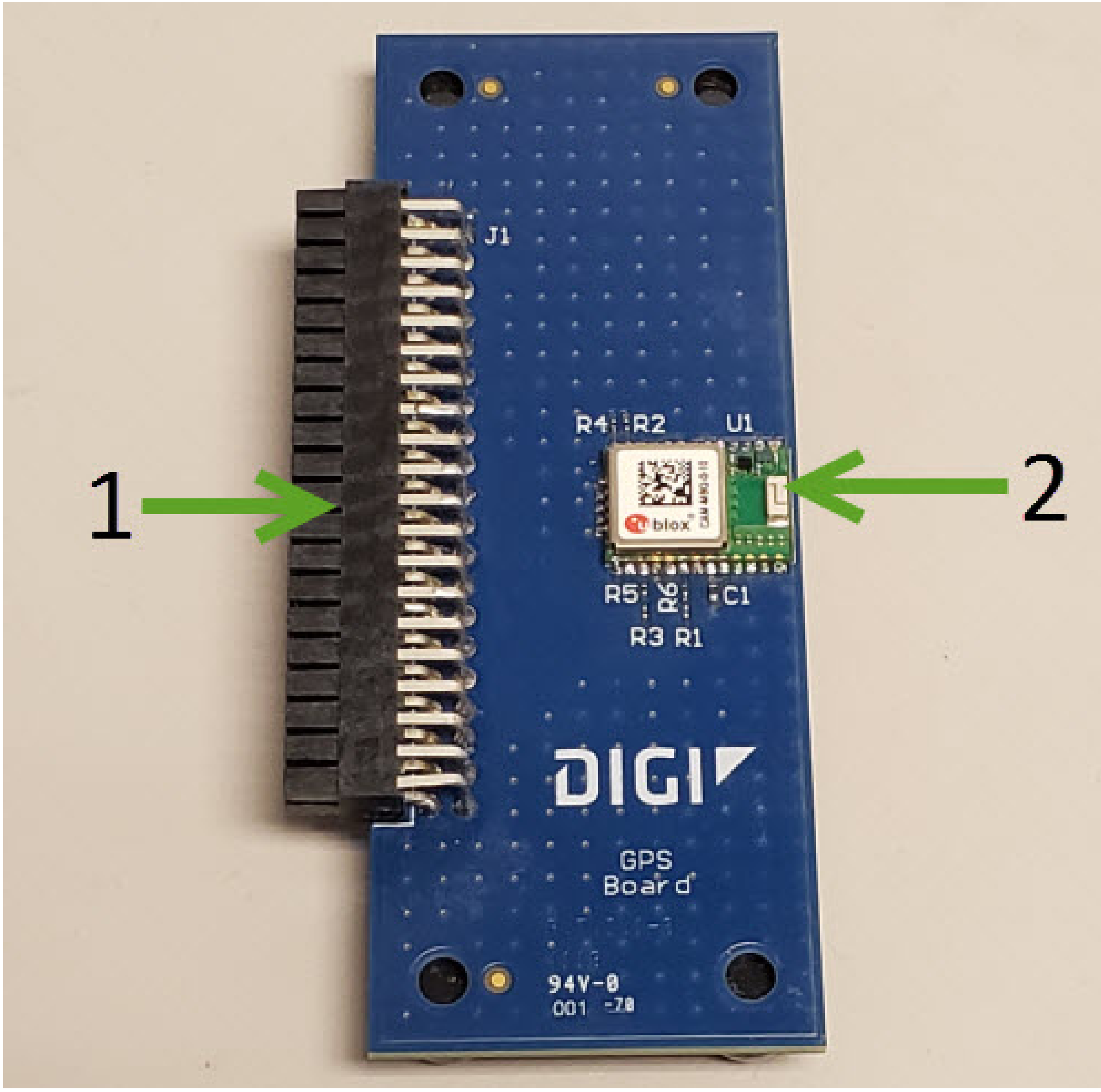
Number	Item	Description
7	Breakout Connector	This 40 pin connector can be used to connect to various XBee pins as shown on the silkscreen on the bottom of the board.
8	UART Dip Switch	This dip switch allows the user to disconnect any of the primary UART lines on the XBee from the USB to UART conversion chip. This allows for testing on the primary UART lines without the USB to UART conversion chip interfering. Push Dip switches to the right to disconnect the USB to UART conversion chip from the XBee.
9	Grove Connector	This connector can be used to attach I2C enabled devices to the development board. Note that I2C needs to be available on the XBee in the board for this functionality to be used. Pin 1: I2C_CLK/XBee DIO1 Pin2: I2C_SDA/XBee DIO11 Pin3: VCC Pin4: GND
10	Temp/Humidity Sensor	This as a Texas Instruments HDC1080 temperature and humidity sensor. This part is accessible through I2C. Be sure that the XBee that is inserted into the development board has I2C if access to this sensor is desired.
11	XBee Socket	This is the socket for the XBee (TH form factor).
12	XBee Test Point Pins	Allows easy access for probes for all 20 XBee TH pins. Pin 1 is shorted to Pin 1 on the XBee and so on.

XBIB-C-GPS reference

This picture shows the XBIB-C-GPS module and the table that follows explains the callouts in the picture.

Note This board is sold separately. You must also have purchased an XBIB-C through-hole, surface-mount, or micro-mount development board.

Note For a demonstration of how to use MicroPython to parse some of the GPS NMEA sentences from the UART, print them and report them to Digi Remote Manager, see Run the MicroPython GPS demo.

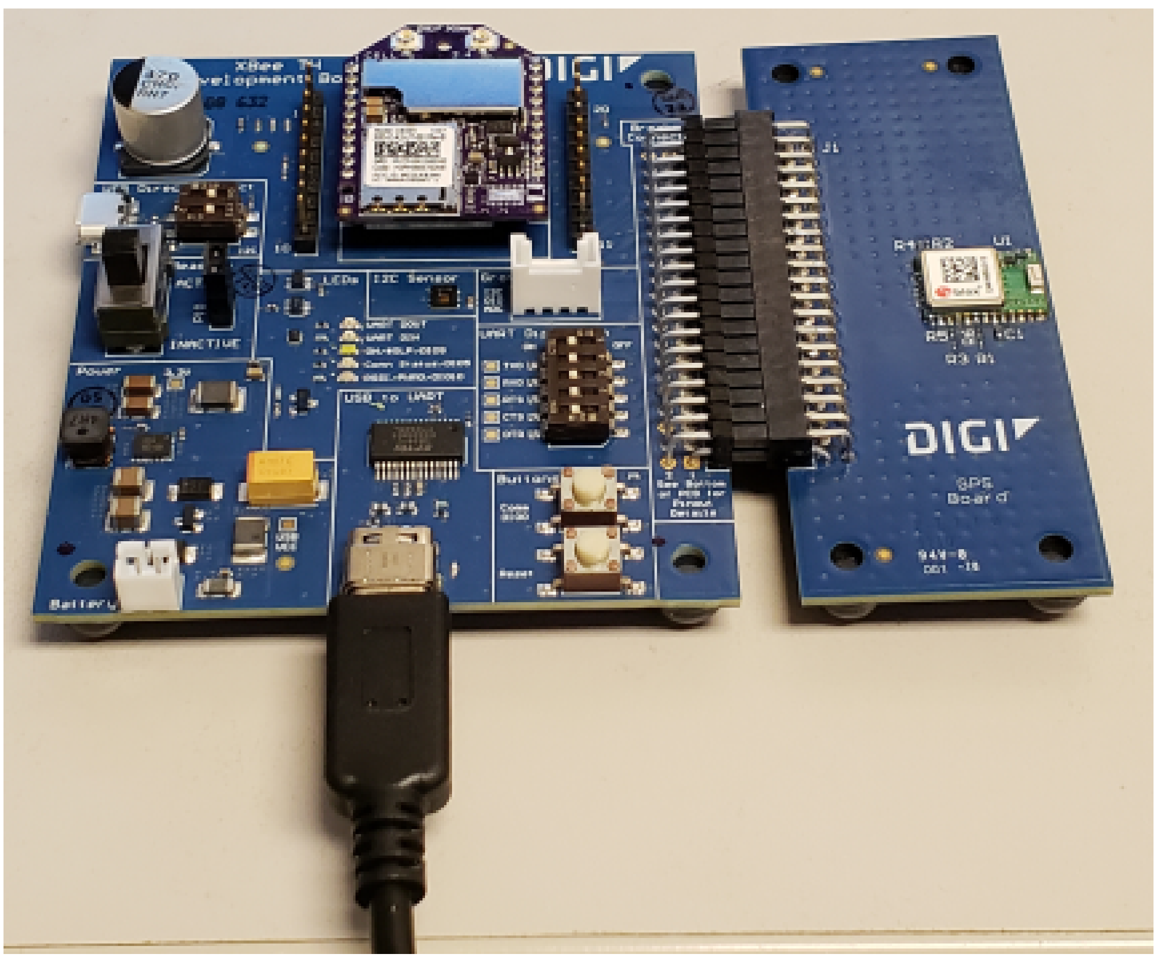

1
2
J1
R4
R2
U1
R5
R6
C1
R3
R1
DIGI
GPS
Board
94V-0

Number	Item	Description
1	40-pin header	This header is used to connect the XBIB-C-GPS board to a compatible XBIB development board. Insert the XBIB-C-GPS module slowly with alternating pressure on the upper and lower parts of the connector. If added or removed improperly, the pins on the attached board could bend out of shape.
2	GPS unit	This is the CAM-M8Q-0-10 module made by u-blox. This is what makes the GPS measurements. Proper orientation is with the board laying completely flat, with the module facing towards the sky.

Interface with the XBIB-C-GPS module

The XBee 3 DigiMesh RF Module can interface with the XBIB-C-GPS board through the large 40-pin header. This header is designed to fit into XBIB-C development board. This allows the XBee 3 DigiMesh RF Module in the XBIB-C board to communicate with the XBIB-C-GPS board—provided the XBee device used has MicroPython capabilities (see this link to determine which devices have MicroPython capabilities). There are two ways to interface with the XBIB-C-GPS board: through the host board's Secondary UART or through the I2C compliant lines.

The following picture shows a typical setup:



I²C communication

There are two I2C lines connected to the host board through the 40-pin header, SCL and SDA. I2C communication is performed over an I2C-compliant Display Data Channel. The XBIB-C-GPS module operates in slave mode. The maximum frequency of the SCL line is 400 kHz. To access data through the I2C lines, the data must be queried by the connected XBee 3 DigiMesh RF Module.

For more information about I2C Operation see the **I2C** section of the *Digi Micro Python Programming Guide*.

For more information on the operation of the XBIB-C-GPS board see the CAM-M8 datasheet. Other CAM-M8 documentation is located here.

UART communication

There are two UART pins connected from the XBIB-C-GPS to the host board by the 40-pin header: RX and TX. By default, the UART on the XBIB-C-GPS board is active and sends GPS readings to the connected device's secondary UART pins. Readings are transmitted once every second. The baud rate of the UART is 9600 baud.

Run the MicroPython GPS demo

The Digi MicroPython github repository contains a GPS demo program that parses some of the GPS NMEA sentences from the UART, prints them and also reports them to Digi Remote Manager.

Note If you are unfamiliar with MicroPython on XBee you should first run some of the tutorials earlier in this manual to familiarize yourself with the environment. See Get started with MicroPython. For more detailed information, refer to the *Digi MicroPython Programming Guide*.

Step 1: Create a Remote Manager developer account

You must have a Remote Manager developer account to be able to use this program. Make sure you know the user name and password for this account.

If you don't currently have a Remote Manager developer account, you can create a free developer account.

Step 2: Download or clone the XBee MicroPython repository

1. Navigate to: **https://github.com/digidotcom/xbee-micropython/**
2. Click **Clone or download**.
3. You must either clone or download a zip file of the repository. You can use either method.
   - **Clone**: If you are familiar with GIT, follow the standard GIT process to clone the repository.
   - **Download**
     a. Click **Download zip** to download a zip file of the repository to the download folder of your choosing.
     b. Extract the repository to a location of your choosing on your hard drive.

Step 3: Edit the MicroPython file

1. Navigate to the location of the repository zip file that you created in Step 2.
2. Navigate to: **samples/gps**
3. Open the MicroPython file: *gpsdemo1.py*
4. Using the editor of your choice, edit the MicroPython file. At the top of the file, enter the user name and password for your Remote Manager developer account. The correct location is indicated in the comments in the file.

Step 4: Run the program

1. Rename the file you edited in Step 3 from *gpsdemo1.py* to *main.py*.
2. Copy the renamed file onto your device's root filesystem directory.
3. Copy the following three modules from the locations specified below into your device's **/lib** directory:
   - From the **/lib** directory of the Digi xbee-micropython repository: *urequest.py* and *remotemanager.py*
   - From the **/lib/sensor** directory of the Digi xbee-micropython repository: *hdc1080.py*

   Note These modules are required to be able to run the *gpsdemo1.py*.

4. Open XCTU and use the MicroPython Terminal to run the demo.
5. Type <CTRL>-R from the MicroPython prompt to run the code.

Get started with MicroPython

This user guide provides an overview of how to use MicroPython with the XBee 3 DigiMesh RF Module. For in-depth information and more complex code examples, refer to the *Digi MicroPython Programming Guide*. Continue with this user guide for simple examples to get started using MicroPython on the XBee 3 DigiMesh RF Module.

About MicroPython

MicroPython is an open-source programming language based on Python 3.0, with much of the same syntax and functionality, but modified to fit on small devices with limited hardware resources, such as an XBee 3 DigiMesh RF Module.

For more information about MicroPython, see www.micropython.org.

For more information about Python, see www.python.org.

MicroPython on the XBee 3 DigiMesh RF Module

The XBee 3 DigiMesh RF Module has MicroPython running on the device itself. You can access a MicroPython prompt from the XBee 3 DigiMesh RF Module when you install it in an appropriate development board (XBDB or XBIB), and connect it to a computer via a USB cable.

Note MicroPython is only available through the UART interface and does not work with SPI.

Note MicroPython programming on the device requires firmware version 3002 or newer.

The examples in this user guide assume:

- You have XCTU on your computer. See Configure the device using XCTU.
- You have a serial terminal program installed on your computer. For more information, see Use the MicroPython Terminal in XCTU. This requires XCTU 6.3.10 or higher.
- You have an XBee 3 DigiMesh RF Module installed on an appropriate development board such as an XBIB-U-DEV or an XBDB-U-ZB.
- The XBee 3 DigiMesh RF Module is connected to the computer via a USB cable and XCTU recognizes it.

Use XCTU to enter the MicroPython environment

To use the XBee 3 DigiMesh RF Module in the MicroPython environment:

1. Use XCTU to add the device(s); see Configure the device using XCTU and Add devices to XCTU.
2. The XBee 3 DigiMesh RF Module appears as a box in the **Radio Modules** information panel. Each module displays identifying information about itself.
3. Click this box to select the device and load its current settings.

   Note To ensure that MicroPython is responsive to input, Digi recommends setting the XBee UART baud rate to 115200 baud. To set the UART baud rate, select **115200 [7]** in the **BD** field and click the **Write** button. We strongly recommend using hardware flow control to avoid data loss, especially when pasting large amounts of code or text. For more information, see UART flow control.

4. To put the XBee 3 DigiMesh RF Module into MicroPython mode, in the **AP** field select **MicroPython REPL [4]** and click the **Write** button.
5. Note which COM port the XBee 3 DigiMesh RF Module is using, because you will need this information when you use the MicroPython terminal.

Use the MicroPython Terminal in XCTU

You can use the MicroPython Terminal to communicate with the XBee 3 DigiMesh RF Module when it is in MicroPython mode.[1] This requires XCTU 6.3.10 or higher. To enter MicroPython mode, follow the steps in Use XCTU to enter the MicroPython environment. To use the MicroPython Terminal:

1. Click the **Tools** drop-down menu and select **MicroPython Terminal**. The terminal window opens.
2. Click **Open** to open the Serial Port Configuration window.
3. In the **Select the Serial/USB port** area, click the COM port that the device uses.
4. Verify that the baud rate and other settings are correct.
5. Click **OK**. The **Open** icon changes to **Close**, indicating that the device is properly connected.

If the **>>>** prompt appears, you are connected properly. You can now type or paste MicroPython code in the terminal.

MicroPython examples

This section provides examples of how to use some of the basic functionality of MicroPython with the XBee 3 DigiMesh RF Module.

Example: hello world

1. At the MicroPython **>>>** prompt, type the Python command: **print("Hello, World!")**
2. Press **Enter** to execute the command. The terminal echos back **Hello, World!**

Example: enter MicroPython paste mode

In the following examples it is helpful to know that MicroPython supports paste mode, where you can copy a large block of code from this user guide and paste it instead of typing it character by character. To use paste mode:

1. Copy the code you want to run. For example, copy the following code that is the code from the "Hello world" example:

```
print("Hello World")
```

Note You can easily copy and paste code from the online version of this guide. Use caution with the PDF version, as it may not maintain essential indentations.

2. In the terminal, at the MicroPython **>>>** prompt type **Ctrl**-+**E** to enter paste mode. The terminal displays **paste mode; Ctrl-C to cancel, Ctrl-D to finish**.
3. Right-click in the MicroPython terminal window and click **Paste** or press **Ctrl+Shift+V** to paste.
4. The code appears in the terminal occupying one line. Each line starts with its line number and three "**=**" symbols. For example, line 1 starts with **1===**.

[1]See Other terminal programs if you do not use the MicroPython Terminal in XCTU.

5. If the code is correct, press **Ctrl**+**D** to run the code; “Hello World” should print.

   Note If you want to exit paste mode without running the code, or if the code did not copy correctly, press **Ctrl**+**C** to cancel and return to the normal MicroPython **>>>** prompt).

Example: use the time module

The time module is used for time-sensitive operations such as introducing a delay in your routine or a timer.

The following time functions are supported by the XBee 3 DigiMesh RF Module:

- **ticks_ms()** returns the current millisecond counter value. This counter rolls over at 0x40000000.
- **ticks_diff()** compares the difference between two timestamps in milliseconds.
- **sleep()** delays operation for a set number of seconds.
- **sleep_ms()** delays operation for a set number of milliseconds.
- **sleep_us()** delays operation for a set number of microseconds.

Note The standard **time.time()** function cannot be used, because this function produces the number of seconds since the epoch. The XBee3 module lacks a realtime clock and cannot provide any date or time data.

The following example exercises the various sleep functions and uses **ticks_diff()** to measure duration:

```
import time

start = time.ticks_ms()  # Get the value from the millisecond counter

time.sleep(1)            # sleep for 1 second
time.sleep_ms(500)       # sleep for 500 milliseconds
time.sleep_us(1000)      # sleep for 1000 microseconds

delta = time.ticks_diff(time.ticks_ms(), start)

print("Operation took {} ms to execute".format(delta))
```

Example: AT commands using MicroPython

AT commands control the XBee 3 DigiMesh RF Module. The "AT" is an abbreviation for "attention", and the prefix "AT" notifies the module about the start of a command line. For a list of AT commands that can be used on the XBee 3 DigiMesh RF Module, see AT commands.

MicroPython provides an **atcmd()** method to process AT commands, similar to how you can use Command mode or API frames.

The **atcmd()** method accepts two parameters:

1. The two character AT command, entered as a string.
2. An optional second parameter used to set the AT command value. If this parameter is not provided, the AT command is queried instead of being set. This value is an integer, bytes object, or string, depending on the AT command.

Note The **xbee.atcmd()** method does not support the following AT commands: **IS**, **AS**, **ED**, **ND**, or **DN**.

The following is example code that queries and sets a variety of AT commands using **xbee.atcmd()**:

```
import xbee

# Set the NI string of the radio
xbee.atcmd("NI", "XBee3 module")

# Configure a destination address using two different data types
xbee.atcmd("DH", 0x0013A200)          # Hex
xbee.atcmd("DL", b'\x12\x25\x89\xF5')  # Bytes

# Read some AT commands and display the value and data type:
print("\nAT command parameter values:")
commands =["DH", "DL", "NI", "CK"]
for cmd in commands:
       val = xbee.atcmd(cmd)
       print("{}: {:20} of type {}".format(cmd, repr(val), type(val)))
```

This example code outputs the following:

```
AT command parameter values:
DH: b'\x00\x13\xa2\x00'  of type <class 'bytes'>
DL: b'\x12%\x89\xf5'     of type <class 'bytes'>
NI: 'XBee3 module'       of type <class 'str'>
CK: 65535                of type <class 'int'>
```

Note Parameters that store values larger than 16-bits in length are represented as bytes. Python attempts to print out ASCII characters whenever possible, which can result in some unexpected output (such as the "%" in the above output). If you want the output from MicroPython to match XCTU, you can use the following example to convert bytes to hex:

```
dl_value = xbee.atcmd("DL")
hex_dl_value = hex(int.from_bytes(dl_value, 'big'))
```

MicroPython networking and communication examples

This section provides networking and communication examples for using MicroPython with the XBee 3 DigiMesh RF Module.

DigiMesh networks with MicroPython

For small networks, it is suitable to use MicroPython on every node. However, there are some inherit limitations that may prevent you from using MicroPython on some heavily trafficked nodes:

- When running MicroPython, any received messages will be stored in a small receive queue. This queue only has room for 4 packets and must be regularly read to prevent data loss. For networks that will be generating a lot of traffic, the data aggregator may need to operate in API mode in order to capture all incoming data.

For the examples in this section, the devices should be pre-configured with identical network settings so that RF communication is possible. To follow the upcoming examples, we need to configure a second XBee 3 DigiMesh RF Module to use MicroPython.

XCTU only allows a single MicroPython terminal. We will be running example code on both modules, which requires a second terminal window.

Open a second instance of XCTU, and configure a different XBee 3 device for MicroPython following the steps in Use XCTU to enter the MicroPython environment.

Example: network Discovery using MicroPython

The **xbee.discover()** method returns an iterator that blocks while waiting for results, similar to executing an **ND** request. For more information, see ND (Network Discover).

Each result is a dictionary with fields based on an **ND** response:

- **sender_nwk**: 16-bit network address.
- **sender_eui64**: 8-byte bytes object with EUI-64 address.
- **parent_nwk**: Set to 0xFFFE on the coordinator and routers; otherwise, this is set to the network address of the end device's parent.
- **node_id**: The device's **NI** value (a string of up to 20 characters, also referred to as Node Identification).
- **node_type**: Value of 0, 1 or 2 for coordinator, router, or end device.
- **device_type**: The device's 32-bit **DD** value, also referred to as Digi Device Type; this is used to identify different types of devices or hardware.
- **rssi**: Relative signal strength indicator (in dBm) of the node discovery request packet received by the sending node.

Note When printing the dictionary, fields for **device_type**, **sender_nwk** and **parent_nwk** appear in decimal form. You can use the MicroPython **hex()** method to print an integer in hexadecimal. Check the function code for **format_eui64** from the Example: communication between two XBee 3 DigiMesh modules topic for code to convert the **sender_eui64** field into a hexadecimal string with a colon between each byte value.

Use the following example code to perform a network discovery:

```
import xbee, time

# Set the network discovery options to include self
xbee.atcmd("NO", 2)
xbee.atcmd("AC")
time.sleep(.5)

# Perform Network Discovery and print out the results
print ("Network Discovery in process...")
nodes = list(xbee.discover())
if nodes:
        for node in nodes:
                print("\nRadio discovered:")
                for key, value in node.items():
                        print("\t{:<12} : {}".format(key, value))

# Set NO back to the default value
xbee.atcmd("NO", 0)
xbee.atcmd("AC")
```

This produces the following output from two discovered nodes:

```
Radio discovered:
        rssi            : -63
        node_id         : Coordinator
```

```
        device_type    : 1179648
        parent_nwk     : 65534
        sender_nwk     : 0
        sender_eui64   : b'\x00\x13\xa2\xff h\x98T'
        node_type      : 0

Radio discovered:
        rssi           : -75
        node_id        : Router
        device_type    : 1179648
        parent_nwk     : 65534
        sender_nwk     : 23125
        sender_eui64   : b'\x00\x13\xa2\xffh\x98c&'
        node_type      : 1
```

Examples: transmitting data

This section provides examples for transmitting data using MicroPython. These examples assume you have followed the above examples and the two radios are on the same network.

Example: transmit message

Use the **xbee** module to transmit a message from the XBee 3 Zigbee device. The **transmit()** function call consists of the following parameters:

1. The Destination Address, which can be any of the following:
   - Integer for 16-bit addressing
   - 8-byte bytes object for 64-bit addressing
   - Constant **xbee.ADDR_BROADCAST** to indicate a broadcast destination
   - Constant **xbee.ADDR_COORDINATOR** to indicate the coordinator
2. The Message as a character string.

If the message is sent successfully, **transmit()** returns **None**. If the transmission fails due to an ACK failure or lack of free buffer space on the receiver, the sent packet will be silently discarded.

Example: transmit a message to the network coordinator

1. From the router, access the MicroPython environment.
2. At the MicroPython >>> prompt, type **import xbee** and press **Enter**.
3. At the MicroPython >>> prompt, type **xbee.transmit(xbee.ADDR_COORDINATOR, "Hello World!")** and press **Enter**.
4. On the coordinator, you can issue an **xbee.receive()** call to output the received packet.

Example: transmit custom messages to all nodes in a network

This program performs a network discovery and sends the message **'Hello <Destination Node Identifier>!'** to individual nodes in the network. For more information, see Example: network Discovery using MicroPython.

```
import xbee

# Perform a network discovery to gather destination address:
print("Discovering remote nodes, please wait...")
node_list = list(xbee.discover())
if not node_list:
```

```
        raise Exception("Network discovery did not find any remote devices")

for node in node_list:
        dest_addr = node['sender_eui64']
        dest_node_id = node['node_id']
        payload_data = "Hello, " + dest_node_id + "!"

        try:
                print("Sending \"{}\" to {}".format(payload_data, dest_addr))
                xbee.transmit(dest_addr, payload_data)
        except Exception as err:
                print(err)

print("complete")
```

Receiving data

Use the **receive()** function from the **xbee** module to receive messages. When MicroPython is active on a device (**AP** is set to 4), all incoming messages are saved to a receive queue within MicroPython. This receive queue is limited in size and only has room for 4 messages. To ensure that data is not lost, it is important to continuously iterate through the receive queue and process any of the packets within.

If the receive queue is full and another message is sent to the device, it will not acknowledge the packet and the sender generates a failure status of 0x24 (Address not found).

The **receive()** function returns one of the following:

- None: No message (the receive queue is empty).
- Message dictionary consisting of:
  - **sender_eui64**: 64-bit address (as a "bytes object") of the sending node.
  - **source_ep**: source endpoint as an integer.
  - **dest_ep**: destination endpoint as an integer.
  - **cluster**: cluster id as an integer.
  - **profile**: profile id as an integer.
  - **broadcast**: True or False depending on whether the frame was broadcast or unicast.
  - **payload**: "Bytes object" of the payload. This is a bytes object instead of a string, because the payload can contain binary data.

Example: continuously receive data

In this example, the **format_packet()** helper formats the contents of the dictionary and **format_eui64 ()** formats the bytes object holding the EUI-64. The **while** loop shows how to poll for packets continually to ensure that the receive buffer does not become full.

```
def format_eui64(addr):
    return ':'.join('%02x' % b for b in addr)

def format_packet(p):
    type = 'Broadcast' if p['broadcast'] else 'Unicast'
    print("%s message from EUI-64 %s (network 0x%04X)" % (type,
        format_eui64(p['sender_eui64']), p['sender_nwk']))
    print("    from EP 0x%02X to EP 0x%02X, Cluster 0x%04X, Profile 0x%04X:" %
        (p['source_ep'], p['dest_ep'], p['cluster'], p['profile']))
    print(p['payload'])
```

```
import xbee, time
while True:
        print("Receiving data...")
        print("Press CTRL+C to cancel.")
        p = xbee.receive()
        if p:
               format_packet(p)
        else:
               time.sleep(0.25)   # wait 0.25 seconds before checking again
```

If this node had previously received a packet, it outputs as follows:

```
Unicast message from EUI-64 00:13:a2:00:41:74:ca:70 (network 0x6D81)
    from EP 0xE8 to EP 0xE8, Cluster 0x0011, Profile 0xC105:
b'Hello World!'
```

Note Digi recommends calling the **receive()** function in a loop so no data is lost. On modules where there is a high volume of network traffic, there could be data lost if the messages are not pulled from the queue fast enough.

Example: communication between two XBee 3 DigiMesh modules

This example combines all of the previous examples and represents a full application that configures a network, discovers remote nodes, and sends and receives messages.

First, we will upload some utility functions into the flash space of MicroPython so that the following examples will be easier to read.

Complete the following steps to compile and execute utility functions using flash mode on both devices:

1. Access the MicroPython environment.
2. Press **Ctrl + F**.
3. Copy the following code:

```
import xbee, time
# Utility functions to perform XBee 3 DigiMesh operations
def format_eui64(addr):
       return ':'.join('%02x' % b for b in addr)

def format_packet(p):
       type = 'Broadcast' if p['broadcast'] else 'Unicast'
       print("%s message from EUI-64 %s" %
               (type, format_eui64(p['sender_eui64'])))
       print("from EP 0x%02X to EP 0x%02X, Cluster 0x%04X, Profile 0x%04X:" %
               (p['source_ep'], p['dest_ep'], p['cluster'], p['profile']))
       print(p['payload'],"\n")
```

4. At the MicroPython 1^^^ prompt, right-click and select the **Paste** option.
5. Press **Ctrl+D** to finish. The code is uploaded to the flash memory and then compiled. At the "Automatically run this code at startup" [Y/N]?" prompt, select **Y**.
6. Press **Ctrl+R** to run the compiled code; this provides access to these utility functions for the next examples.

WARNING! MicroPython code stored in flash is saved in the file system as **main.py**. If the file system has not been formatted, then the following error is generated:
OSError: [Errno 7019] ENODEV
The file system can be formatted in one of three ways:
In XCTU by using the File System Manager.
In Command mode using the **ATFS FORMAT confirm** command—see FS (File System).
In MicroPython by issuing the following code:

```
import os
   os.format()
```

Example code on the aggregator module

The following example code configures DigiMesh network settings, performs a network discovery to find the remote node, and continuously prints out any incoming data.

1. Access the MicroPython environment.
2. Copy the following sample code:

```
print("Configuring DigiMesh network settings...")
xbee.atcmd("NI", "Aggregator")
network_settings = {"CH": 0x13, "ID": 0x1111, "EE": 0}
for command, value in network_settings.items():
       xbee.atcmd(command, value)
xbee.atcmd("AC")  # Apply changes
time.sleep(1)


print("Waiting for a remote node to join...")
node_list = []
while len(node_list) == 0:
       # Perform a network discovery until the router joins
       node_list = list(xbee.discover())
print("Remote node found, transmitting data")

for node in node_list:
       dest_addr = node['sender_eui64'] # using 64-bit addressing
       dest_node_id = node['node_id']
       payload_data = "Hello, " + dest_node_id + "!"

       print("Sending \"{}\" to {}".format(payload_data, repr(dest_addr)))
       xbee.transmit(dest_addr, payload_data)

# Start the receive loop
print("Receiving data...")
print("Hit CTRL+C to cancel")
while True:
       p = xbee.receive()
       if p:
               format_packet(p)
       else:
               time.sleep(0.25)
```

3. Press **Ctrl+E** to enter paste mode.
4. At the **MicroPython >>>** prompt, right-click and select the **Paste** option. Once you paste the code, it executes immediately.

Example code on the router module

The following example code joins the Zigbee network from the previous example, and continuously prints out any incoming data. This device also sends its temperature data every 5 seconds to the coordinator address.

1. Access the MicroPython environment.
2. Copy the following sample code:

```
print("Configuring network settings...")
xbee.atcmd("NI", "Remote")
network_settings = {"CH": 0x13, "ID": 0x1111, "EE": 0}
for command, value in network_settings.items():
       xbee.atcmd(command, value)
xbee.atcmd("AC")  # Apply changes
time.sleep(1)


print("Network configured\n")

last_sent = time.ticks_ms()
interval = 5000  # How often to send a message

# Start the transmit/receive loop
print("Sending temp data every {} seconds".format(interval/1000))
while True:
       p = xbee.receive()
       if p:
               format_packet(p)
       else:
               # Transmit temperature if ready
               if time.ticks_diff(time.ticks_ms(), last_sent) > interval:
                               temp = "Temperature: {}C".format(xbee.atcmd("TP"))
                               print("\tsending " + temp)
                               try:
                                       xbee.transmit(xbee.ADDR_BROADCAST, temp)
                               except Exception as err:
                                       print(err)
                               last_sent = time.ticks_ms()
               time.sleep(0.25)
```

3. Press **Ctrl+E** to enter paste mode.
4. At the **MicroPython >>>** prompt, right-click and select the **Paste** option. Once you paste the code, it executes immediately.

Exit MicroPython mode

To exit MicroPython mode:

1. In the XCTU MicroPython terminal, click the green **Close** button .
2. Click **Close** at the bottom of the terminal to exit the terminal.

3. In XCTU's Configuration working mode, change **AP API Enable** to another mode and click the **Write** button. We recommend changing to Transparent mode **[0]**, as most of the examples use this mode.

Other terminal programs

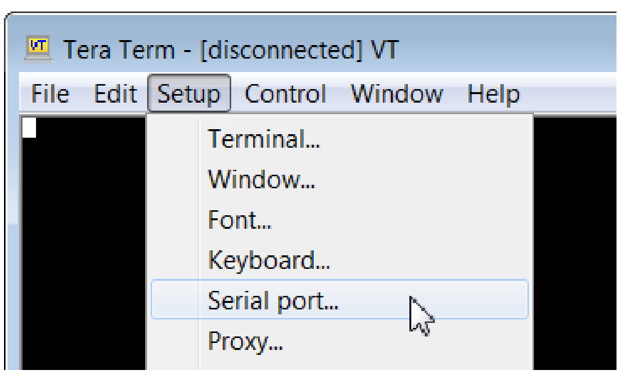
If you do not use the MicroPython terminal in XCTU, you can use other terminal programs to communicate with the XBee 3 DigiMesh RF Module. If you use Microsoft Windows, follow the instructions for Tera Term; if you use Linux, follow the instructions for picocom. To download these programs:

- Tera Term for Windows, see ttssh2.osdn.jp/index.html.en.
- Picocom for Linux, see developer.ridgerun.com/wiki/index.php/Setting_up_Picocom_-_Ubuntu
- Source code and in-depth information, see github.com/npat-efault/picocom.

Tera Term for Windows

With the XBee 3 DigiMesh RF Module in MicroPython mode (**AP** = **4**), you can access the MicroPython prompt using a terminal.

1. Open Tera Term. The **Tera Term: New connection** window appears.
2. Click the **Serial** radio button to select a serial connection.
3. From the **Port:** drop-down menu, select the COM port that the XBee 3 DigiMesh RF Module is connected to.
4. Click **OK**. The **COMxx - Tera Term VT** terminal window appears and Tera Term attempts to connect to the device at a baud rate of 9600 bps. The terminal will not allow communication with the device since the baud rate setting is incorrect. You must change this rate as it was previously set to 115200 bps.
5. Click **Setup** and **Serial Port**. The **Tera Term: Serial port setup** window appears.



6. In the **Tera Term: Serial port setup** window, set the parameters to the following values:
   - **Port**: Shows the port that the XBee 3 DigiMesh RF Module is connected on.
   - **Baud rate**: 115200
   - **Data**: 8 bit
   - **Parity**: none

- **Stop**: 1 bit
- **Flow control**: hardware
- **Transmit delay**: N/A

7. Click **OK** to apply the changes to the serial port settings. The settings should go into effect right away.
8. To verify that local echo is not enabled and that extra line-feeds are not enabled:
   a. In Tera Term, click **Setup** and select **Terminal**.
   b. In the **New-line** area of the **Tera Term: Serial port setup** window, click the **Receive** drop-down menu and select **AUTO** if it does not already show that value.
   c. Make sure the **Local echo** box is not checked.
9. Click **OK**.
10. Press **Ctrl+B** to get the MicroPython version banner and prompt.

```
MicroPython v1.9.3-716-g507d0512 on 2018-02-20; XBee3 DigiMesh with
EFR32MG
Type "help()" for more information.
>>>
```

Now you can type MicroPython commands at the **>>>** prompt.

Use picocom in Linux

With the XBee 3 DigiMesh RF Module in MicroPython mode (**AP** = **4**), you can access the MicroPython prompt using a terminal.

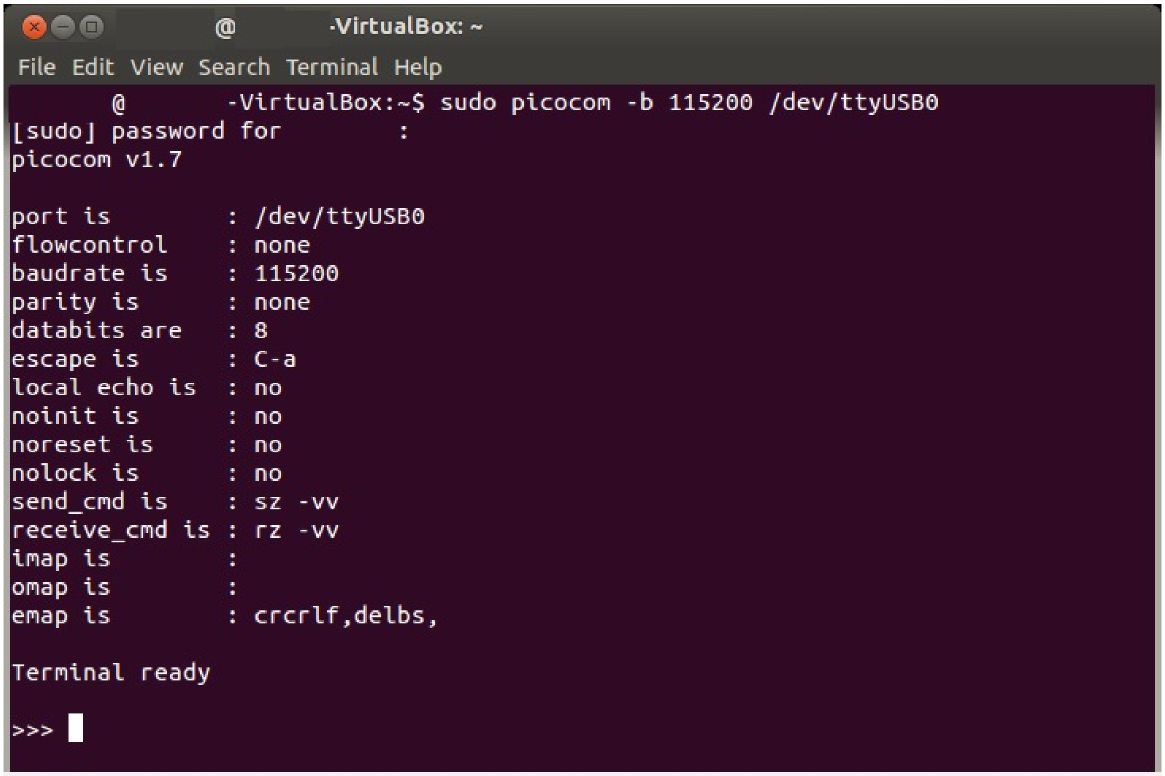
Note The user must have read and write permission for the serial port the XBee 3 DigiMesh RF Module is connected to in order to communicate with the device.

1. Open a terminal in Linux and type **picocom -b 115200 /dev/ttyUSB0**. This assumes you have no other USB-to-serial devices attached to the system.
2. Press **Ctrl+B** to get the MicroPython version banner and prompt. You can also press **Enter** to bring up the prompt.

If you do have other USB-to-serial devices attached:

1. Before attaching the XBee 3 DigiMesh RF Module, check the directory **/dev/** for any devices named **ttyUSBx**, where **x** is a number. An easy way to list these is to type: **ls /dev/ttyUSB***. This produces a list of any device with a name that starts with **ttyUSB**.
2. Take note of the devices present with that name, and then connect the XBee 3 DigiMesh RF Module.
3. Check the directory again and you should see one additional device, which is the XBee 3 DigiMesh RF Module.
4. In this case, replace **/dev/ttyUSB0** at the top with **/dev/ttyUSB<number>**, where **<number>** is the new number that appeared.

   It connects and shows "Terminal ready".



You can now type MicroPython commands at the **>>>** prompt.

Micropython help ()

When you type the **help()** command at the prompt, it provides a link to online help, control commands and also usage examples.

```
>>> help()
Welcome to MicroPython!
For online docs please visit http://docs.micropython.org/.
Control commands:
CTRL-A        -- on a blank line, enter raw REPL mode
CTRL-B        -- on a blank line, enter normal REPL mode
CTRL-C        -- interrupt a running program
CTRL-D        -- on a blank line, reset the REPL
CTRL-E        -- on a blank line, enter paste mode
CTRL-F        -- on a blank line, enter flash upload mode
For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')
-------------------------------------------------------
------------------------------------------------------
```

When you type **help('modules')** at the prompt, it displays all available Micropython modules.

```
------------------------------------------------------
>>> help('modules')
__main__            io                    time               uos
array               json                  ubinascii          ustruct
binascii            machine               uerrno             utime
```

```
builtins            micropython          uhashlib            xbee
errno               os                   uio
gc                  struct               ujson
hashlib             sys                  umachine

Plus any modules on the filesystem

------------------------------------------------------
```

When you import a module and type **help()** with the module as the object, you can query all the functions that the object supports.

```
------------------------------------------------------
>>> import sys
>>> help(sys)
object <module 'sys'> is of type module
  __name__ -- sys
  path -- ['', '/flash', '/flash/lib']
  argv -- ['']
  version -- 3.4.0
  version_info -- (3, 4, 0)
  implementation -- ('micropython', (1, 10, 0))
  platform -- xbee3-DigiMesh
  byteorder -- little
  maxsize -- 2147483647
  exit -- <function>
  stdin -- <io.FileIO 0>
  stdout -- <io.FileIO 1>
  stderr -- <io.FileIO 2>
  modules -- {}
  print_exception -- <function>
-------------------------------------------------------
```

Secure access

By default, the XBee 3 DigiMesh RF Module is easy to configure and allows for rapid prototyping. For deployment, you can encrypt networks to prevent unauthorized access. This can prevent entities outside of the network from accessing data on that network. Some customers may also desire a way to restrict communication between nodes from inside the same network.

There are three ways to secure your device against unauthorized access:

- Secure remote session
- Disable functionality

Secure session protects against external man-in-the middle attacks by requiring remote devices to authenticate before they are allowed to make configuration changes.

You can also disable device functionality in order to prevent unexpected malicious use of the product. For example disable MicroPython so that remote code cannot be uploaded and executed.

Secure Sessions

Secure Sessions provide a way to password-protect communication between two nodes on a network above and beyond the security of the network itself. With secure sessions, a device can 'log in', or create a session with another device that is encrypted and only readable by the two nodes involved. By restricting certain actions—such as remote AT commands or FOTA updates—to only be allowed over one of these secure sessions, you can make it so access to the network does not allow network configuration. A password must be set and the proper bits of SA (Secure Access) must be set to enable this feature.

The following definitions relate to secure Sessions:

Term	Definition
Client	The device that is attempting to log in and send secured data or commands is called the client.
Server	The device that is being logged into and will receive secured data or commands is called the server.
Secure Session	A secure connection between a server and a client where the pair can send and receive encrypted data that only they can decrypt.
Secure Remote Password (SRP)	Name of the authentication protocol used to create the secure connection between the nodes.
Salt	A random value generated as part of the authentication process.
Verifier	A value derived from a given salt and password.

Configure the secure session password for a device

For a device to act as a secure session server it needs to have a password configured. The password is configured on the server in the form of a salt and verifier used for the SRP authentication process. The salt and verifier can be configured in XCTU by selecting the **Secure Session Authentication** option.

We recommend using XCTU to set a password which will then generate the salt and verifier parameters, although the salt and verifier values can also be set manually. See *S (Secure Session Salt) and *V, *W, *X, *Y (Secure Session Verifier) for more information.

Note There is not an enforced password length. We recommend a minimum length of at least eight characters. The password should not exceed 64 characters, as it will exceed the maximum length of an API frame.

Start a secure session

A secure session can only be started in API mode. Once you have been authenticated you may send data in API mode or Transparent mode, but API mode is the recommended way to communicate.

To start a secure session:

1. Send a type Secure Session Control - 0x2E to your local client device with the address of the server device (not a broadcast address), the options bit field set to **0x00**, the timeout for the session, and the password that was previously set on the server.
2. The client and server devices will send/exchange several packets to authenticate the session.

3. When authentication is complete, the client device will output a Secure Session Response - 0xAE to indicate whether the login was a success or failure.

At this point if authentication was successful, the secure session is established and the client can send secured data to the server until the session times out.

Note A device can have one outgoing session—a session in which the node is a client—at a time. Attempting to start a new session while a session is already in progress automatically ends the previous session.

Note A device can have up to four incoming sessions—sessions in which the device is a server—at a time. Once that number has been reached, additional authentication requests are rejected until one of the active sessions ends.

End a secure session

A client can end a session by either waiting for the timeout to expire or by ending it manually. To end a session, send a Secure Session Control - 0x2E to the local client device with bit 0 of the options field set and with no password.

The device ends the outgoing secure session with the node whose address is specified in the type 0x2E frame. This frame can be sent even if the node does not have a session with the specified address—the device will send a message to the specified server prompting it to clear out any incoming session data related to the client (this can be used if the server and client fall out of sync. For example, if the client device unexpectedly loses power during a session.

Sending a type 0x2E frame with the logout option bit set, and the address field set to the broadcast address will end whatever outgoing session is currently active on the client and broadcast a request to all servers to clear any incoming session data related to that client.

Secured remote AT commands

Secure a node against unauthorized remote configuration

Secured Access is enabled by setting bits of SA (Secure Access). Additionally, an SRP Salt (***S**) and verifier (***V**, ***W**, ***X**, ***Y**) must be set. You can use XCTU to generate the salt and verifier based on a password.

Configure a node with a salt and verifier

In this example, the password is **pickle**.

1. The salt is randomly generated and the verifier is derived from the salt and password as follows:

   ***S = 0x1938438E**

   ***V = 0x0771F57C397AE4019347D36FD1B9D91FA05B2E5D7365A161318E46F72942A45D**

   ***W = 0xD4E44C664B5609C6D2BE3258211A7A20374FA65FC7C82895C6FD0B3399E73770**

   ***X = 0x63018D3FEA59439A9EFAE3CD658873F475EAC94ADF7DC6C2C005b930042A0B74**

   ***Y = 0xAEE84E7A00B74DD2E19E257192EDE6B1D4ED993947DF2996CAE0D644C28E8307**

Note The salt and verifier will not always be the same even if the same password is used to generate them.

2. Enforce secure access for Remote AT Commands by setting Bit 1 of the **SA** command:

   SA = **0x02**

3. Write the configuration to flash using WR (Write).

WARNING! Make sure that this step is completed. If your device resets for any reason and *S and SA are not written to flash they will revert to defaults, rendering the node open to insecure access.

4. From now on, any attempt to issue a Remote AT Command Request - 0x17 to this device will be rejected with a **0x0B** status unless a secure session is established first.

Remotely configure a node that has been secured

In the example above a node is secured against unauthorized remote configuration. In this instance, the secured node acts as a Secure Session Server (remote). The sequence below describes how a Secure Session Client (local) can authenticate and securely configure the server remotely.

Establish a secure session using the password that was set on the server node

1. Generate a Secure Session Control - 0x2E.

- The destination address must match the 64-bit address (**SH** + **SL**) of the remote server.
- Since you are logging in, leave the options field as **0x00**.
- Set a five minute timeout, which should give sufficient time for ad hoc configuration. The units are in tenths of a second, so **0x0BB8** gives you five minutes.
- The options are set for a fixed duration, so after the five minutes expire, both the server and client emit a modem status indicating the session ended.
- Enter the original password used to generate the verifier from the random salt above.

2. Pass the type 0x2E Control frame into the serial interface of the local client:

- For example, to log into a Secure Session server at address **0013A200 417B2162** for a five minute duration using the password **pickle**, use the following frame:
  7E 00 12 2E 00 13 A2 00 41 7B 21 62 00 0B B8 70 69 63 6B 6C 65 A2

3. Wait for a Secure Session Response - 0xAE to indicate the session establishment succeeded or failed with the reason.

- The address of the remote that is responding and the status is included in the response.
- For example, the response to the request above is as follows:
  7E 00 0B AE 00 00 13 A2 00 41 7B 21 62 **00** 5D. The 0**x00** status indicates success.

4. Send remote AT Commands to the remote server using the Remote AT Command Request - 0x17 with bit 4 of the Command Options field set. Bit 4 indicates the AT command should be sent securely.

Send data to a secured remote node

The process to send secured data is very similar to remotely configuring a node. The following steps show how a client node can authenticate with a server node and send data securely.

1. Send a Secure Session Control - 0x2E to the client node with:
   - The server's 64-bit address.
   - The desired timeout.
   - The options field set to **0x00** for fixed timeout login or to **0x04** for inter-packet timeout refresh login.
   - The password of the server node.
2. Wait for the Secure Session Response - 0xAE to determine if the the authentication was successful.
3. Data can now be sent securely with Transmit Request - 0x10 and Explicit Addressing Command Request - 0x11 provided that:
   - Bit 4 in the transmit options field is set to indicate that the data should be sent encrypted.
4. The returned Receive Packet - 0x90 and Explicit Receive Indicator - 0x91 receive options fields should also have bit 4 set.

Note The maximum payload per transmission size is reduced by four bytes due to the additional encryption overhead. NP (Maximum Packet Payload Bytes) will not reflect this change when the session is going on.

A node can be secured against emitting data out the serial port that was received insecurely via the **SA** command. This means that a remote node will not emit any serial data if it was received insecurely (TO (Transmit Options) bit 4 was not set). This includes any data in Transparent mode, **0x80**, **0x90** and **0x91** frames.

Note When a device rejects a data transmission (0x80, 0x90, 0x91, or Transparent data) because of its **SA** configuration, it does not send an error back to the sender. This means that data transmissions to a device give a success status even if they are rejected.

End a session from a server

If bit 3 of AZ (Extended API Options) is set, the server emits an extended modem status (whenever a client establishes a session with it) that includes the 64-bit address of the client. Using these statuses the MCU connected to the server can keep track of sessions established with the server. To end a session from the server do the following:

1. Send a Secure Session Control - 0x2E to the server node with:
   - The client's 64-bit address.
   - The options field set to **0x02** for server side session termination.
   - Set the timeout to **0x0000**.
2. Wait for the Secure Session Response - 0xAE to determine if the termination was successful.
   - The client will emit a modem status **0x3C** (Session Ended).
   - The server will also emit a modem status (or an extended modem status depending on **AZ**) of **0x3C** (Session Ended).

Note The 64-bit address can be set to the broadcast address to end all incoming sessions.

Note This functionality can be used to end orphaned client-side sessions—in case the server unexpectedly reset for some reason.

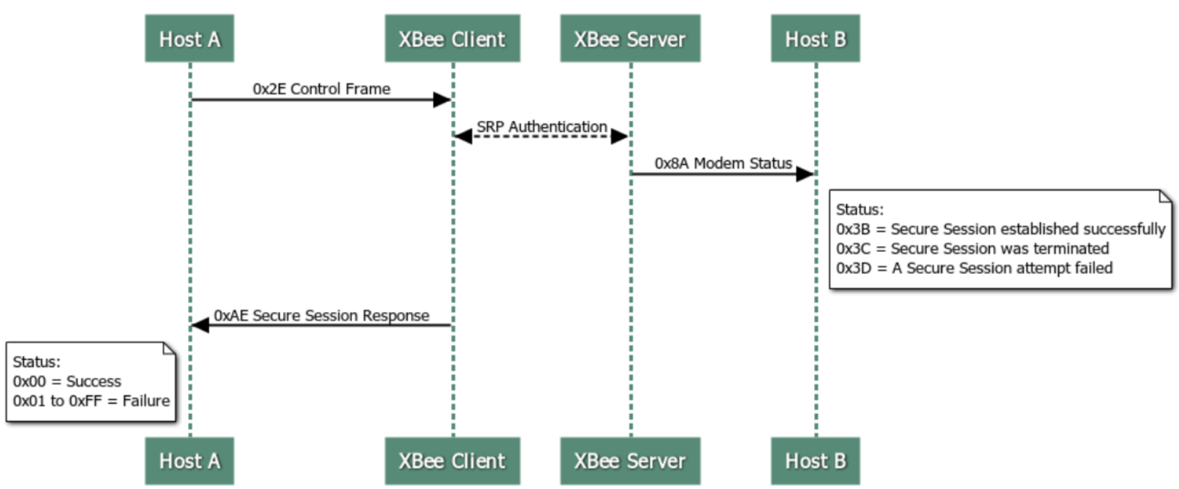
Secure Session API frames

Secure Session can only be established from a node that is operating in API mode. The server-side can be in Transparent mode, but the client must be in API mode. Once a session has been established between a client and server node, the client can be transitioned to Transparent mode; and if bit 4 of **TO** is set, the client will encrypt data sent in Transparent mode for the duration of session.

There are four frames that are used for controlling and observing a secure session.

- Secure Session Control - 0x2E: This frame is passed to the client that wishes to log into or out of a server. Any attempt to use the Control frame will generate a response frame.
- Secure Session Response - 0xAE: This frame returns the status of the previously sent 0x2E frame indicating whether it was successful or not.
- Modem Status - 0x8A: The server will also emit a modem status whenever an attempt succeeds, fails, or was terminated. The client will also emit modem statuses if the session times out.
- Extended Modem Status - 0x98: If bit 3 of **AZ** is set then modem statuses will be replaced with extended modem statuses. These frames will contain the status that caused them to be emitted as well as the address of the node that initiated the session, the session options, and the timeout value.

Frame exchanges:



Secure transmission failures

This section describes the error messages you can see when trying to send a secure packet.

Data Frames - 0x10 and 0x11 frames

- Response frame type: Extended Transmit Status - 0x8B

Possible error statuses:

Status	Description	Reason
0x34	No Secure Session Connection	The sending node does not have an active session with the destination node.
0x35	Encryption Failure	The encryption process failed. Only likely to be seen when using manual SRP and when an invalid encryption parameter was passed in.

Remote AT Commands- 0x17 frames

- Response frame type: Remote AT Command Response- 0x97

Possible error statuses:

Status	Description	Reason
0x0B	No Secure Session Connection	The sending node does not have an active session with the destination node.
0x0C	Encryption Error	There was an internal encryption error on the radio.
0x0D	TO Bit Not Set	The client has a session with the server but forgot to set the **TO** bit.

File system

For detailed information about using MicroPython on the XBee 3 DigiMesh RF Module refer to the *Digi MicroPython Programming Guide*.

Overview of the file system

CAUTION! You need to format the file system if upgrading a device that originally shipped with older firmware. You can use XCTU, AT commands or MicroPython for that initial format or to erase existing content at any time.

Note To use XCTU with file system, you need XCTU 6.4.0 or newer.

See **FS FORMAT confirm** in FS (File System) and ensure that the format is complete.

Directory structure

The XBee 3 DigiMesh RF Module's internal flash appears in the file system as **/flash**, the only entry at the root level of the file system. Files and directories other than **/flash** cannot be created within the root directory, only within **/flash**.

By default **/flash** contains a lib directory intended for MicroPython modules.

Paths

The XBee 3 DigiMesh RF Module stores all of its files in the top-level directory **/flash**. On startup, the **ATFS** commands and MicroPython each use that directory as their current working directory. When specifying the path to a file or directory, it is interpreted as follows:

- Paths starting with a forward slash are "absolute" and must start with **/flash** to be valid.
- All other paths are relative to the current working directory.
- The directory **..** refers to the parent directory, so an operation on **../filename.txt** that takes place in the directory **/flash/test** accesses the file **/flash/filename.txt**.
- The directory **.** refers to the current directory, so **ATFS ls .** is the same as **ATFS ls**, which lists files in the current directory.
- Names are case-insensitive, so **FILE.TXT**, **file.txt** and **FiLe.TxT** all refer to the same file.
- File and directory names are limited to 64 characters, and can only contain letters, numbers, periods, dashes and underscores. A period at the end of the name is ignored.
- The full, absolute path to a file or directory is limited to 255 characters.

Limitations

The file system on the XBee 3 DigiMesh RF Module has a few limitations when compared to conventional file systems:

- When a file on the file system is deleted, the space it was using is only reclaimed if it is found at the end of the file system. Deleted data that is contiguous with the last placed deleted file is also reclaimed.
- The file system can only have one file open for writing at a time.
- The file system cannot create new directories while a file is open for writing.
- Files cannot be renamed.

- The contents of the file system will be lost when any firmware update is performed. See OTA file system upgrades for information on how to put files on a device after a FOTA update.

XCTU interface

XCTU releases starting with 6.4.0 include a **File System Manager** in the **Tools** menu. You can upload files to and download files from the device, in addition to renaming and deleting existing files and directories. See the File System manager tool section of the *XCTU User Guide* for details of its functionality.

Configure the XBee 3 DigiMesh RF Module

Software libraries

One way to communicate with the XBee 3 DigiMesh RF Module is by using a software library. The libraries available for use with the XBee 3 DigiMesh RF Module include:

- XBee Java library
- XBee Python library

The XBee Java Library is a Java API. The package includes the XBee library, its source code and a collection of samples that help you develop Java applications to communicate with your XBee devices.

The XBee Python Library is a Python API that dramatically reduces the time to market of XBee projects developed in Python and facilitates the development of these types of applications, making it an easy process.

Firmware over-the-air (FOTA) update

The XBee 3 DigiMesh RF Module supports FOTA updates using XCTU version 6.3.0 or higher. For instructions on performing a FOTA firmware update with XCTU, see How to update the firmware of your modules in the XCTU User Guide.

Custom defaults

Custom defaults allow you to preserve a subset of the device configuration parameters even after returning to default settings using RE (Restore Defaults). This can be useful for settings that identify the device—such as NI (Network Identifier)—or settings that could make remotely recovering the device difficult if they were reset—such as ID (Network ID).

Note You must send these commands as local AT commands, they cannot be set using Remote AT Command Request - 0x17.

Set custom defaults

Use %F (Set Custom Default) to set custom defaults. When the XBee 3 DigiMesh RF Module receives **%F** it takes the next command it receives and applies it to both the current configuration and the custom defaults.

To set custom defaults for multiple commands, send a **%F** before each command.

Restore factory defaults

!C (Clear Custom Defaults) clears all custom defaults, so that RE (Restore Defaults) will restore the device to factory defaults. Alternatively, R1 (Restore Factory Defaults) restores all parameters to factory defaults without erasing their custom default values.

Limitations

There is a limitation on the number of custom defaults that can be set on a device. The number of defaults that can be set depends on the size of the saved parameters and the devices' firmware version. When there is no more room for custom defaults to be saved, any command sent immediately after a **%F** returns an error.

Custom configuration: Create a new factory default

You can create a custom configuration that is used as a new factory default. This feature is useful if, for example, you need to maintain certain settings for manufacturing or want to ensure a feature is always enabled. When you use RE (Restore Defaults) to perform a factory reset on the device, the custom configuration is set on the device after applying the original factory default settings.

For example, by default Bluetooth is disabled on devices. You can create a custom configuration in which Bluetooth is enabled by default. When you use **RE** to reset the device to the factory defaults, the Bluetooth configuration set to the custom configuration (enabled) rather than the original factory default (disabled).

The custom configuration is stored in non-volatile memory. You can continue to create and save custom configurations until the XBee 3 DigiMesh RF Module's memory runs out of space. If there is no space left to save a configuration, the device returns an error.

You can use !C (Clear Custom Defaults) to clear or overwrite a custom configuration at any time.

Set a custom configuration

1. Open XCTU and load your device.
2. Enter Command mode.
3. Perform the following process for each configuration that you want to set as a factory default.
   a. Send the Set Custom Default command, **AT%F**. This command enables you to enter a custom configuration.
   b. Send the custom configuration command. For example: **ATBT 1**. This command sets the default for Bluetooth to enabled.

Clear all custom configuration on a device

After you have set configurations using %F (Set Custom Default), you can return all configurations to the original factory defaults.

1. Open XCTU and load the device.
2. Enter Command mode.
3. Send **AT!C**.

XBee bootloader

You can update firmware on the XBee 3 DigiMesh RF Module serially. This is done by invoking the XBee bootloader and transferring the firmware image using XMODEM.

This process is also used for updating a local device's firmware using XCTU.

XBee devices use a modified version of Silicon Labs' Gecko bootloader. This bootloader version supports a custom entry mechanism that uses module pins DIN, DTR/SLEEP_RQ, and RTS.

To invoke the bootloader using hardware flow control lines, do the following:

1. Set DTR/SLEEP_RQ low (CMOS0V) and RTS high.
2. Send a serial break to the DIN pin and power cycle or reset the module.
3. When the device powers up, set DTR/SLEEP_RQ and DIN to low (CMOS0V) and RTS should be high.
4. Terminate the serial break and send a carriage return at 115200 baud to the device.

5. If successful, the device sends the Silicon Labs' Gecko bootloader menu out the DOUT pin at 115200 baud.
6. You can send commands to the bootloader at 115200 baud.

Note Disable hardware flow control when entering and communicating with the bootloader.

All serial communications with the module use 8 data bits, no parity bit, and 1 stop bit.

You can also invoke the bootloader from the XBee application by sending %P (Invoke Bootloader).

Send a firmware image

After invoking the bootloader, a menu is sent out the UART at 115200 baud. To upload a firmware image through the UART interface:

1. Look for the bootloader prompt **BL** > to ensure the bootloader is active.
2. Send an ASCII **1** character to initiate a firmware update.
3. After sending a **1**, the device waits for an XModem CRC upload of a .gbl image over the serial line at 115200 baud. Send the .gbl file to the device using standard XMODEM-CRC.

If the firmware image is successfully loaded, the bootloader outputs a “complete” string. Invoke the newly loaded firmware by sending a **2** to the device.

If the firmware image is not successfully loaded, the bootloader outputs an "aborted string". It returns to the main bootloader menu. Some causes for failure are:

- Over 1 minute passes after the command to send the firmware image and the first block of the image has not yet been sent.
- A power cycle or reset event occurs during the firmware load.
- A file error or a flash error occurs during the firmware load. The following table contains errors that could occur during the XMODEM transfer.

Error	Cause	Workaround
0x18	This error is observed when a serial upload attempt has been abruptly discontinued by invoking **Ctrl**+**C** and subsequently another attempt is made to upload a gbl by pressing **1** on the bootloader menu.	Press **2** on the bootloader menu. The bootloader performs a reboot and the menu gets displayed again. Now press **1** and begin uploading the gbl.

XBee Network Assistant

The XBee Network Assistant is an application designed to inspect and manage RF networks created by Digi XBee devices. Features include:

- Join and inspect any nearby XBee network to get detailed information about all the nodes it contains.
- Update the configuration of all the nodes of the network, specific groups, or single devices based on configuration profiles.
- Geo-locate your network devices or place them in custom maps and get information about the connections between them.

- Export the network you are inspecting and import it later to continue working or work offline.
- Use automatic application updates to keep you up to date with the latest version of the tool.

See the *XBee Network Assistant User Guide* for more information.

To install the XBee Network Assistant:

1. Navigate to digi.com/xbeenetworkassistant.
2. Click **General Diagnostics, Utilities and MIBs**.
3. Click the **XBee Network Assistant - Windows x86** link.
4. When the file finishes downloading, run the executable file and follow the steps in the XBee Network Assistant Setup Wizard.

XBee Multi Programmer

The XBee Multi Programmer is a combination of hardware and software that enables partners and distributors to program multiple Digi Radio frequency (RF) devices simultaneously. It provides a fast and easy way to prepare devices for distribution or large networks deployment.

The XBee Multi Programmer board is an enclosed hardware component that allows you to program up to six RF modules thanks to its six external XBee sockets. The XBee Multi Programmer application communicates with the boards and allows you to set up and execute programming sessions. Some of the features include:

- Each XBee Multi Programmer board allows you to program up to six devices simultaneously. Connect more boards to increase the programming concurrency.
- Different board variants cover all the XBee form factors to program almost any Digi RF device.

Download the XBee Multi Programmer application from: digi.com/support/productdetail?pid=5641

See the *XBee Multi Programmer User Guide* for more information.

Modes

Transparent operating mode

Devices operate in this mode by default. The device acts as a serial line replacement when it is in Transparent operating mode. The device queues all UART data it receives through the DIN pin for RF transmission. When a device receives RF data, it sends the data out through the DOUT pin.

API operating mode

API operating mode is an alternative to Transparent operating mode. API mode is a frame-based protocol that allows you to direct data on a packet basis. The device communicates UART or SPI data in packets, also known as API frames. This mode allows for structured communications with computers and microcontrollers.

The advantages of API operating mode include:

- It is easier to send information to multiple destinations
- The host receives the source address for each received data frame
- You can change parameters without entering Command mode
- You can query or set a configuration parameter while a pending command—for example **ND**—is in progress. This cannot be done in Command mode.

Command mode

Command mode is a state in which the firmware interprets incoming characters as commands. It allows you to modify the device's configuration using parameters you can set using AT commands. When you want to read or set any parameter of the XBee 3 DigiMesh RF Module using this mode, you have to send an AT command. Every AT command starts with the letters **AT** followed by the two characters that identify the command and then by some optional configuration values.

The operating modes of the XBee 3 DigiMesh RF Module are controlled by the AP (API Enable) setting, but Command mode is always available as a mode the device can enter while configured for any of the operating modes.

Command mode is available on the UART interface for all operating modes.

You cannot use the SPI interface to enter Command mode unless using SPI for the serial interface.

Enter Command mode

When using the default configuration values for **GT** and **CC**, you must enter **+++** preceded and followed by one second of silence—no input—to enter Command mode. However, both **GT** and **CC** are configurable. This means that the silence before and after the escape sequence—**GT**—and the escape characters themselves—**CC**—can be changed. For example, if **GT** is **5DC** and **CC** is **31**, then Command mode can be entered by typing **111** preceded and followed by 1.5 seconds of silence. When the entrance criteria are met the device responds with **OK\r** on UART signifying that it has entered Command mode successfully and is ready to start processing AT commands.

If configured to operate in Transparent operating mode, when entering Command mode the XBee 3 DigiMesh RF Module knows to stop sending data and start accepting commands locally.

Note Do not press **Return** or **Enter** after typing **+++** because it interrupts the guard time silence and prevents you from entering Command mode.

When the device is in Command mode, it listens for user input and is able to receive AT commands on the UART. If **CT** time (default is 10 seconds) passes without any user input, the device drops out of

Command mode and returns to the previous operating mode. You can force the device to leave Command mode by sending CN (Exit Command mode).

You can customize the command character, the guard times and the timeout in the device’s configuration settings. For more information, see CC (Command Character), CT (Command Mode Timeout) and GT (Guard Time).

Troubleshooting

Failure to enter Command mode is often due to baud rate mismatch. Ensure that the baud rate of the connection matches the baud rate of the device. By default, BD (UART Baud Rate) = **3** (9600 b/s).

There are two alternative ways to enter Command mode:

- A serial break for six seconds enters Command mode. You can issue the "break" command from a serial console, it is often a button or menu item.
- Asserting DIN (serial break) upon power up or reset enters Command mode. XCTU guides you through a reset and automatically issues the break when needed.

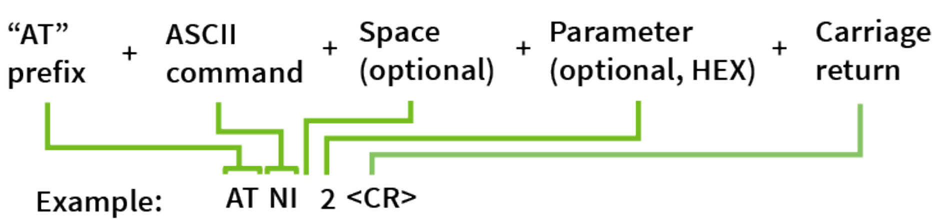
Note You must assert $\overline{\text{RTS}}$ for both of these methods, otherwise the device enters the bootloader.

Both of these methods temporarily set the device's baud rate to 9600 and return an **OK** on the UART to indicate that Command mode is active. When Command mode exits, the device returns to normal operation at the baud rate that **BD** is set to.

Send AT commands

Once the device enters Command mode, use the syntax in the following figure to send AT commands. Every AT command starts with the letters **AT**, which stands for "attention." The AT is followed by two characters that indicate which command is being issued, then by some optional configuration values.

To read a parameter value stored in the device’s register, omit the parameter field.



The preceding example changes NI (Network Identifier) to **2**.

Multiple AT commands

You can send multiple AT commands at a time when they are separated by a comma in Command mode; for example, **ATNIMy XBee,AC<cr>**.

Note The behavior of the comma is the same as the behavior of the <CR> in the previous example except that the next command following the comma is not preceded by AT. The only real purpose of the comma is to reduce keystrokes.

The preceding example changes the **NI (Node Identifier)** to **My XBee** and makes the setting active through AC (Apply Changes).

Parameter format

Refer to the list of AT commands for the format of individual AT command parameters. Valid formats for hexidecimal values include with or without a leading **0x** for example **FFFF** or **0xFFFF**.

Response to AT commands

When using AT commands to set parameters the XBee 3 DigiMesh RF Module responds with **OK<cr>** if successful and **ERROR<cr>** if not.

Apply command changes

Any changes you make to the configuration command registers using AT commands do not take effect until you apply the changes. For example, if you send the **BD** command to change the baud rate, the actual baud rate does not change until you apply the changes. To apply changes:

1. Send AC (Apply Changes).
2. Send WR (Write). In this case, changes are only applied following a reset. The **WR** command by itself does not apply changes.
   or:
3. Exit Command mode. You can exit Command mode in two ways: Either enter the **CN** command or wait for Command mode to timeout as specified by the **CT** parameter.

Make command changes permanent

Send a WR (Write) command to save the changes. **WR** writes parameter values to non-volatile memory so that parameter modifications persist through subsequent resets.

Send an RE (Restore Defaults) followed by **WR** to restore parameters back to their factory defaults. The next time the device is reset the default settings are applied.

Exit Command mode

1. Send CN (Exit Command mode) followed by a carriage return.
   or:
2. If the device does not receive any valid AT commands within the time specified by CT (Command Mode Timeout), it returns to Transparent or API mode. The default Command mode timeout is 10 seconds.

For an example of programming the device using AT Commands and descriptions of each configurable parameter, see AT commands.

Transmit mode

Transmit mode is the mode in which the device is transmitting data. This typically happens after data is received from the serial port.

Receive mode

This is the default mode for the XBee 3 DigiMesh RF Module. The device is in Receive mode when it is not transmitting data. If a destination node receives a valid RF data packet, the destination node transfers the data to its serial transmit buffer.

Serial communication

Serial interface

The XBee 3 DigiMesh RF Module interfaces to a host device through a serial port. The device can communicate through its serial port:

- Through logic and voltage compatible universal asynchronous receiver/transmitter (UART).
- Through a level translator to any serial device, for example through an RS-232 or USB interface board.
- Through SPI, as described in SPI communications.

Serial receive buffer

When serial data enters the XBee 3 DigiMesh RF Module through the serial port, the device stores the data in the serial receive buffer until it can be processed. Under certain conditions, the device may receive data when the serial receive buffer is already full. In that case, the device discards the data.

The serial receive buffer becomes full when data is streaming into the serial port faster than it can be processed and sent over the air (OTA). The size of the Serial receive buffer is 292 Bytes; the serial buffer may be reduced in size if RAM requirements cannot be met in future firmware releases. While the speed of receiving the data on the serial port can be much faster than the speed of transmitting data for a short period, sustained operation in that mode causes the device to drop data due to running out of places to put the data. Some things that may delay over the air transmissions are address discovery, route discovery, and retransmissions. Processing received RF data can also take away time and resources for processing incoming serial data.

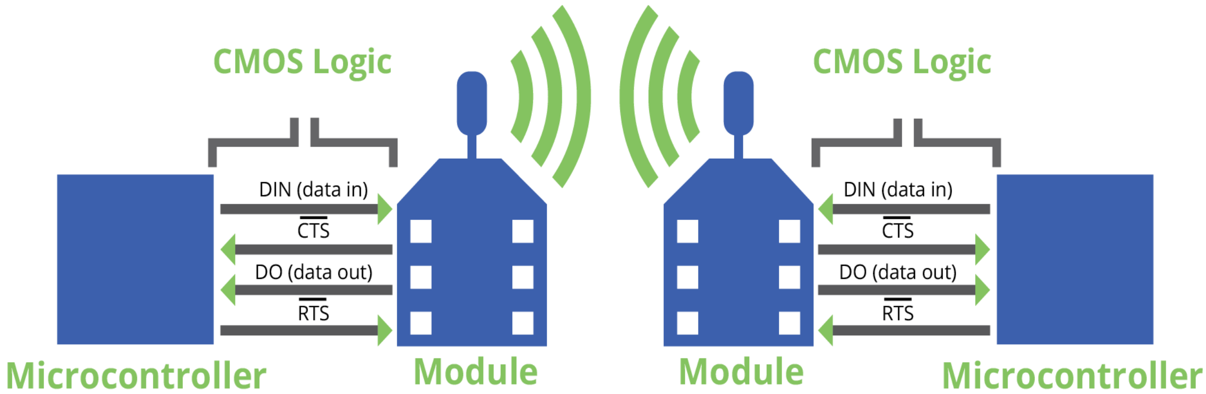
If the UART is the serial port and you enable the $\overline{\text{CTS}}$ flow control, the device alerts the external data source when the receive buffer is almost full. The host delays sending data to the device until the module asserts $\overline{\text{CTS}}$ again, allowing more data to come in.

Serial transmit buffer

When the device receives RF data, it moves the data into the serial transmit buffer and sends it out the UART. If the serial transmit buffer becomes full and the system buffers are also full, then it drops the entire RF data packet. The size of the Serial transmit buffer is 137 Bytes; the serial buffer may be reduced in size if RAM requirements cannot be met in future firmware releases. Whenever the device receives data faster than it can process and transmit the data out the serial port, there is a potential of dropping data.

UART data flow

Devices that have a UART interface connect directly to the pins of the XBee 3 DigiMesh RF Module as shown in the following figure. The figure shows system data flow in a UART-interfaced environment. Low-asserted signals have a horizontal line over the signal name.



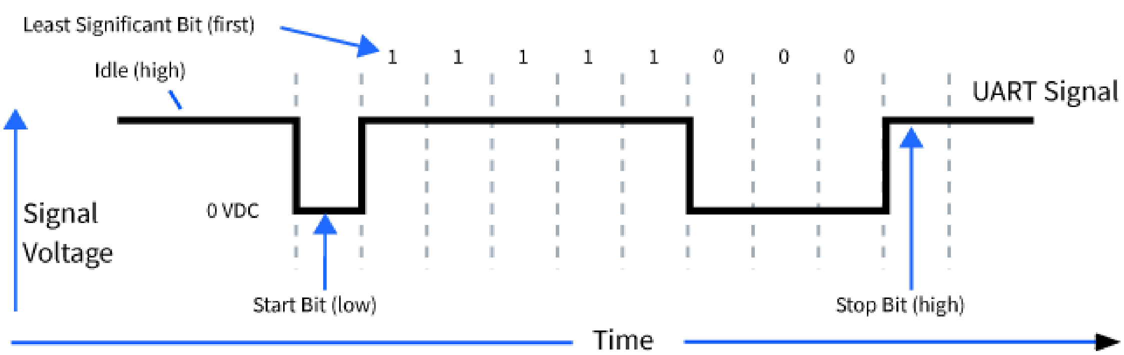
For more information about hardware specifications for the UART, see the *XBee 3 Hardware Reference Manual*.

Serial data

A device sends data to the XBee 3 DigiMesh RF Module's UART as an asynchronous serial signal. When the device is not transmitting data, the signals should idle high.

For serial communication to occur, you must configure the UART of both devices (the microcontroller and the XBee 3 DigiMesh RF Module) with compatible settings for the baud rate, parity, start bits, stop bits, and data bits.

Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following diagram illustrates the serial bit pattern of data passing through the device. The diagram shows UART data packet 0x1F (decimal number 31) as transmitted through the device.

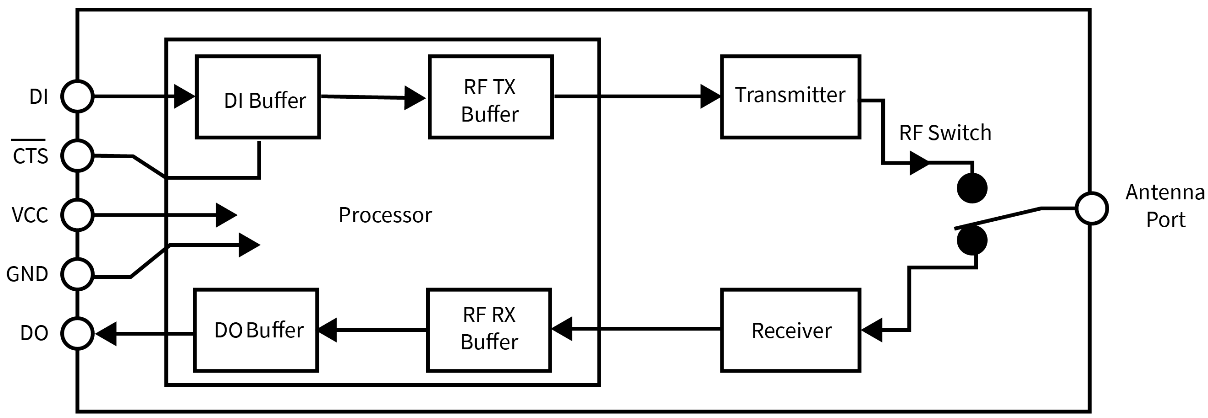


You can configure the UART baud rate, parity, and stop bits settings on the device with the **BD**, **NB**, and **SB** commands respectively. For more information, see UART interface commands.

Flow control

The XBee 3 DigiMesh RF Module maintains buffers to collect serial and RF data that it receives. The serial receive buffer collects incoming serial characters and holds them until the device can process them. The serial transmit buffer collects the data it receives via the RF link until it transmits that data out the serial port. The following figure shows the process of device buffers collecting received serial data.

Use D6 (DIO6/RTS Configuration) and D7 (DIO7/CTS Configuration) to set flow control.



Clear-to-send (CTS) flow control

If you enable CTS flow control (D7 (DIO7/CTS Configuration)), when the serial receive buffer is more than **FT** bytes full, the device de-asserts CTS (sets it high) to signal to the host device to stop sending serial data. The device reasserts CTS after the serial receive buffer has less than **FT** bytes in it. See FT (Flow Control Threshold) to configure and read this threshold.

RTS flow control

If you set D6 (DIO6/RTS Configuration) to enable RTS flow control, the device does not send data in the serial transmit buffer out the DOUT pin as long as RTS is de-asserted (set high). Do not de-assert RTS for long periods of time or the serial transmit buffer will fill. If the device receives an RF data packet and the serial transmit buffer does not have enough space for all of the data bytes, it discards the entire RF data packet.

If the device sends data out the UART when RTS is de-asserted (set high) the device could send up to five characters out the UART port after RTS is de-asserted.

Cases in which the DO buffer may become full, resulting in dropped RF packets:

1. If the RF data rate is set higher than the interface data rate of the device, the device may receive data faster than it can send the data to the host. Even occasional transmissions from a large number of devices can quickly accumulate and overflow the transmit buffer.
2. If the host does not allow the device to transmit data out from the serial transmit buffer due to being held off by hardware flow control.

SPI operation

This section specifies how SPI is implemented on the device, what the SPI signals are, and how full duplex operations work.

SPI communications

The XBee 3 DigiMesh RF Module supports SPI communications in slave mode. Slave mode receives the clock signal and data from the master and returns data to the master. The following table shows the signals that the SPI port uses on the device.

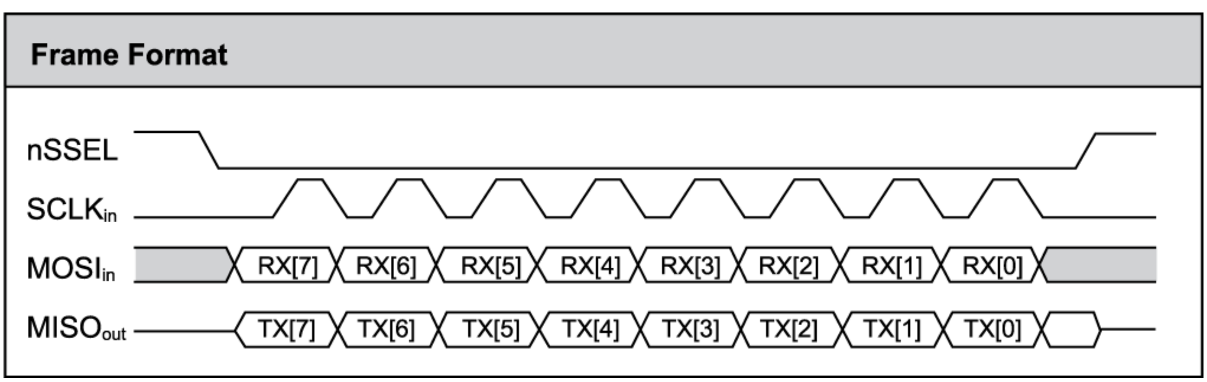
Refer to the XBee 3 Hardware Reference Guide for the pinout of your device.

Signal	Direction	Function
SPI_MOSI (Master Out, Slave In)	Input	Inputs serial data from the master
SPI_MISO (Master In, Slave Out)	Output	Outputs serial data to the master
SPI_SCLK (Serial Clock)	Input	Clocks data transfers on MOSI and MISO
SPI_$\overline{\text{SSEL}}$ (Slave Select)	Input	Enables serial communication with the slave
SPI_$\overline{\text{ATTN}}$ (Attention)	Output	Alerts the master that slave has data queued to send. The XBee 3 DigiMesh RF Module asserts this pin as soon as data is available to send to the SPI master and it remains asserted until the SPI master has clocked out all available data.

In this mode:

- SPI clock rates up to 5 MHz (burst) are possible.
- Data is most significant bit (MSB) first; bit 7 is the first bit of a byte sent over the interface.
- Frame Format mode 0 is used. This means CPOL= 0 (idle clock is low) and CPHA = 0 (data is sampled on the clock's leading edge).
- The SPI port only supports API Mode (**AP** = **1**).

The following diagram shows the frame format mode 0 for SPI communications.



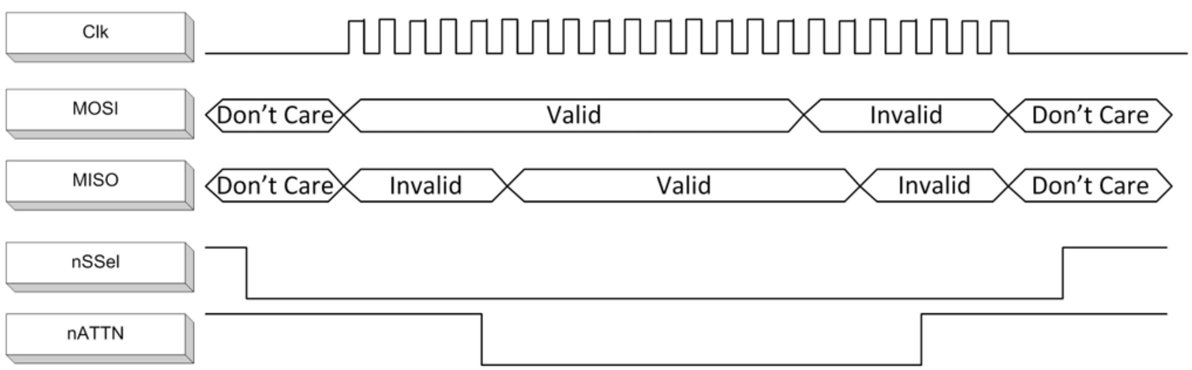
SPI mode is chip to chip communication. We do not supply a SPI communication interface on the XBee development evaluation boards included in the development kit.

Full duplex operation

When using SPI on the XBee 3 DigiMesh RF Module the device uses API operation without escaped characters to packetize data. The device ignores the configuration of **AP** because SPI does not operate in any other mode. SPI is a full duplex protocol, even when data is only available in one direction. This means that whenever a device receives data, it also transmits, and that data is normally invalid. Likewise, whenever a device transmits data, invalid data is probably received. To determine whether or not received data is invalid, the firmware places the data in API packets.

SPI allows for valid data from the slave to begin before, at the same time, or after valid data begins from the master. When the master sends data to the slave and the slave has valid data to send in the middle of receiving data from the master, a full duplex operation occurs, where data is valid in both directions for a period of time. Not only must the master and the slave both be able to keep up with the full duplex operation, but both sides must honor the protocol.

The following figure illustrates the SPI interface while valid data is being sent in both directions.



Low power operation

Sleep modes generally work the same on SPI as they do on UART. However, due to the addition of SPI mode, there is an option of another sleep pin, as described below.

By default, Digi configures DIO8 (SLEEP_REQUEST) as a peripheral and during pin sleep it wakes the device and puts it to sleep. This applies to both the UART and SPI serial interfaces.

If SLEEP_REQUEST is not configured as a peripheral and SPI_SSEL is configured as a peripheral, then pin sleep is controlled by SPI_SSEL rather than by SLEEP_REQUEST. Asserting SPI_SSEL by driving it low either wakes the device or keeps it awake. Negating SPI_SSEL by driving it high puts the device to sleep.

Using SPI_SSEL to control sleep and to indicate that the SPI master has selected a particular slave device has the advantage of requiring one less physical pin connection to implement pin sleep on SPI. It has the disadvantage of putting the device to sleep whenever the SPI master negates SPI_SSEL (meaning time is lost waiting for the device to wake), even if that was not the intent.

If the user has full control of SPI_SSEL so that it can control pin sleep, whether or not data needs to be transmitted, then sharing the pin may be a good option in order to make the SLEEP_REQUEST pin available for another purpose. Without control of SPI_SSEL while using it for sleep request, the device may go to sleep at inopportune times.

If the device is one of multiple slaves on the SPI, then the device sleeps while the SPI master talks to the other slave, but this is acceptable in most cases.

If you do not configure either pin as a peripheral, then the device stays awake, being unable to sleep in **SM**1 mode.

Select the SPI port

To force SPI mode on through-hole devices, hold DOUT/DIO13 low while resetting the device until SPI_ATTN asserts. This causes the device to disable the UART and go straight into SPI communication mode. Once configuration is complete, the device queues a modem status frame to the SPI port, which causes the SPI_ATTN line to assert. The host can use this to determine that the SPI port is configured properly.

On surface-mount devices, forcing DOUT low at the time of reset has no effect. To use SPI mode on the SMT modules, assert the SPI_SSEL low after reset and before any UART data is input.

Forcing DOUT low on TH devices forces the device to enable SPI support by setting the following configuration values:

Through-hole	Micro and Surface-mount	SPI signal
D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)	P9 (DIO19/SPI_ATTN Configuration)	$\overline{\text{ATTN}}$
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	P8 (DIO18/SPI_CLK Configuration)	SCLK
D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)	P7 (DIO17/SPI_SSEL Configuration)	$\overline{\text{SSEL}}$
D4 (DIO4/TH_SPI_MOSI Configuration)	P6 (DIO16/SPI_MOSI Configuration)	MOSI
P2 (DIO12/TH_SPI_MISO Configuration)	P5 (DIO15/SPI_MISO Configuration)	MISO

Note The $\overline{\text{ATTN}}$ signal is optional—you can still use SPI mode if you disable the SPI_$\overline{\text{ATTN}}$ pin (**D1** on through-hole or **P9** on surface-mount devices).

As long as the host does not issue a **WR** command, these configuration values revert to previous values after a power-on reset. If the host issues a **WR** command while in SPI mode, these same parameters are written to flash, and after a reset the device continues to operate in SPI mode.

If the UART is disabled and the SPI is enabled in the written configuration, then the device comes up in SPI mode without forcing it by holding DOUT low. If both the UART and the SPI are configured (P3 (DIO13/UART_DOUT) through P9 (DIO19/SPI_ATTN Configuration) are set to **1**) at the time of reset, then output goes to the UART until the host sends the first input to the SPI interface. As soon as the first input comes on the SPI port, then all subsequent output goes to the SPI port and the UART is disabled.

Once you select a serial port (UART or SPI), all subsequent output goes to that port, even if you apply a new configuration. Once the SPI interface is made active, the only way to switch the selected serial port back to UART is to reset the device.

When the master asserts the slave select (SPI_$\overline{\text{SSEL}}$) signal, SPI transmit data is driven to the output pin SPI_MISO, and SPI data is received from the input pin SPI_MOSI. The SPI_SSEL pin has to be asserted to enable the transmit serializer to drive data to the output signal SPI_MISO. A rising edge on SPI_SSEL causes the SPI_MISO line to be tri-stated such that another slave device can drive it, if so desired.

If the output buffer is empty, the SPI serializer transmits the last valid bit repeatedly, which may be either high or low. Otherwise, the device formats all output in API mode 1 format, as described in Operate in API mode. The attached host is expected to ignore all data that is not part of a formatted API frame.

Force UART operation

If you configure a device with only the SPI enabled and no SPI master is available to access the SPI slave port, you can recover the device to UART operation by holding DIN / CONFIG low at reset time. DIN/CONFIG forces a default configuration on the UART at 9600 baud and brings up the device in Command mode on the UART port. You can then send the appropriate commands to the device to configure it for UART operation. If you write those parameters, the device comes up with the UART enabled on the next reset.

I/O support

The following topics describe analog and digital I/O line support, line passing and output control.

Digital I/O support

Digital I/O is available on lines DIO0 through DIO12 (D0 (DIO0/ADC0/Commissioning Configuration) - D9 (DIO9/ON_SLEEP Configuration) and P0 (DIO10/RSSI/PWM0 Configuration) - P4 (DIO14/UART_DIN Configuration)). Digital sampling is enabled on these pins if configured as 3, 4, or 5 with the following meanings:

- 3 is digital input.
  - Use PR (Pull-up/Down Resistor Enable) to enable internal pull up/down resistors for each digital input. Use PD (Pull Up/Down Direction) to determine the direction of the internal pull up/down resistor. All disabled and digital input pins are pulled up by default.
- 4 is digital output low.
- 5 is digital output high.

Function	Micro Pin	SMT Pin	TH Pin	AT Command
DIO0	31	33	20	D0 (DIO0/ADC0/Commissioning Configuration)
DIO1	30	32	19	D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)
DIO2	29	31	18	D2 (DIO2/ADC2/TH_SPI_CLK Configuration)
DIO3	28	30	17	D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)
DIO4	23	24	11	D4 (DIO4/TH_SPI_MOSI Configuration)
DIO5	26	28	15	D5 (DIO5/Associate Configuration)
DIO6	27	29	16	D6 (DIO6/RTS Configuration)
DIO7	24	25	12	D7 (DIO7/CTS Configuration)
DIO8	9	10	9	D8 (DIO8/DTR/SLP_Request Configuration)
DIO9	25	26	13	D9 (DIO9/ON_SLEEP Configuration)
DIO10	7	7	6	P0 (DIO10/RSSI/PWM0 Configuration)
DIO11	8	8	7	P1 (DIO11/PWM1 Configuration)
DIO12	5	5	4	P2 (DIO12/TH_SPI_MISO Configuration)
DIO13	3	3	2	P3 (DIO13/UART_DOUT)
DIO14	4	4	3	P4 (DIO14/UART_DIN Configuration)

I\O sampling is not available for pins P5 through P9. See the *XBee 3 Hardware Reference Manual* for full pinouts and functionality.

Analog I/O support

Analog input is available on **D0** through **D3**. Configure these pins to **2** (ADC) to enable analog sampling.

PWM output is available on **P0** and **P1**, which can be used for Analog line passing. Use M0 (PWM0 Duty Cycle) and M1 (PWM1 Duty Cycle) to set a fixed PWM level.

Function	Micro Pin	SMT Pin	TH Pin	AT Command
ADC0	31	33	20	D0 (DIO0/ADC0/Commissioning Configuration)
ADC1	30	32	19	D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)
ADC2	29	31	18	D2 (DIO2/ADC2/TH_SPI_CLK Configuration)
ADC3	28	30	17	D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)
PWM0	7	7	6	P0 (DIO10/RSSI/PWM0 Configuration)
PWM1	8	8	7	P1 (DIO11/PWM1 Configuration)

AV (Analog Voltage Reference) specifies the analog reference voltage used for the 10-bit ADCs. Analog sample data is represented as a 2-byte value. For a 10-bit ADC, the acceptable range is from **0x0000** to **0x03FF**. To convert this value to a useful voltage level, apply the following formula:

ADC / 1023 (vREF) = Voltage

Note ADCs sampled through MicroPython will have 12-bit resolution.

Example

An ADC value received is 0x01AE; to convert this into a voltage the hexadecimal value is first converted to decimal (0x01AE = 430). Using the default **AV** reference of 1.25 V, apply the formula as follows:

430 / 1023 (1.25 V) = 525 mV

Monitor I/O lines

You can monitor pins you configure as digital input, digital output, or analog input and generate I/O sample data. If you do not define inputs or outputs, no sample data is generated.

Typically, I/O samples are generated by configuring the device to sample I/O pins periodically (based on a timer) or when a change is detected on one or more digital pins. These samples are always sent over the air to the destination address specified with DH (Destination Address High) and DL (Destination Address Low).

You can also gather sample data using on-demand sampling, which allows you to interrogate the state of the device's I/O pins by issuing an AT command. You can do this on either a local or remote device via an AT command request.

The three methods to generate sample data are:

- Periodic sample (IR (Sample Rate))
  - Periodic sampling based on a timer
  - Samples are taken immediately upon wake (excluding pin sleep)
  - Sample data is sent to **DH**+**DL** destination address
  - Can be used with line passing
  - Requires API mode on receiver
- Change detect (IC (DIO Change Detect))
  - Samples are generated when the state of specified digital input pin(s) change
  - Sample data is sent to **DH**+**DL** destination address

- Can be used with line passing
- Requires API mode on receiver

■ On-demand sample (IS (I/O Sample))

- Immediately query the device's I/O lines
- Can be issued locally in Command Mode
- Can be issued locally or remotely in API mode

These methods are not mutually exclusive and you can use them in combination with each other.

I/O sample data format

Regardless of how I/O data is generated, the format of the sample data is always represented as a series of bytes in the following format:

Bytes	Name	Description
1	Sample sets	Number of sample sets. There is always one sample set per frame.
2	Digital channel mask	Indicates which digital I/O lines have sampling enabled. Each bit corresponds to one digital I/O line on the device. bit 0 = DIO0 bit 1 = DIO1 bit 2 = DIO2 bit 3 = DIO3 bit 4 = DIO4 bit 5 = DIO5 bit 6 = DIO6 bit 7 = DIO7 bit 8 = DIO8 bit 9 = DIO9 bit 10 = DIO10 bit 11 = DIO11 bit 12 = DIO12 bit 13 = DIO13 bit 14 = DIO14 bit 15 = N/A Example: a digital channel mask of 0x002F means DIO0, 1, 2, 3 and 5 are configured as digital inputs or outputs.
1	Analog channel mask	Indicates which lines have analog inputs enabled for sampling. Each bit in the analog channel mask corresponds to one analog input channel. If a bit is set, then a corresponding 2-byte analog data set is included. bit 0 = AD0/DIO0 bit 1 = AD1/DIO1 bit 2 = AD2/DIO2 bit 3 = AD3/DIO3
2	Digital data set	Each bit in the digital data set corresponds to a bit in the digital channel mask and indicates the digital state of the pin, whether high (1) or low (0). If the digital channel mask is 0x0000, then these two bytes are omitted as no

Bytes	Name	Description
		digital I/O lines are enabled.
2	Analog data set (multiple)	Each enabled ADC line in the analog channel mask will have a separate 2-byte value based on the number of ADC inputs on the originating device. The data starts with AD0 and continues sequentially for each enabled analog input channel up to AD3. If the analog channel mask is 0x00, then no analog sample bytes is included.

API frame support

I/O samples generated using Periodic I/O sampling (**IR**) and Digital I/O change detection (**IC**) are transmitted to the destination address specified by **DH** and **DL**. In order to display the sample data, the receiver must be operating in API mode (**AP** = **1** or **2**). The sample data is represented as an I/O sample API frame.

See I/O Sample Indicator - 0x92 for more information on the frame's format and an example.

On-demand sampling

You can use IS (I/O Sample) to query the current state of all digital I/O and ADC lines on the device and return the sample data as an AT command response. If no inputs or outputs are defined, the command returns an ERROR.

On-demand sampling can be useful when performing initial deployment, as you can send **IS** locally to verify that the device and connected sensors are correctly configured. The format of the sample data matches what is periodically sent using other sampling methods. You can also send **IS** remotely using a remote AT command. When sent remotely from a gateway or server to each sensor node on the network, on-demand sampling can improve battery life and network performance as the remote node transmits sample data only when requested instead of continuously.

If you send **IS** using Command mode, then the device returns a carriage return delimited list containing the I/O sample data. If **IS** is sent either locally or remotely via an API frame, the I/O sample data is presented as the parameter value in the AT command response frame (Local AT Command Response - 0x88 or Remote AT Command Response- 0x97).

Example: Command mode

An **IS** command sent in Command mode returns the following sample data:

Output	Description
01	One sample set
0C0C	Digital channel mask, indicates which digital lines are sampled (0x0C0C = 0000 **11**00 0000 **11**00b = DIO2, 3, 10, 11)
03	Analog channel mask, indicates which analog lines are sampled (0x03 = 0000 00**11**b = AD0, 1)
0408	Digital sample data that corresponds with the digital channel mask 0x0408 = 0000 **01**00 0000 **1**000b = DIO3 and DIO10 are high, DIO2 and DIO11 are low

Output	Description
03D0	Analog sample data for AD0
0124	Analog sample data for AD1

Example: Local AT command in API mode

The **IS** command sent to a local device in API mode would use a Local AT Command Request - 0x08 or Queue Local AT Command Request - 0x09 frame:

7E 00 04 08 53 49 53 08

The device responds with a Local AT Command Response - 0x88 that contains the sample data:

7E 00 0F 88 53 49 53 00 01 0C 0C 03 04 08 03 D0 01 24 68

Output	Field	Description
7E	Start Delimiter	Indicates the beginning of an API frame
00 0F	Length	Length of the packet
88	Frame type	AT Command response frame
53	Frame ID	This ID corresponds to the Frame ID of the 0x08 request
49 53	AT Command	Indicates the AT command that this response corresponds to 0x49 0x53 = **IS**
00	Status	Indicates success or failure of the AT command **00** = OK if no I/O lines are enabled, this will return 01 (ERROR)
01	I/O sample data	One sample set
0C 0C		Digital channel mask, indicates which digital lines are sampled (0x0C0C = 0000 **11**00 0000 **11**00b = DIO2, 3, 10, 11)
03		Analog channel mask, indicates which analog lines are sampled (0x03 = 0000 00**11**b = AD0, 1)
04 08		Digital sample data that corresponds with the digital channel mask 0x0408 = 0000 **01**00 0000 **10**00b = DIO3 and DIO10 are high, DIO2 and DIO11 are low
03 D0		Analog sample data for AD0
01 24		Analog sample data for AD1
68	Checksum	Can safely be discarded on received frames

Example: Remote AT command in API mode

The **IS** command sent to a remote device with an address of 0013A200 12345678 uses a Remote AT Command Request - 0x17:

7E 00 0F 17 87 00 13 A2 00 12 34 56 78 FF FE 00 49 53 FF

The sample data from the device is returned in a Remote AT Command Response- 0x97 frame with the sample data as the parameter value:

7E 00 19 97 87 00 13 A2 00 12 34 56 78 00 00 49 53 00 01 0C 0C 03 04 08 03 FF 03 FF 50

Output	Field	Description
7E	Start Delimiter	Indicates the beginning of an API frame
00 19	Length	Length of the packet
97	Frame type	Remote AT Command response frame
87	Frame ID	This ID corresponds to the Frame ID of the 0x17 request
0013A200 12345678	64-bit source	The 64-bit address of the node that responded to the request
0000	16-bit source	The 16-bit address of the node that responded to the request
49 53	AT Command	Indicates the AT command that this response corresponds to 0x49 0x53 = **IS**
00	Status	Indicates success or failure of the AT command **00** = **OK** if no I/O lines are enabled, this will return **01** (ERROR)
01	I/O sample data	One sample set
0C 0C		Digital channel mask, indicates which digital lines are sampled (0x0C0C = 0000 1100 0000 1100b = DIO2, 3, 10, 11)
03		Analog channel mask, indicates which analog lines are sampled (0x03 = 0000 0011b = AD0, 1)
04 08		Digital sample data that corresponds with the digital channel mask 0x0408 = 0000 0100 0000 1000b = DIO3 and DIO10 are high, DIO2 and DIO11 are low
03 D0		Analog sample data for AD0
01 24		Analog sample data for AD1
50	Checksum	Can safely be discarded on received frames

Periodic I/O sampling

Periodic sampling allows a device to take an I/O sample and transmit it to a remote device at a periodic rate.

Source

Use IR (Sample Rate) to set the periodic sample rate for enabled I/O lines.

- To disable periodic sampling, set **IR** to **0**.
- For all other **IR** values, the device samples data when **IR** milliseconds elapse and transmits the sampled data to the destination address.

The DH (Destination Address High) and DL (Destination Address Low) commands determine the destination address of the I/O samples. You must configure at least one pin as a digital I/O or ADC input on the sending node to generate sample data.

Destination

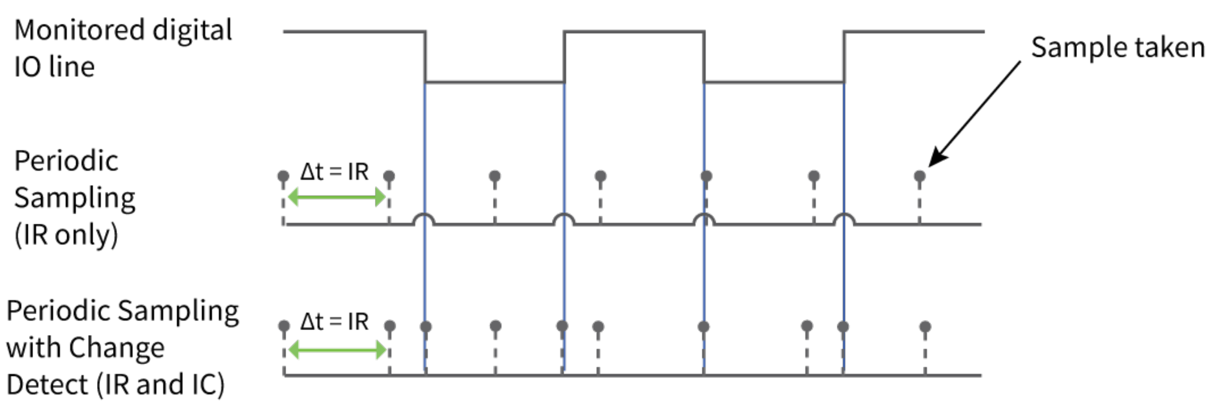
If the receiving device is operating in API operating mode the I/O sample data format is emitted out of the serial port. Devices that are in Transparent operating mode discard the I/O data samples they receive unless you enable line passing.

Digital I/O change detection

You can configure devices to transmit a data sample immediately whenever a monitored digital I/O pin changes state. IC (DIO Change Detect) is a bitmask that determines which digital I/O lines to monitor for a state change. If you set one or more bits in **IC**, the device transmits an I/O sample as soon it observes a state change on the monitored digital I/O line(s) using edge detection.

Change detection is only applicable to digital I/O pins that are configured as digital input (**3**) or digital output (**4** or **5**).

The figure below shows how I/O change detection can work in combination with Periodic I/O sampling to improve sampling accuracy. In the figure, the gray dashed lines with a dot on top represent samples taken from the monitored DIO line. The top graph shows only periodic IR samples, the bottom graph shows a combination of **IR** periodic samples and **IC** detected changes. In the top graph, the humps indicate that the sample was not taken at that exact moment and needed to wait for the next **IR** sample period.



Note Use caution when combining change detect sampling with sleep modes. **IC** only causes a sample to be generated if a state change occurs during a wake period. If the device is sleeping when the digital transition occurs, then no change is detected and an I/O sample is not generated.
Use periodic sampling with **IR** in conjunction with **IC** in this instance, since **IR** generates an I/O sample upon wakeup and ensures that the change is properly observed.

I/O line passing

Line passing allows you to affect the output pins of one device by sampling the I/O pins of another. To support line passing, you must configure a device to generate I/O sample data using periodic sampling (IR (Sample Rate)) and/or change detection (IC (DIO Change Detect)).

On the device that receives I/O samples, enable line passing setting IA (I/O Input Address) with the address of the device that has the appropriate inputs enabled. This effectively binds the outputs to a particular device's input. This does not affect the ability of the device to receive I/O line data from other devices—only its ability to update enabled outputs. Set **IA** to **0xFFFF** (broadcast address) to affect the output using input data from any device on the network.

Digital line passing

Digital I/O lines are mapped in pairs; pins configured as digital input on the transmitting device affect the corresponding digital output pin on the receiving device. For example, a device that samples **D5** as an input (3) only affects **D5** on the receiver if **D5** is configured as an output (4 or 5).

Each digital pin has an associated timeout value. When an I/O sample is received that affects a digital output pin, the pin returns to its configured state after the timeout period expires. For pins **D0** through **D9**, the associated timeout commands are T0 (D0 Timeout) through T9 (D9 Timeout). For pins **P0** through **P4**, the associated timeout commands are Q0 (P0 Timeout) through Q2 (P2 Timeout).

Digital line passing is only available on pins **D0** through **P3**. You cannot use UART and SPI pins for line passing.

Example: Digital line passing

A sampling XBee 3 DigiMesh RF Module is configured with the following settings:

AT command	Parameter value
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	3 (digital input)
IR (Sample Rate)	0x7D0 (2 seconds)
DH (Destination Address High)	0013A200
DL (Destination Address Low)	12345678

Every two seconds, an I/O sample is generated and sent to the address specified by **DH** and **DL**. The receiver is configured with the following settings:

AT command	Parameter value
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	5 (digital output low)
T2 (D2 Output Timeout)	0x64 (10 seconds)
IA (I/O Input Address)	00103A20012345678

When this device receives an incoming I/O sample, if the source address matches the one set by **IA**, the device sets the output of **D2** to match the input of **D2** of the receiver. This output level holds for ten seconds before the pin returns to a digital output low state.

Analog line passing

Similar to digital line passing, analog line passing pairs the Analog I/O support of one device to a PWM output of another. There are two PWM output pins that can simulate the voltage measured by the ADC inputs. Be aware that ADC inputs are on different pins than the corresponding PWM outputs: AD0 corresponds to PWM0, and AD1 corresponds to PWM1. See Analog I/O support for the pinouts.

You can set the analog line passing timeout value with PT (PWM Output Timeout), which affects both PWM output pins. You can explicitly set a PWM output level using the M0 (PWM0 Duty Cycle) and M1 (PWM1 Duty Cycle) commands, when an I/O sample is received that affects a PWM output pin, it returns to its configured state after the **PT** timeout period expires.

Example: Analog line passing

A sampling device is configured with the following settings:

AT command	Parameter value
	2 (ADC input)
IR (Sample Rate)	0x7D0 (2 seconds)
DH (Destination Address High)	0013A200
DL (Destination Address Low)	12345678

Every two seconds, an I/O sample frame is generated and sent to the address specified by **DH** and **DL**. The receiver is configured with the following settings:

AT command	Parameter value
P0	2 (PWM output)
M0	0
PT	0x12C (30 seconds)
IA	0013A20087654321

When this device receives an incoming I/O sample, if the source address matches the one set by **IA**, the device sets the PWM output of **P0** to match the ADC input of **D0** of the receiver. This output level holds for thirty seconds before the pin returns to a digital output low state.

Output sample data

If a device receives an I/O sample whose address matches that set by IA (I/O Input Address), it triggers line passing. Line passing operates whether the receiving device is operating in API or Transparent mode.

By default, if the receiver is configured for API mode, it outputs the I/O sample frame in addition to affecting output pins. You can suppress the I/O sample frame output by setting IU (Send I/O Sample to Serial Port) to **0**. This only suppresses I/O samples that trigger line passing, a sample generated from a device whose address does not match the **IA** address is sent regardless of **IU**.

Output control

IO (Set Digital I/O Lines) controls the output levels of D0 (DIO0/ADC0/Commissioning Configuration) through D7 (DIO7/CTS Configuration) that are configured as output pins (either **4** or **5**). These values override the configured output levels of the pins until they are changed again (the pins do not automatically revert to their configured values after a timeout.)

You can use **IO** to trigger a sample on change detect.

I/O behavior during sleep

When the device sleeps (**SM !** = **0**) the I/O lines are optimized for a minimal sleep current.

Digital I/O lines

Digital I/O lines set as digital output high or low maintain those values during sleep. Disabled or input pins continue to be controlled by the **PR**/**PD** settings. Peripheral pins (with the exception of CTS) are set low during sleep and SPI pins are set high. Peripheral and SPI pins resume normal operation upon wake.

Digital I/O lines that have been set using I/O line passing hold their values during sleep, however the digital timeout timer (**T0** through **T9**, and **Q0** through **Q2**) are suspended during sleep and resume upon wake.

Analog and PWM I/O lines

Lines configured as analog inputs or PWM output are not affected during sleep. PWM lines are shut down (set low) during sleep and resume normal operation upon wake.

PWM output pins set by analog line passing are shutdown during sleep and revert to their preset values (**M0** and **M1**) on wake. This happens regardless of whether the timeout has expired or not.

Networking

Network identifiers

You define DigiMesh networks with a unique network identifier. Use the **ID** command to set this identifier. For devices to communicate, you must configure them with the same network identifier and the same operating channel. For devices to communicate, the **CH** and **ID** commands must be equal on all devices in the network.

The **ID** command directs the devices to talk to each other by establishing that they are all part of the same network. The **ID** parameter allows multiple DigiMesh networks to co-exist on the same physical channel.

Operating channels

The XBee 3 DigiMesh RF Module operates over the 2.4 GHz band using direct sequence spread spectrum (DSSS) modulation. DSSS modulation allows the device to operate over a channel or frequency that you specify.

The 2.4 GHz frequency band defines 16 operating channels. The XBee 3 DigiMesh RF Module supports all 16 channels, but output power on channel 26 on the XBee 3 PRO RF Module is limited.

Use the **CH** command to select the operating channel on a device. **CH** tells the device the frequency to use to communicate.

For devices to communicate, the **CH** and **ID** commands must be equal on all devices in the network.

Note these requirements for communication:

- A device can only receive data from other devices within the same network (with the same **ID** value) and using the same channel (with the same **CH** value).
- A device can only transmit data to other devices within the same network (with the same **ID** value) and using the same channel (with the same **CH** value).

Delivery methods

The TO (Transmit Options) command sets the default delivery method that the device uses when in Transparent mode. In API mode, the TxOptions field of the API frame overrides the **TO** command, if non-zero.

The XBee 3 DigiMesh RF Module supports three delivery methods:

- Point-to-multipoint (**TO** = 0x40).
- Repeater (directed broadcast) (**TO** = 0x80).
- DigiMesh (**TO** = 0xC0).

Point-to-multipoint

To select point-to-multipoint, set the transmit options to 0x40.

In Transparent mode, use the **TO** (Transmit Options) command to set the transmit options.

In API mode, use the Transmit Request (0x10) and Explicit Addressing Command (0x11) frames to set the transmit options. However, if the transmit options in the API frame are zero, then the transmit options in the **TO** command apply.

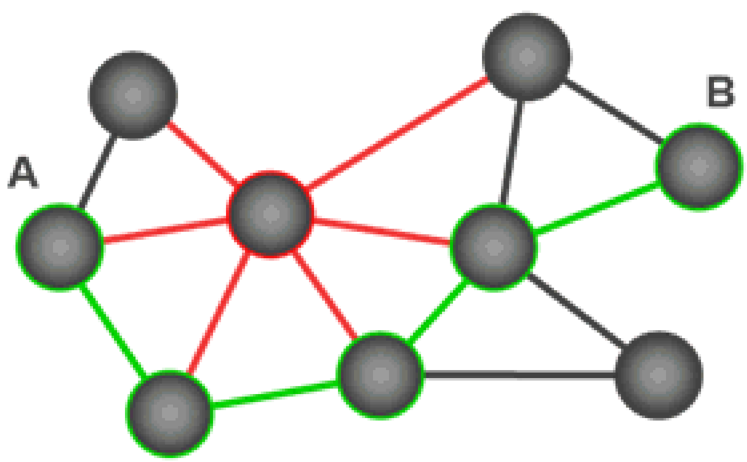
Point-to-multipoint transmissions occur between two adjacent nodes within RF range. No route discovery and no routing occur for these types of transmissions. The networking layer is entirely skipped.

Point-to-multipoint has an advantage over DigiMesh for two adjacent devices due to less overhead. However, it cannot work over multiple hops.

DigiMesh networking

A mesh network is a topology in which each node in the network is connected to other nodes around it. Each node cooperates in transmitting information. Mesh networking provides these important benefits:

- **Routing**. With this technique, the message is propagated along a path by hopping from node to node until it reaches its final destination.
- **Ad-hoc network creation**. This is an automated process that creates an entire network of nodes on the fly, without any human intervention.
- **Self-healing**. This process automatically figures out if one or more nodes on the network is broken, impeded by the environment, or powered off and repairs any broken routes.
- **Peer-to-peer architecture**. No hierarchy and no parent-child relationships are needed.
- **Quiet protocol**. Routing overhead will be reduced by using a reactive protocol similar to AODV.
- **Route discovery**. Rather than maintaining a network map, routes will be discovered and created only when needed.
- **Selective acknowledgments**. Only the destination node will reply to route requests.
- **Reliable delivery**. Reliable delivery of data is accomplished by means of acknowledgments.
- **Sleep modes.** Low power sleep modes with synchronized wake are supported with variable sleep and wake times.



With mesh networking, the distance between two nodes does not matter as long as there are enough nodes in between to pass the message along. When one node wants to communicate with another, the network automatically calculates the best path.

A mesh network is also reliable and offers redundancy. For example, If a node can no longer operate because it has been removed from the network or because a barrier blocks its ability to communicate, the rest of the nodes can still communicate with each other, either directly or through intermediate nodes.

Broadcast addressing

All of the routers in a network receive and repeat broadcast transmissions. Broadcast transmissions do not use ACKs, so the sending device sends the broadcast multiple times. By default, the sending device sends a broadcast transmission four times. The transmissions become automatic retries without acknowledgments. This results in all nodes repeating the transmission four times as well.

In order to avoid RF packet collisions, the network inserts a random delay before each router relays the broadcast message. You can change this random delay time with the **NN** parameter.

Sending frequent broadcast transmissions can quickly reduce the available network bandwidth. Use broadcast transmissions sparingly.

The broadcast address is a 64 bit address with the lowest 16 bits set to 1. The upper bits are set to 0. To send a broadcast transmission:

- Set **DH** to 0.
- Set **DL** to 0xFFFF.

In API operating mode, this sets the destination address to 0x000000000000FFFF.

Unicast addressing

When devices transmit using DigiMesh unicast, the network uses retries and acknowledgments (ACKs) for reliable data delivery. In a retry and acknowledgment scheme, for every data packet that a device sends, the receiving device must send an acknowledgment back to the transmitting device to let the sender know that the data packet arrived at the receiver. If the transmitting device does not receive an acknowledgment then it re-sends the packet. It sends the packet a finite number of times before the system times out.

The **MR** (Mesh Network Retries) parameter determines the number of mesh network retries. The sender device transmits RF data packets up to **MR** + 1 times across the network route, and the receiver transmits ACKs when it receives the packet. If the sender does not receive a network ACK within the time it takes for a packet to traverse the network twice, the sender retransmits the packet.

If a device sends a unicast that uses both MAC and NWK retries and acknowledgments:

- Use MAC retries and acknowledgments for transmissions between adjacent devices in the route.
- Use NWK retries and acknowledgments across the entire route.

To send unicast messages while in Transparent operating mode, set the **DH** and **DL** on the transmitting device to match the corresponding **SH** and **SL** parameter values on the receiving device.

Route discovery

Route discovery is a process that occurs when:

1. The source node does not have a route to the requested destination.
2. A route fails. This happens when the source node uses up its network retries without receiving an ACK.

Route discovery begins by the source node broadcasting a route request (RREQ). We call any router that receives the RREQ and is not the ultimate destination, an intermediate node.

Intermediate nodes may either drop or forward a RREQ, depending on whether the new RREQ has a better route back to the source node. If so, the node saves, updates and broadcasts the RREQ.

When the ultimate destination receives the RREQ, it unicasts a route reply (RREP) back to the source node along the path of the RREQ. It does this regardless of route quality and regardless of how many times it has seen an RREQ before.

This allows the source node to receive multiple route replies. The source node selects the route with the best round trip route quality, which it uses for the queued packet and for subsequent packets with the same destination address.

Routing

A device within a mesh network determines reliable routes using a routing algorithm and table. The routing algorithm uses a reactive method derived from Ad-hoc On-demand Distance Vector (AODV). The firmware uses an associative routing table to map a destination node address with its next hop. A device sends a message to the next hop address, and the message either reaches its destination or forwards to an intermediate router that routes the message on to its destination.

If a message has a broadcast address, it is broadcast to all neighbors, then all routers that receive the message rebroadcast the message **MT**+1 times. Eventually, the message reaches the entire network.

Packet tracking prevents a node from resending a broadcast message more than **MT**+1 times. This means that a node that relays a broadcast will only relay it after it receives it the first time and it will discard repeated instances of the same packet.

Routers

You can use the **CE** command to configure devices in a DigiMesh network to act as routers or end devices. All devices in a DigiMesh network act as routers by default. Any devices that you configure as routers actively relay network unicast and broadcast traffic.

Repeater/directed broadcast

All of the routers in a network receive and repeat directed broadcast transmissions. Because it does not use ACKs, the originating node sends the broadcast multiple times. By default a broadcast transmission is sent four times—the extra transmissions become automatic retries without acknowledgments. This results in all nodes repeating the transmission four times. Sending frequent broadcast transmissions can quickly reduce the available network bandwidth, so use broadcast transmissions sparingly.

MAC layer

The MAC layer is the building block that is used to build repeater capability. To implement Repeater mode, we use a network layer header that comes after the MAC layer header in each packet. In this network layer there is additional packet tracking to eliminate duplicate broadcasts.

In this delivery method, the device sends both unicast and broadcast packets out as broadcasts that are always repeated. All repeated packets are sent to every device. The devices that receive the broadcast send broadcast data out their serial port.

When a device sends a unicast, it specifies a destination address in the network header. Then, only the device that has the matching destination address sends the unicast out its serial port. This is called a directed broadcast.

Any node that has a **CE** parameter set to router rebroadcasts the packet if its **BH** (broadcast hops) or broadcast radius values are not depleted. If a node has already seen a repeated broadcast, it ignores the broadcast.

The **BH** parameter sets the maximum number of hops that a broadcast is repeated, but there are two special cases. If **BH** is **0** or if **BH** is > **NH**, then **NH** specifies the maximum hops for broadcasts instead.

By default the **CE** parameter is set to route all broadcasts. As such, all nodes that receive a repeated packet will repeat it. If you change the **CE** parameter, you can limit which nodes repeat packets, which helps dense networks from becoming overly congested while packets are being repeated.

Transmission timeout calculations for Repeater/directed broadcast mode are the same as for DigiMesh broadcast transmissions.

The MAC layer is the building block that is used to build repeater capability. To implement Repeater mode, we use a network layer header that comes after the MAC layer header in each packet. In this network layer there is additional packet tracking to eliminate duplicate broadcasts.

Encryption

XBee 3 DigiMesh provides greater security against replay attacks and attempts to determine the plaintext. The XBee 3 DigiMesh RF Module performs Counter (CTR) mode encryption instead of Electronic Codebook (ECB) mode encryption. Since the counter is passed over-the-air (OTA) and changes with each frame, the same text is always encrypted differently and there are no known attacks to determine the plaintext from the ciphertext.

A side effect of this implementation is that the maximum payload is reduced by the size of the counter (8 bytes). Therefore, no frames can exceed 65 bytes with encryption enabled. The maximum payload is still 73 bytes with encryption disabled.

Also effective with XBee 3 DigiMesh, the key is 256 bits rather than 128 bits. 256 bits is 32 bytes. Since the key is entered with ASCII HEX characters in Command mode, up to 64 ASCII HEX characters may be entered for the **KY** command.

For compatibility with nodes in the same network that do not support CTR mode encryption, C8 (Compatibility Options) bit 2 was introduced to enable the 128-bit key with ECB mode encryption as supported previously. In this case, only the last 32 ASCII HEX characters of the key are used, even if more characters were previously entered for the key.

Maximum payload

DigiMesh uses the 802.15.4 PHY layer including a 2-byte CRC at the end of the frame. This reduces the size of each frame to 125 bytes. After the MAC header, the NWK header, and the APP header are included at the beginning of the packet, the remaining space is 73 bytes for payload. If CTR mode encryption is enabled, this number is further reduced to 65 bytes. The best way to determine the maximum payload is to read NP (Maximum Packet Payload Bytes).

Sending a packet securely across a Secure Session (API transmit option bit 4 enabled) reduces the maximum payload size by 4 bytes.

These maximums only apply in API mode. If you attempt to send an API packet with a larger payload than specified, the device responds with a Transmit Status frame (0x8B) with the Status field set to **74** (Data payload too large).

In Transparent mode, the firmware splits the data as necessary to cope with maximum payloads.

Network commissioning and diagnostics

We call the process of discovering and configuring devices in a network for operation, "network commissioning." Devices include several device discovery and configuration features. In addition to configuring devices, you must develop a strategy to place devices to ensure reliable routes. To accommodate these requirements, modules include features to aid in placing devices, configuring devices, and network diagnostics.

Local configuration

You can configure devices locally using serial commands in Command mode or API mode, or remotely using remote AT commands. Devices that are in API mode can send configuration commands to set or read the configuration settings of any device in the network.

Remote configuration

When you do not have access to the device's serial port, you can use a separate device in API mode to remotely configure it. To remotely configure devices, use the following steps.

Send a remote command

To send a remote command, populate the Remote AT Command Request - 0x17 with:

1. The 64-bit address of the remote device.
2. The correct command options value.
3. Optionally, the command and parameter data.
4. If you want a command response, set the Frame ID field to a non-zero value.

XCTU has a Frames Generator tool that can assist you with building and sending a remote AT frame; see Frames generator tool in the *XCTU User Guide*.

Apply changes on remote devices

When you use remote commands to change the command parameter settings on a remote device, you must apply the parameter changes or they do not take effect. For example, if you change the **BD** parameter, the actual serial interface rate does not change on the remote device until you apply the changes. You can apply the changes using remote commands in one of three ways:

1. Set the apply changes option bit in the API frame.
2. Send an **AC** command to the remote device.
3. Send the **WR** command followed by the **FR** command to the remote device to save the changes and reset the device.

Remote command response

If a local device sends a command request to a remote device, and the API frame ID is non-zero, the remote device sends a remote command response transmission back to the local device.

When the local device receives a remote command response transmission, it sends a remote command response API frame out its UART. The remote command response indicates:

1. The status of the command, which is either success or the reason for failure.
2. In the case of a command query, it includes the register value.

The device that sends a remote command does not receive a remote command response frame if:

1. It could not reach the destination device.
2. You set the frame ID to 0 in the remote command request.

Build aggregate routes

In many applications, many or all of the nodes in the network must transmit data to a central aggregator node. In a new DigiMesh network, the overhead of these nodes discovering routes to the aggregator node can be extensive and taxing on the network. To eliminate this overhead, you can use the **AG** command to automatically build routes to an aggregate node in a DigiMesh network.

To send a unicast, devices configured for Transparent mode (**AP** = 0) must set their **DH**/**DL** registers to the MAC address of the node that they need to transmit to. In networks of Transparent mode devices that transmit to an aggregator node it is necessary to set every device's **DH**/**DL** registers to the MAC address of the aggregator node. This can be a tedious process. A simple and effective method is to use the **AG** command to set the **DH**/**DL** registers of all the nodes in a DigiMesh network to that of the aggregator node.

Upon deploying a DigiMesh network, you can issue the **AG** command on the desired aggregator node to cause all nodes in the network to build routes to the aggregator node. You can optionally use the **AG** command to automatically update the **DH**/**DL** registers to match the MAC address of the aggregator node.

The **AG** command requires a 64-bit parameter. The parameter indicates the current value of the **DH**/**DL** registers on a device; typically you should replace this value with the 64-bit address of the node sending the **AG** broadcast. However, if you do not want to update the **DH**/**DL** of the device receiving the **AG** broadcast you can use the invalid address of 0xFFFE. The receiving nodes that are configured in API mode output an Aggregator Update API frame (0x8E) if they update their **DH**/**DL** address; for a description of the frame, see Aggregate Addressing Update - 0x8E.

All devices that receive an **AG** broadcast update their routing table information to build a route to the sending device, regardless of whether or not their **DH**/**DL** address is updated. The devices use this routing information for future DigiMesh unicast transmissions.

DigiMesh routing examples

Example one

In a scenario where you deploy a network, and then you want to update the **DH** and **DL** registers of all the devices in the network so that they use the MAC address of the aggregator node, which has the MAC address 0x0013A200 4052C507, you could use the following technique.

1. Deploy all devices in the network with the default **DH**/**DL** of 0xFFFF.
2. Serially, send an ATAGFFFF command to the aggregator node so it sends the broadcast transmission to the rest of the nodes.

All the nodes in the network that receive the **AG** broadcast set their **DH** to 0x0013A200 and their **DL** to 0x4052C507. These nodes automatically build a route to the aggregator node.

Example two

If you want all of the nodes in the network to build routes to an aggregator node with a MAC address of 0x0013A200 4052C507 without affecting the **DH** and **DL** registers of any nodes in the network:

1. Send the ATAGFFFE command to the aggregator node. This sends an **AG** broadcast to all of the nodes in the network.
2. All of the nodes internally update only their routing table information to contain a route to the aggregator node.

3. None of the nodes update their **DH** and **DL** registers because none of the registers are set to the 0xFFFE address.

Replace nodes

You can use the **AG** command to update the routing table and **DH**/**DL** registers in the network after you replace a device. To update only the routing table information without affecting the **DH** and **DL** registers, use the process in example two, above.

To update the **DH** and **DL** registers of the network, use the following example.

Example

This example shows how to cause all devices to update their **DH** and **DL** registers to the MAC address of the sending device. In this case, assume you are using a device with a serial number of 0x0013A200 4052C507 as a network aggregator, and the sending device has a MAC address of 0x0013A200 F5E4D3B2 To update the **DH** and **DL** registers to the sending device's MAC address:

1. Replace the aggregator with 0x0013A200 F5E4D3B2.
2. Send the ATAG0013A200 4052C507 command to the new device.

Test links between adjacent devices

It often helps to test the quality of a link between two adjacent modules in a network. You can use the Test Link Request Cluster ID to send a number of test packets between any two devices in a network. To clarify the example, we refer to "device A" and "device B" in this section.

To request that device B perform a link test against device A:

1. Use device A in API mode (**AP** = 1) to send an Explicit Addressing Command (0x11) frame to device B.
2. Address the frame to the Test Link Request Cluster ID (0x0014) and destination endpoint: 0xE6.
3. Include a 12-byte payload in the Explicit Addressing Command frame with the following format:

Number of bytes	Field name	Description
8	Destination address	The address the device uses to test its link. For this example, use the device A address.
2	Payload size	The size of the test packet. Use the **NP** command to query the maximum payload size for the device.
2	Iterations	The number of packets to send. This must be a number between 1 and 4000.

4. Device B should transmit test link packets.
5. When device B completes transmitting the test link packets, it sends the following data packet to device A's Test Link Result Cluster (0x0094) on endpoint (0xE6).
6. Device A outputs the following information as an API Explicit RX Indicator (0x91) frame:

Number of bytes	Field name	Description
8	Destination address	The address the device used to test its link.
2	Payload size	The size of the test packet device A sent to test the link.
2	Iterations	The number of packets that device A sent.
2	Success	The number of packets that were successfully acknowledged.
2	Retries	The number of MAC retries used to transfer all the packets.
1	Result	0x00 - the command was successful. 0x03 - invalid parameter used.
1	RR	The maximum number of MAC retries allowed.
1	maxRSSI	The strongest RSSI reading observed during the test.
1	minRSSI	The weakest RSSI reading observed during the test.
1	avgRSSI	The average RSSI reading observed during the test.

Example

Suppose that you want to test the link between device A (**SH**/**SL** = 0x0013A200 40521234) and device B (**SH**/**SL**=0x0013A 200 4052ABCD) by transmitting 1000 40-byte packets:

Send the following API packet to the serial interface of device A.

In the following example packet, whitespace marks fields, bold text is the payload portion of the packet:

7E 0020 11 01 0013A20040521234 FFFE E6 E6 0014 C105 00 00 **0013A2004052ABCD 0028 03E8** EB

When the test is finished, the following API frame may be received:

7E 0027 91 0013A20040521234 FFFE E6 E6 0094 C105 00 **0013A2004052ABCD 0028 03E8 03E7 0064 00 0A 50 53 52** 9F

This means:

- 999 out of 1000 packets were successful.
- The device made 100 retries.
- **RR** = 10.
- maxRSSI = -80 dBm.
- minRSSI = -83 dBm.
- avgRSSI = -82 dBm.

If the Result field does not equal zero, an error has occurred. Ignore the other fields in the packet.

If the Success field equals zero, ignore the RSSI fields.

The device that sends the request for initiating the Test link and outputs the result does not need to be the sender or receiver of the test. It is possible for a third node, "device C", to request device A to perform a test link against device B and send the results back to device C to be output. It is also possible for device B to request device A to perform the previously mentioned test. In other words, the frames can be sent by either device A, device B or device C and in all cases the test is the same: device A sends data to device B and reports the results.

Trace route option

In many networks, it is useful to determine the route that a DigiMesh unicast takes to its destination, particularly when you set up a network or want to diagnose problems within a network.

Note Because of the large number of Route Information Packet frames that a unicast with trace route enabled can generate, we suggest you only use the trace route option for occasional diagnostic purposes and not for normal operations.

The Transmit Request (0x10 and 0x11) frames contain a trace route option, which transmits routing information packets to the originator of the unicast using the intermediate nodes.

When a device sends a unicast with the trace route option enabled, the unicast transmits to its destination devices, which forward the unicast to its eventual destination. The destination device transmits a Route Information Packet (0x8D) frame back along the route to the unicast originator.

The Route Information Packet frame contains:

- Addressing information for the unicast
- Addressing information for the intermediate hop
- Timestamp
- Other link quality information

For a full description of the Route Information Packet frame, see Route Information - 0x8D.

Trace route example

Suppose that you successfully unicast a data packet with trace route enabled from device A to device E, through devices B, C, and D. The following sequence would occur:

- After the data packet makes a successful MAC transmission from device A to device B, device A outputs a Route Information Packet frame indicating that the transmission of the data packet from device A to device E was successful in forwarding one hop from device A to device B.
- After the data packet makes a successful MAC transmission from device B to device C, device B transmits a Route Information Packet frame to device A. When device A receives the Route Information packet, it outputs it over its serial interface.
- After the data packet makes a successful MAC transmission from device C to device D, device C transmits a Route Information Packet frame to device A (through device B). When device A receives the Route Information packet, it outputs it over its serial interface.
- After the data packet makes a successful MAC transmission from device D to device E, device D transmits a Route Information Packet frame to device A (through device C and device B). When device A receives the Route Information packet, it outputs it over its serial interface.

There is no guarantee that Route Information Packet frames will arrive in the same order as the route taken by the unicast packet. On a weak route, it is also possible for the transmission of Route Information Packet frames to fail before arriving at the unicast originator.

NACK messages

Transmit Request (0x10 and 0x11) frames contain a negative-acknowledge character (NACK) API option (Bit 2 of the Transmit Options field).

If you use this option when transmitting data, when a MAC acknowledgment failure occurs on one of the hops to the destination device, the device generates a Route Information Packet (0x8D) frame and sends it to the originator of the unicast.

This information is useful because it allows you to identify and repair marginal links.

RSSI indicators

The received signal strength indicator (RSSI) measures the amount of power present in a radio signal. It is an approximate value for signal strength received on an antenna.

You can use the **DB** command to measure the RSSI on a device. **DB** returns the RSSI value measured in -dBm of the last packet the device received. This number can be misleading in multi-hop DigiMesh networks. The **DB** value only indicates the received signal strength of the last hop. If a transmission spans multiple hops, the **DB** value provides no indication of the overall transmission path, or the quality of the worst link, it only indicates the quality of the last link.

To determine the **DB** value in hardware:

1. Set **PO** to 1 to enable the RSSI pulse-width modulation (PWM) functionality.
2. Use the DIO10/RSSI/PWM0 module pin (Micro pin 7/SMT pin 7/TH pin 6). When the device receives data, it sets the RSSI PWM duty cycle to a value based on the RSSI of the packet it receives.

This value only indicates the quality of the last hop of a multi-hop transmission. You could connect this pin to an LED to indicate if the link is stable or not.

Associate LED

The Associate pin (Micro pin 26/SMT pin 28) provides an indication of the device's status. To take advantage of these indications, connect an LED to the Associate pin.

To enable the Associate LED functionality, set the **D5** command to 1; it is enabled by default. If enabled, the Associate pin is configured as an output. This section describes the behavior of the pin.

The pin functions as a power indicator.

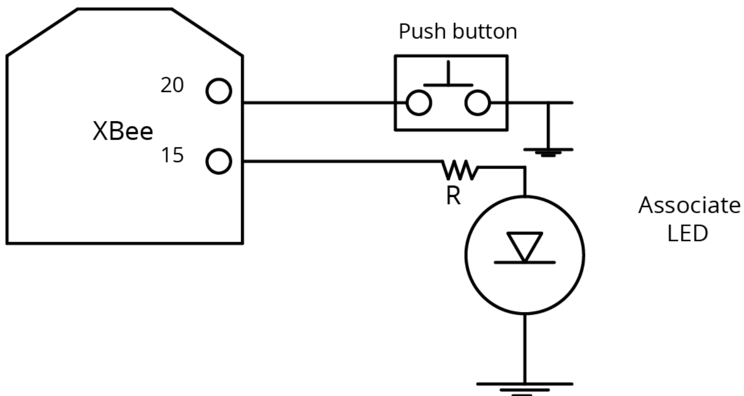
Use the **LT** command to override the blink rate of the Associate pin. If you set **LT** to 0, the device uses the default blink time of 250 ms.

The following table describes the Associate LED functionality.

LED Status	Meaning
On, blinking	The device has power and is operating properly

The Commissioning Pushbutton

The XBee 3 DigiMesh RF Module supports a set of commissioning and LED functions to help you deploy and commission devices. These functions include the Commissioning Pushbutton definitions and the associated LED functions. The following diagram shows how the hardware can support these features.



To support the Commissioning Pushbutton and its associated LED functions, connect a pushbutton and an LED to device pins 20 and 15 respectively.

Definitions

To enable the Commissioning Pushbutton functionality on pin 20, set the **D0** command to 1. The functionality is enabled by default.

You must perform the designated number of button presses within two seconds. If any number of commissioning button presses occur while the device is asleep, it will wake up until the sleep cycle is finished or for 30 seconds, whichever occurs first.

The following table provides the pushbutton definitions.

Button presses	Action
1	Sends a Node Identification broadcast transmission. All devices that receive this transmission blink their Associate LED rapidly for one second. Additionally, receiving devices that are operating in API mode also send a Node Identification frame (0x95) out their UART.
2	This function only applies for synchronous sleep networks. Two button presses nominate a node as the sleep coordinator by sending out a sync message. If the sending node has seniority over the current sleep coordinator, the sending node becomes the sleep coordinator. Otherwise, the current sleep coordinator retains that role.
4	Restores the node to default configuration. If custom defaults are in use, they will be applied on top of the factory defaults. Unlike RE (Restore Defaults), this function not only restores the default configuration, but it also applies those changes.

Use the Commissioning Pushbutton

Use the **CB** command to simulate button presses in software. Send **CB** with a parameter set to the number of button presses to perform. For example, if you send **ATCB1**, the device performs the action (s) associated with a single button press.

Node Identification Indicator - 0x95 is similar to Remote AT Command Response- 0x97 – it contains the device's address, node identifier string (**NI** command), and other relevant data. All devices in API operating mode that receive the Node Identification Indicator frame send it out their UART as a Node Identification Indicator frame.

Node discovery

Node discovery has three variations as shown in the following table:

Commands	Syntax	Description
Node Discovery	**ND**	Seeks to discover all nodes in the network (on the current Network ID).
Directed Node Discovery	**ND** <NI String>	Seeks to discover if a particular node named <NI String> is found in the network.
Destination Node	**DN** <NI String>	Sets **DH**/**DL** to point to the MAC address of the node whose <NI String> matches.

The node discovery command (without an NI string designated) sends out a broadcast to every node in the Network ID. Each node in the network sends a response back to the requesting node.

When the node discovery command is issued in Command mode, all other AT commands are inhibited until the node discovery command times out, as determined by the **N?** parameter. After the timeout, an extra CR is output to the terminal window, indicating that new AT commands can be entered. This is the behavior whether or not there were any nodes that responded to the broadcast.

When the node discovery command is issued in API mode, the behavior is the same except that the response is output in API mode. If no nodes respond, there will be no responses at all to the node discover command. The requesting node is not able to process a new AT command until **N?** times out.

Discover all the devices on a network

You can use the **ND** (Network Discovery) command to discover all devices on a network. When you send the **ND** command:

1. The device sends a broadcast **ND** command through the network.
2. All devices that receive the command send a response that includes their addressing information, node identifier string and other relevant information. For more information on the node identifier string, see NI (Network Identifier).

ND is useful for generating a list of all device addresses in a network.

When a device receives the network discovery command, it waits a random time before sending its own response. You can use the **NT** command to set the maximum time delay on the device that you use to send the **ND** command.

- The device that sends the **ND** includes its **NT** setting in the transmission to provide a random delay window for all devices in the network. When devices respond at random intervals during the **NT** window, fewer collisions occur and more responses can be obtained.
- The default **NT** value is 0x82 (13 seconds).

Directed node discovery

The directed node discovery command (**ND** with an **NI** string parameter) sends out a broadcast to find a node in the network with a matching **NI** string. If such a node exists, it sends a response with its information back to the requesting node.

In Transparent mode, the requesting node outputs an extra carriage return following the response from the designated node and the command terminates; it is then ready to accept a new AT command. In the event that the requested node does not exist or is too slow to respond, the requesting node outputs an ERROR response after **N?** expires.

In API mode, the response from the requesting node will be output in API mode and the command will terminate immediately. If no response comes from the requested node, the requesting node outputs an error response in API mode after **N?** expires. The device's software assumes that each node has a unique **NI** string.

The directed node discovery command terminates after the first node with a matching **NI** string responds. If that **NI** string is duplicated in multiple nodes, the first responding node may not always be the same node or the desired node.

Destination Node

The Destination Node command (**DN** with an **NI** string parameter) sends out a broadcast containing the **NI** string being requested. The responding node with a matching **NI** string sends its information back to the requesting node. The local node then sets **DH**/**DL** to match the address of the responding node. As soon as this response occurs, the command terminates successfully. If the device is in AT Command mode, an OK string is output and Command mode exits. In API mode, you may enter another AT command.

If an **NI** string parameter is not provided, the **DN** command terminates immediately with an error. If a node with the given NI string does not respond, the **DN** command terminates with an error after **N?** times out.

In Transparent mode, unlike **ND** (with or without an **NI** string), **DN** does not cause the information from the responding node to be output; rather it simply sets **DH**/**DL** to the address of the responding node.

In API mode, the response from the requesting node outputs in API mode and the command terminates immediately. If no response comes from the requested node, the requesting node outputs an error response in API mode after **N?** expires.

The device's software assumes that each node has a unique **NI** string. The directed destination node command terminates after the first node with a matching **NI** string responds. If that **NI** string is duplicated in multiple nodes, **DH**/**DL** may not be set to the desired value.

Discover devices within RF range

The **FN** (Find Neighbor) command works the same as the **ND** (Node Discovery) except that it is limited to neighboring devices (devices that are only one hop away). See FN (Find Neighbors) for details.

- You can use the **FN** (Find Neighbors) command to discover the devices that are immediate neighbors (within RF range) of a particular device.
- **FN** is useful in determining network topology and determining possible routes.

You can send **FN** locally on a device in Command mode or you can use a local Local AT Command Request - 0x08.

To use **FN** remotely, send the target node a Remote AT Command Request - 0x17 using **FN** as the name of the AT command.

The device you use to send **FN** transmits a zero-hop broadcast to all of its immediate neighbors. All of the devices that receive this broadcast send an RF packet to the device that transmitted the **FN** command. If you sent **FN** remotely, the target devices respond directly to the device that sent the **FN** command. The device that sends **FN** outputs a response packet in the same format as an Local AT Command Response - 0x88.

Sleep support

Sleep is implemented to support installations where a mains power source is not available and a battery is required. In order to increase battery life, the device sleeps, which means it stops operating. It can be woken by a timer expiration or a pin.

Sleep modes

A number of low-power modes exist to enable devices to operate for extended periods of time on battery power. Use SM (Sleep Mode) to enable these sleep modes. The sleep modes are characterized as either:

- Asynchronous (**SM** = **1**, **4**, **5**, **6**).
- Synchronous (**SM** = **7**, **8**).

In Synchronous sleep networks, a device functions in one of three roles:

1. A sleep coordinator.
2. A potential coordinator.
3. A non-coordinator.

The difference between a potential coordinator and a non-coordinator is that a non-coordinator node has its SO (Sleep Options) parameter set so that it will not participate in coordinator nomination and election and cannot ever be a sleep coordinator.

Note Synchronous and asynchronous sleep modes are incompatible. Synchronous and asynchronous sleep nodes should not be configured in the same network. Asynchronous sleep does not apply in a mesh network. It can only operate over one hop where a designated node holds messages for the sleeping node.

Asynchronous sleep modes

Use the asynchronous sleep modes to control the sleep state on a device by device basis.

Do not use devices operating in asynchronous sleep mode to route data.

We strongly encourage you to set asynchronous sleeping devices as end-devices using CE (Routing / Messaging Mode). This prevents the node from attempting to route data.

Asynchronous Pin Sleep mode (SM = 1)

Pin Sleep mode minimizes quiescent power (power consumed when in a state of rest or inactivity). In order to use Pin Sleep mode, configure SM (Sleep Mode) to **1** and configure D8 (DIO8/DTR/SLP_Request Configuration) (Micro pin 9/SMT pin 10) for DTR/SLEEP_RQ input (**D8** = **1**). This mode is voltage level-activated; when SLEEP_RQ is asserted, the device finishes any transmit or receive activities, enters Idle mode, and then enters a state of sleep. The device does not respond to either serial or RF activity while in pin sleep.

To wake a sleeping device operating in Pin Sleep mode, de-assert DTR/SLEEP_RQ. The device wakes when SLEEP_RQ is de-asserted and is ready to transmit or receive when the CTS line is low.

Devices with SPI functionality can use the SPI_SSEL pin instead of **D8** for pin sleep control. If **D8** = **0** and **P7** = **1**, SPI_SSEL takes the place of DTR/SLEEP_RQ and functions as described above. In order to use SPI_SSEL for sleep control while communicating on the UART, SPI pins—**P5**, **P6**, and **P8**—must not be set to 1 (peripheral). See Low power operation for information on using SPI_SSEL for sleep control while communicating over SPI.

Asynchronous Cyclic Sleep mode (SM = 4)

The Cyclic Sleep modes allow devices to periodically check for RF data. When the **SM** parameter is set to **4**, the XBee 3 DigiMesh RF Module is configured to sleep, then wakes once per cycle to check for data from a coordinator. The Cyclic Sleep Remote sends a poll request to the coordinator at a specific interval set by SP (Cyclic Sleep Period). The coordinator transmits any queued data addressed to that specific remote upon receiving the poll request.

If no data is queued for the remote, the messaging coordinator does not transmit and the remote returns to sleep for another cycle. If queued data is transmitted back to the remote, it stays awake to allow for back and forth communication until the ST (Cyclic Sleep Wake Time) timer expires. You can also set SO (Sleep Options) bit 8 to force the device to always wake for the full **ST** time.

To configure a node to act as a coordinator **CE** must be set to 1. A sleeping node also needs to be configured to know which node is its coordinator. This is done by setting the **DH** and **DL** of the sleeper to the **SH** and **SL** of the coordinator node. **CE** on the sleeper node must also be set to **4**. In order for the coordinator to queue transmissions meant for the sleeping node any transmissions sent to the sleeping node must be sent using 0x40 (point-to-point) **TO** options.

If configured, CTS goes low each time the remote wakes, allowing for communication initiated by the remote host if desired. If ON_SLEEP is configured it goes high (ON) after SN (Number of Sleep Periods) sleep periods. Change **SN** to allow external circuitry to sleep for longer periods if no data is received.

Asynchronous Cyclic Sleep with Pin Wake-up mode (SM = 5)

Use this mode to wake a sleeping remote device through either the RF interface or by asserting (low) DTR/SLEEP_RQ for event-driven communications. The cyclic sleep mode works as described previously with the addition of a pin-controlled wake-up at the remote device.

The DTR/SLEEP_RQ pin is level-triggered. The device wakes when a low is detected then sets CTS low as soon as it is ready to transmit or receive. The device stays awake as long as DTR/SLEEP_RQ is low; once DTR/SLEEP_RQ goes high the device returns to cyclic sleep operation. If DTR/SLEEP_RQ is momentarily pulsed low, the minimum wake time is ST (Cyclic Sleep Wake Time) even if DTR/SLEEP_RQ is low for less time.

Once awake, any activity resets the ST (Cyclic Sleep Wake Time) timer, so the device goes back to sleep only after there is no RF activity for the duration of the timer.

MicroPython sleep with optional pin wake (SM = 6)

The MicroPython sleep option allows a user's MicroPython program to exclusively control the device's sleep operation (with optional pin wake). For full details refer to the *Digi MicroPython Programming Guide*.

Synchronous sleep modes

Synchronous sleep makes it possible for all nodes in the network to synchronize their sleep and wake times. All synchronized cyclic sleep nodes enter and exit a low power state at the same time. This allows all or most devices in a network to use low power because, unlike Zigbee, low power devices do not need to be adjacent to mains powered devices.

Synchronous sleep forms a cyclic sleeping network with these features:

- A device acting as a sleep coordinator sends a special RF packet called a sync message to synchronize nodes.
- To make a device in the network a coordinator, a node uses several resolution criteria.
- The sleep coordinator sends one sync message at the beginning of each wake period. The coordinator sends the sync message as a broadcast and every routing node in the network repeats it.
- You can change the sleep and wake times for the entire network by locally changing the settings on an individual device. The network uses the most recently set sleep settings.

Synchronous sleep support mode (SM = 7)

Note Sleep support nodes should be mains powered because they do not sleep.

Set **SM** to **7** to enter synchronous sleep support mode.

A device in synchronous sleep support mode synchronizes itself with a sleeping network but will not itself sleep. At any time, the device responds to new devices that are attempting to join the sleeping network with a sync message. A sleep support device only transmits normal data when the other devices in the sleeping network are awake. You can use sleep support devices as sleep coordinator devices and as aids in adding new devices to a sleeping network.

Synchronous cyclic sleep mode (SM = 8)

Set **SM** to 8 to enter synchronous cyclic sleep mode.

A device in synchronous cyclic sleep mode sleeps for a programmed time, wakes in unison with other nodes, exchanges data and sync messages, and then returns to sleep. While asleep, it cannot receive RF messages or receive data (including commands) from the UART port.

Generally, the network's sleep coordinator specifies the sleep and wake times based on its **SP** and **ST** settings. The device only uses these parameters at startup until the device synchronizes with the network.

When a device has synchronized with the network, you can query its sleep and wake times with the **OS** and **OW** commands respectively.

If **D9** = 1 (ON_$\overline{\text{SLEEP}}$ enabled) on a cyclic sleep node, the ON_$\overline{\text{SLEEP}}$ line goes high when the device is awake and goes low when the device is asleep.

If **D7** = 1, the device de-asserts $\overline{\text{CTS}}$ while asleep.

A newly-powered, unsynchronized, sleeping device polls for a synchronized message and then sleeps for the period that the **SP** command specifies, repeating this cycle until it synchronizes by receiving a sync message. Once it receives a sync message, the device synchronizes itself with the network.

Note Configure all nodes in a synchronous sleep network to operate in either synchronous sleep support mode or synchronous cyclic sleep mode. asynchronous sleeping nodes are not compatible with synchronous sleeping nodes.

Sleep parameters

The following AT commands are associated with the sleep modes. See the linked commands for the parameter's description, range and default values.

- SM (Sleep Mode)
- SN (Number of Sleep Periods)
- SO (Sleep Options)
- ST (Cyclic Sleep Wake Time)
- SP (Cyclic Sleep Period)
- WH (Wake Host Delay)

Sleep pins

The following table describes the three external device pins associated with sleep.

See the *XBee 3 RF Module Hardware Reference Manual* for the pinout of your device.

Pin name	Pin number	Description
DTR/SLEEP_RQ	Micro pin 9/SMT pin 10	For **SM** = **1**, high puts the device to sleep and low wakes it up. For **SM** = **5**, a high to low transition wakes the device up for **ST** time. The device ignores a low to high transition in **SM** = **5**.
SPI_SSEL	Micro pin 14/SMT pin 15	This pin operates the same as SLEEP_RQ when **D8** is **0**.
CTS	Micro pin 24/SMT pin 25	If **D7** = **1**, high indicates that the device is asleep and low indicates that it is awake and ready to receive serial data.
ON_SLEEP	Micro pin 25/SMT pin 26	Low indicates that the device is asleep and high indicates that it is awake and ready to receive serial data.

Sleep conditions

Since instructions stop executing while the device is sleeping, it is important to avoid sleeping when the device has work to do. For example, the device will not sleep if any of the following are true:

1. The device is operating in Command mode, or in the process of getting into Command mode with the **+++** sequence.
2. The device is processing AT commands from API mode
3. The device is processing remote AT commands
4. Something is queued to the serial port and that data is not blocked by RTS flow control

If each of the above conditions are false, then sleep may still be blocked in these cases:

1. Enough time has not expired since the device has awakened.
   a. If the device is operating in pin sleep, the amount of time needed for one character to be received on the UART is enough time.
   b. If the device is operating in asynchronous cyclic sleep, enough time is defined by a timer. The duration of that timer is:
      i. defined by **ST** if in **SM** 5 mode and it is awakened by a pin
      ii. 30 ms to allow enough time for a poll and a poll response
   c. In addition, if the device is operating in Asynchronous Cyclic Sleep, the wake time is extended by an additional **ST** time when new OTA data or serial data is received.
2. Sleep Request pin is not asserted when operating in pin sleep mode
3. Data is waiting to be sent OTA.

The sleep timer

If the device receives serial or RF data in Asynchronous cyclic sleep mode and Asynchronous cyclic sleep with pin wake up modes (**SM** = 4 or **SM** = 5), it starts a sleep timer (time until sleep).

- If the device receives any data serially or by RF link, the timer resets.
- Use ST (Cyclic Sleep Wake Time) to set the duration of the timer.
- When the sleep timer expires the device returns to sleep.

Sleep coordinator sleep modes in the network

In a synchronized sleeping network, one node acts as the sleep coordinator. During normal operations, at the beginning of a wake cycle the sleep coordinator sends a sync message as a broadcast to all nodes in the network. This message contains synchronization information and the wake and sleep times for the current cycle. All cyclic sleep nodes that receive a sync message remain awake for the wake time and then sleep for the specified sleep period.

The sleep coordinator sends one sync message at the beginning of each cycle with the current wake and sleep times. All router nodes that receive this sync message relay the message to the rest of the network. If the sleep coordinator does not hear a rebroadcast of the sync message by one of its immediate neighbors, then it re-sends the message one additional time.

If you change the **SP** or **ST** parameters, the network does not apply the new settings until the beginning of the next wake time. For more information, see Change sleep parameters.

A sleeping router network is robust enough that an individual node can go several cycles without receiving a sync message, due to RF interference, for example. As a node misses sync messages, the time available for transmitting messages during the wake time reduces to maintain synchronization accuracy. By default, a device reduces its active sleep time progressively as it misses consecutive sync messages.

Synchronization messages

A sleep coordinator regularly sends sync messages to keep the network in sync. Unsynchronized nodes also send messages requesting sync information.

Sleep compatible nodes use Deployment mode when they first power up and the sync message has not been relayed. A sleep coordinator in Deployment mode rapidly sends sync messages until it receives a relay of one of those messages. Deployment mode:

- Allows you to effectively deploy a network.
- Allows a sleep coordinator that resets to rapidly re-synchronize with the rest of the network.

If a node exits deployment mode and then receives a sync message from a sleep coordinator that is in Deployment mode, it rejects the sync message and sends a corrective sync to the sleep coordinator.

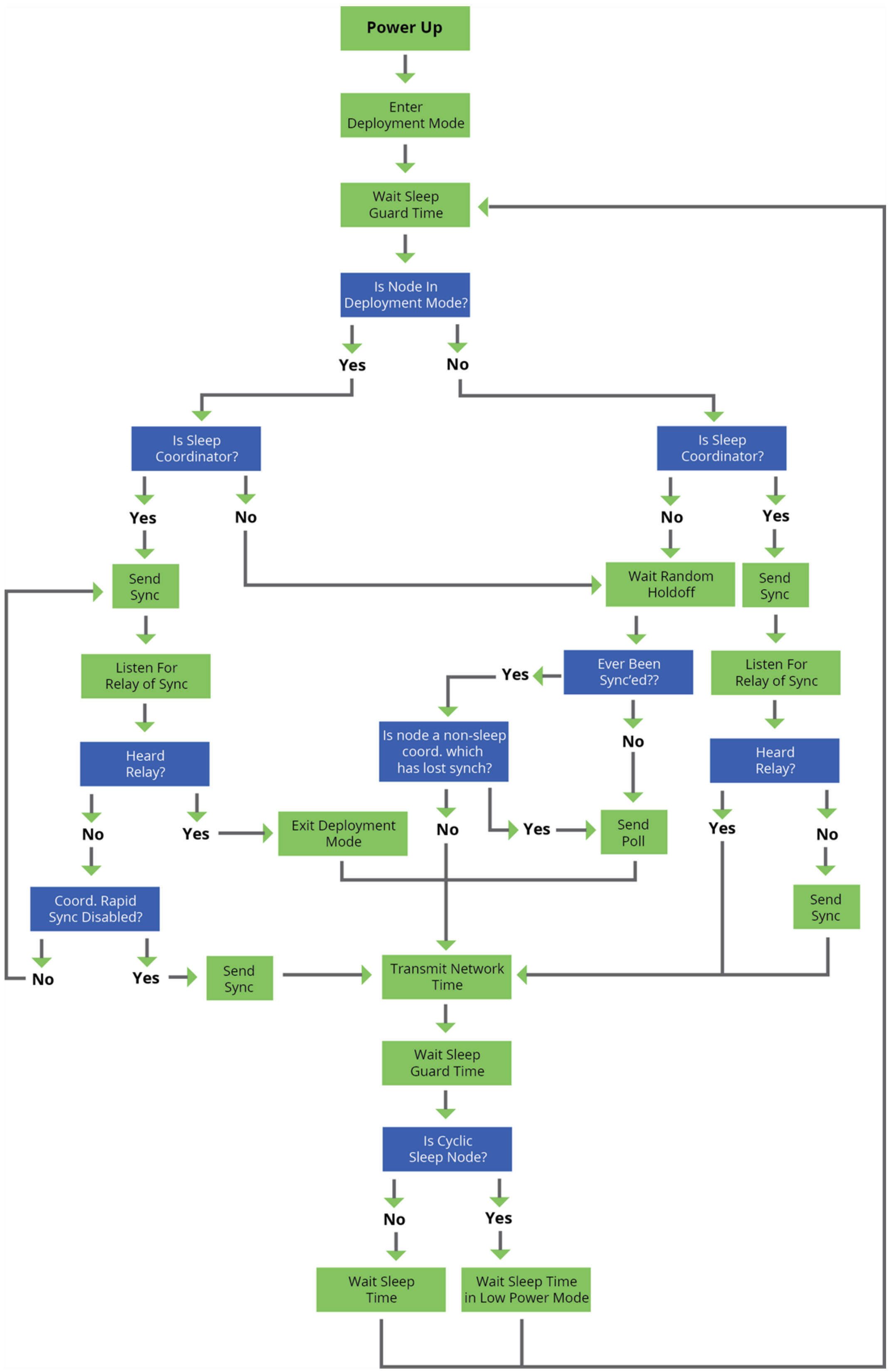
Use the **SO** (sleep options) command to disable deployment mode. This option is enabled by default.

A sleep coordinator that is not in deployment mode sends a sync message at the beginning of the wake cycle. The sleep coordinator listens for a neighboring node to relay the sync. If it does not hear the relay, the sleep coordinator sends the sync one additional time.

A node that is not a sleep coordinator and has never been synchronized sends a message requesting sync information at the beginning of its wake cycle. Synchronized nodes which receive one of these messages respond with a synchronization packet.

If you use the **SO** command to configure nodes as non-coordinators, and if the non-coordinators go six or more sleep cycles without hearing a sync, they send a message requesting sync at the beginning of their wake period.

The following diagram illustrates the synchronization behavior of sleep compatible devices.


Power Up
Enter Deployment Mode
Wait Sleep Guard Time
Is Node In Deployment Mode?
Yes
No
Is Sleep Coordinator?
Is Sleep Coordinator?
Yes
No
No
Yes
Send Sync
Wait Random Holdoff
Send Sync
Listen For Relay of Sync
Ever Been Sync'ed??
Listen For Relay of Sync
Yes
Is node a non-sleep coord. which has lost synch?
No
Heard Relay?
Heard Relay?
No
Yes
Exit Deployment Mode
No
Yes
Send Poll
Yes
No
Coord. Rapid Sync Disabled?
Send Sync
No
Yes
Send Sync
Transmit Network Time
Wait Sleep Guard Time
Is Cyclic Sleep Node?
No
Yes
Wait Sleep Time
Wait Sleep Time in Low Power Mode

Become a sleep coordinator

In DigiMesh networks, a device can become a sleep coordinator in one of four ways:

- Define a sleep coordinator
- A potential sleep coordinator misses three or more sync messages
- Press the Commissioning Pushbutton twice on a potential sleep coordinator
- Change the sleep timing values on a potential sleep coordinator

Set the sleep coordinator option

You can specify that a node will always act as a sleep coordinator. To do this, set the sleep coordinator bit (bit 0) in the **SO** command to **1**.

A node with the sleep coordinator bit set always sends a sync message at the beginning of a wake cycle. To avoid network congestion and synchronization conflicts, do not set this bit on more than one node in the network.

A node that is centrally located in the network can serve as a good sleep coordinator, because it minimizes the number of hops a sync message takes to get across the network.

A sleep support node and/or a node that is mains powered is a good candidate to be a sleep coordinator.

CAUTION! Use the sleep coordinator bit with caution. The advantages of using the option become weaknesses if you use it on a node that is not in the proper position or configuration. Also, it is not valid to have the sleep coordinator option bit set on more than one node at a time.

You can also use the sleep coordinator option when you set up a network for the first time. When you start a network, you can configure a node as a sleep coordinator so it will begin sending sleep messages. After you set up the network, we recommend that you disable the sleep coordinator bit.

Resolution criteria and selection option

There is an automatic selection process with resolution criteria that occurs on a node if it loses contact with the network sleep coordinator.

A sleep compatible node may become a sleep coordinator if it:

- Misses three or more sync messages and it:
- Is not configured as a non-coordinator by setting bit 1 of **SO**.

If such a node wins out on the selection process, it becomes the new network sleep coordinator.

It is possible for multiple nodes to declare themselves as the sleep coordinator. If this occurs, the firmware uses the following resolution criteria to identify the sleep coordinator from among the nodes using the selection process:

1. Newer sleep parameters: the network considers a node using newer sleep parameters (**SP** and **ST**) as higher priority than a node using older sleep parameters. See Commissioning Pushbutton option. Note that when **SP** and/or **ST** is changed, it increments the sequence number such that it sends the newest sync message and it has priority to become the sleep coordinator.

2. Sleep coordinator: a node configured as the sleep coordinator is higher priority than other nodes.
3. Sleep support node: sleep support nodes are higher priority than cyclic sleep nodes. You can modify this behavior using the **SO** parameter.
4. Serial number: If the previous factors do not resolve the priority, the network considers the node with the higher serial number to be higher priority.

Commissioning Pushbutton option

If you enable the Commissioning Pushbutton functionality, you can immediately select a device as a sleep coordinator by pressing the Commissioning Pushbutton twice or by issuing the **CB2** command. The device you select in this manner is still subject to the resolution criteria process.

Only sleep coordinator nodes honor Commissioning Pushbutton nomination requests. A node configured as a non-sleep coordinator ignores commissioning button nomination requests.

Overriding syncs

Any sleep compatible node in the network that does not have the non-coordinator sleep option set can send an overriding sync and become the network sleep coordinator. An overriding sync effectively changes the synchronization of all nodes in the network to the **ST** and **SP** values of the node sending the overriding sync. It also selects the node sending the overriding sync as the network sleep coordinator. While this is a powerful operation, it may be an undesired side effect because the current sleep coordinator may have been carefully selected and it is not desired to change it. Additionally the current wake and sleep cycles may be desired rather than the parameters on the node sending the overriding sync. For this reason, it is important to know what kicks off an overriding sync.

An overriding sync occurs whenever **ST** or **SP** is changed to a value different than **OW** or **OS** respectively. For example no overriding sync will occur if **SP** is changed from 190 to C8 if the network was already operating with **OS** at C8. On the other hand, if **SP** is changed from 190 to 190 (meaning no change), and **OS** is C8, than an overriding sync will occur because the network parameters are being changed.

Even parameters that seem unrelated to sleep can kick off an overriding sync. These are **NH**, **NN**, **RN**, and **MT**. When any of these parameters are changed, they can affect network traversal time. If such changes cause the configured value of **ST** to be smaller than the value needed for network traversal, then **ST** is increased and if that increased value is different than **OW**, then an overriding sync will occur.

For most applications, we recommend configuring the **NH**, **NN**, **RN**, and **MT** network parameters during initial deployment only. The default values of **NH** and **NN** are optimized to work for most deployments. Additionally, it would be best to set **ST** and **SP** the same on all nodes in the network while keeping **ST** sufficiently large so that it won’t be affected by an inadvertent change of **NH**, **NN**, **RN**, or **MT**.

Sleep guard times

To compensate for variations in the timekeeping hardware of the various devices in a sleeping router network, the network allocates sleep guard times at the beginning and end of the wake period. The size of the sleep guard time varies based on the sleep and wake times you select and the number of sleep cycles that elapse since receiving the last sync message. The sleep guard time guarantees that a destination module will be awake when the source device sends a transmission. As a node misses more and more consecutive sync messages, the sleep guard time increases in duration and decreases the available transmission time.

Auto-early wake-up sleep option

If you have nodes that are missing sync messages and could be going out of sync with the rest of the network, enabling an early wake gives the device a better chance to hear the sync messages that are being broadcast.

Similar to the sleep guard time, the auto early wake-up option decreases the sleep period based on the number of sync messages a node misses. This option comes at the expense of battery life.

Use bit 3 of the **SO** command to disable auto-early wake-up sleep.

This option is enabled by default.

Select sleep parameters

Choosing proper sleep parameters is vital to creating a robust sleep-enabled network with a desirable battery life. To select sleep parameters that will be good for most applications, follow these steps:

1. Choose **NN** and **NH**.

   Based on the placement of the nodes in your network, select the appropriate values for the **NH** (Network Hops) and **NN** (Network Delay Slots) parameters.

   We optimize the default values of **NH** and **NN** to work for the majority of deployments. In most cases, we suggest that you do not modify these parameters from their default values. Decreasing these parameters for small networks can improve battery life, but take care to not make the values too small.
2. Calculate the Sync Message Propagation Time (SMPT).

   This is the maximum amount of time it takes for a sleep synchronization message to propagate to every node in the network. You can estimate this number with the following formula:

   SMPT = **NH** * (**MT** + 1) * 4 ms.

   Note The 4 msec constant applies to XBee 3 DigiMesh, but it is different for every platform on which DigiMesh runs.
3. Select the duty cycle you want.
   The ratio of sleep time to wake time is the factor that has the greatest effect on the device's power consumption. Battery life can be estimated based on the following factors:
   - sleep period
   - wake time
   - sleep current
   - RX current
   - TX current
   - battery capacity
4. Choose the sleep period and wake time.

   The wake time must be long enough to transmit the desired data as well as the sync message. The **ST** parameter automatically adjusts upwards to its minimum value when you change other AT commands that affect it (**SP**, **NN**, and **NH**).

   Use a value larger than this minimum. If a device misses successive sync messages, it reduces its available transmit time to compensate for possible clock drift. Budget a large enough **ST** time to allow for the device to miss a few sync messages and still have time for normal data transmissions.

Sleep immediate

In order to ensure that the needed messages have time to traverse the network, **ST** must be sufficiently large. Additionally, your application is a factor in determining what **ST** should be. When **ST**/**SP** increases, the batteries burn out faster. Yet, **ST** must be large enough for a functional network.

To mitigate this problem, the Sleep Immediate command is available (in version **0x300B** and newer). The Sleep Immediate (**SI**) command can be sent by your application after it determines that all needed transmissions are completed. This not only puts the node that issues the command asleep, but it also sends a broadcast to put the whole network to sleep. The network will then remain asleep for the remainder of the wake time and the subsequent sleep time. Then the entire network will awaken again, resuming the same sleep cycle as before.

In the event that one or more nodes fail to receive the sleep immediate broadcast, they will not get the power savings, but they will still remain synchronized to the network because the sleep cycle would not have changed.

Start a sleeping synchronous network

By default, all new nodes operate in normal (non-sleep) mode. To start a synchronous sleeping network, follow these steps:

1. Set **SO** to 1 to enable the sleep coordinator option on one of the nodes.
2. Set its **SM** to a synchronous sleep compatible mode (7 or 8) with its **SP** and **ST** set to a quick cycle time. The purpose of a quick cycle time is to allow the network to send commands quickly through the network during commissioning.
3. Power on the new nodes within range of the sleep coordinator. The nodes quickly receive a sync message and synchronize themselves to the short cycle **SP** and **ST** set on the sleep coordinator.
4. Configure the new nodes to the sleep mode you want, either cyclic sleeping modes or sleep support modes.
5. Set the **SP** and **ST** values on the sleep coordinator to the values you want for the network.
6. In order to reduce the possibility of an unintended overriding sync, set **SP** and **ST** to the intended sleep/wake cycle on all nodes in the network. Be sure that **ST** is large enough to prevent it from being inadvertently increased by changing **NN**, **NH**, or **MT**.
7. Wait a sleep cycle for the sleeping nodes to sync themselves to the new **SP** and **ST** values.
8. Disable the sleep coordinator option bit on the sleep coordinator unless you want to force a particular sleep coordinator.
9. Deploy the nodes to their positions.

Alternatively, prior to deploying the network you can use the **WR** command to set up nodes with their sleep settings pre-configured and written to flash. If this is the case, you can use the Commissioning Pushbutton and associate LED to aid in deployment:

1. If you are going to use a sleep coordinator in the network, deploy it first.
2. If more than one node can be the sleep coordinator, select a node for deployment, power it on and press the Commissioning Pushbutton twice. This causes the node to begin emitting sync messages.
3. Verify that the first node is emitting sync messages by watching its associate LED. A slow blink indicates that the node is acting as a sleep coordinator.

4. Power on nodes in range of the sleep coordinator or other nodes that have synchronized with the network. If the synchronized node is asleep, you can wake it by pressing the Commissioning Pushbutton once.
5. Wait a sleep cycle for the new node to sync itself.
6. Verify that the node syncs with the network. The associate LED blinks when the device is awake and synchronized.
7. Continue this process until you deploy all of the nodes.

Add a new node to an existing network

To add a new node to the network, the node must receive a sync message from a node already in the network. On power-up, an unsynchronized, sleep compatible node periodically sends a broadcast requesting a sync message and then sleeps for its **SP** period. Any node in the network that receives this message responds with a sync. Because the network can be asleep for extended periods of time, and cannot respond to requests for sync messages, there are methods you can use to sync a new node while the network is asleep.

1. Power the new node on within range of a sleep support node. Sleep support nodes are always awake and able to respond to sync requests promptly.
2. You can wake a sleeping cyclic sleep node in the network using the Commissioning Pushbutton. Place the new node in range of the existing cyclic sleep node. Wake the existing node by pressing the Commissioning Pushbutton once. The existing node stays awake for 30 seconds and responds to sync requests while it is awake.

If you do not use one of these two methods, you must wait for the network to wake up before adding the new node.

Place the new node in range of the network with a sleep/wake cycle that is shorter than the wake period of the network.

The new node periodically sends sync requests until the network wakes up and it receives a sync message.

Change sleep parameters

To change the sleep and wake cycle of the network, select any sleep coordinator capable node in the network and change the **SP** and/or **ST** of the node to values different than those the network currently uses.

- If you configure a particular sleep coordinator or if you know which node acts as the sleep coordinator, we suggest that you use this node to make changes to network settings.
- If you do not know the network sleep coordinator, you can use any node that does not have the non-sleep coordinator sleep option bit set. For details on the bit, see SO (Sleep Options).

When you make changes to a node's **SP** and/or **ST** parameters and that node does not have the non-sleep coordinator option set then:

- That node broadcasts an overriding sync to the network to advertise the new sleep cycle.
- That node nominates itself to become the sleep coordinator.
- That node will remain the sleep coordinator unless another node in the network designates itself as the sleep coordinator.
- The network will apply the new sleep parameters at the beginning of the next wake cycle.

Changing sleep parameters increases the chances that nodes will lose sync. If a node does not receive the sync message with the new sleep settings, it continues to operate on its old settings. To minimize the risk of a node losing sync and to facilitate the re-syncing of a node that does lose sync, take the following precautions:

1. Whenever possible, avoid changing sleep parameters.
2. Enable the missed sync early wake up sleep option in the **SO** command. This option is enabled by default. This command tells a node to wake up progressively earlier based on the number of cycles it goes without receiving a sync. This increases the probability that the un-synced node will be awake when the network wakes up and sends the sync message.

Note Using this sleep option increases reliability but may decrease battery life. Nodes using this sleep option that miss sync messages increase their wake time and decrease their sleep time during cycles where they miss the sync message. This increases power consumption.

When you are changing between two sets of sleep settings, choose settings so that the wake periods of the two sleep settings occur at the same time. In other words, try to satisfy the following equation:

$$(SP_1 + ST_1) = N * (SP_2 + ST_2)$$

where SP_1/ST_1 and SP_2/ST_2 are the desired sleep settings and *N* is an integer.

Rejoin nodes that lose sync

DigiMesh networks get their robustness from routing redundancies which may be available. We recommend architecting the network with redundant mesh nodes to increase robustness.

If a scenario exists where the only route connecting a subnet to the rest of the network depends on a single node, and that node fails or the wireless link fails due to changing environmental conditions (a catastrophic failure condition), then multiple subnets may arise using the same wake and sleep intervals. When this occurs the first task is to repair, replace, and strengthen the weak link with new and/or redundant devices to fix the problem and prevent it from occurring in the future.

If a network has multiple subnets that drift out of phase with each other, get the subnets back in phase with the following steps:

1. Place a sleep support node in range of both subnets.
2. Select a node in the subnet that you want the other subnet to sync with.
3. Use this node to slightly change the sleep cycle settings of the network, for example, increment **ST**.
4. Wait for the subnet’s next wake cycle. During this cycle, the node you select to change the sleep cycle parameters sends the new settings to the entire subnet it is in range of, including the sleep support node that is in range of the other subnet.
5. Wait for the out of sync subnet to wake up and send a sync. When the sleep support node receives this sync, it rejects it and sends a sync to the subnet with the new sleep settings.
6. The subnets will now be in sync. You can remove the sleep support node.
7. You can also change the sleep cycle settings back to the previous settings.

If you only need to replace a few nodes, you can use this method:

1. Reset the out of sync node and set its sleep mode to Synchronous Cyclic Sleep mode (**SM** = **8**).
2. Set up a short sleep cycle.

3. Place the node in range of a sleep support node or wake a sleeping node with the Commissioning Pushbutton.
4. The out of sync node receives a sync from the node that is synchronized to the network. It then syncs to the network sleep settings.

Diagnostics

The following diagnostics are useful in applications that manage a sleeping router network:

Query sleep cycle

Use the **OS** and **OW** commands to query the current operational sleep and wake times that a device uses.

Sleep status

Use the **SS** command to query useful information regarding the sleep status of the device. Use this command to query if the node is currently acting as a network sleep coordinator.

Missed sync messages command

Use the **MS** command to query the number of cycles that elapsed since the device received a sync message.

Sleep status API messages

When you use the **SO** command to enable this option, a device that is in API operating mode outputs modem status frames immediately after it wakes up and prior to going to sleep.

AT commands

Networking commands

The following commands affect the DigiMesh network.

CH (Operating Channel)

The operating channel devices use to transmit and receive data.

In order for devices to communicate with each other, they must share the same channel number. A network can use different channels to prevent devices in one network from listening to the transmissions of another and to reduce interference.

The command uses IEEE 802.15.4 channel numbers.

Parameter range

0xB - 0x1A

Default

0xC (channel 12)

ID (Network ID)

The device's PAN (Personal Area Network) identifier. PAN IDs allows for the logical separation of multiple networks that share the same RF channel.

In order for devices to communicate, they must be configured with the same PAN ID and channel.

Parameter range

0 - 0xFFFF

Default

0x7FFF

CE (Routing / Messaging Mode)

The routing mode of the XBee 3 DigiMesh RF Module.

A routing device forwards broadcasts and route discoveries for unicasts. A non-routing device does neither.

Indirect Messaging Coordinator

Device will not transmit point to multi-point unicasts until an end device requests them. Indirect messaging is only applicable for point-to-multipoint messages (TO (Transmit Options) = **0x40**).

Indirect Messaging Poller

Device will periodically poll a coordinator for messages.

Parameter range

0 - 6

Parameter	Description	Routes packets
0	Standard router	Yes
1	Indirect message coordinator	Yes
2	Non-routing device	No
3	Non-routing coordinator	No
4	Indirect message poller	Yes
5	N/A	N/A
6	Non-routing poller	No

Parameter	Description	Routes packets
0	Standard router	Yes
1	Indirect message coordinator	Yes
2	Non-routing device	No
3	Non-routing coordinator	No
4	Indirect message poller	Yes
5	N/A	N/A
6	Non-routing poller	No

Default

0

C8 (Compatibility Options)

Sets or displays the operational compatibility with a legacy DigiMesh 2.4 device (S1 or S2C hardware). This parameter should only be set when operating in a mixed network that contains XBee Series 1 or XBee S2C devices.

Parameter range

0, 4

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Meaning	Setting	Description
2	TX compatibility	0	When encryption is enabled, AES Counter mode is used with a 256-bit key.
		1	When encryption is enabled AES ECB mode is used with a 128-bit key. This is compatible with legacy versions of DigiMesh 2.4.

Bit	Meaning	Setting	Description
3	Use XBee S1 compatible synchronous sleep messages	0	Use native XBee 3 synchronous sleep messages. This mode involves the least processing to keep the nodes synchronized.
		1	Convert synchronous sleep messages to be compatible with XBee S1. This mode must be used on all the nodes in the network if there are any XBee S1 nodes in the network. This mode may always be used, even if there are no S1 nodes in the network. But doing so reduces performance and accuracy and is not recommended unless XBee S1 nodes exist in the same network.

Default

0x00

Discovery commands

Network Discovery and corresponding discovery options.

NI (Network Identifier)

The node identifier is a user-defined name or description of the device. Use this string with network discovery commands in order to easily identify devices on the network.

Use the ND (Network Discover) command with this string as an argument to filter network discovery results.

Use the DN (Discover Node) command with this string as an argument to resolve the 64-bit address of a node with a matching **NI** string.

Parameter range

A string of case-sensitive ASCII printable characters from 1 to 20 bytes in length. A carriage return or a comma automatically ends the command.

Default

0x20 (an ASCII space character)

NT (Network Discovery Back-off)

Sets the amount of time a base node waits for responses from other nodes when using the ND (Network Discover), DN (Discover Node), and FN (Find Neighbors) commands.

When a discovery is performed, the broadcast transmission includes the **NT** value to provide all remote devices with a response timeout. Remote devices wait a random time, less than **NT**, before sending their response to avoid collisions.

The **N?** command should be used to determine how long the actual discovery timeout will be based on current device configuration.

Parameter range

0x20 - 0x2EE0 (x 100 ms)

Default

0x82 (13 seconds)

N? (Network Discovery Timeout)

The maximum response time, in milliseconds, for **ND** (Network Discovery) **DN** (Discover Node) and **FN** (Find Neighbor) responses. The timeout is the sum of **NT** (Network Discovery Back-off Time) and the network propagation time.

Parameter range

0x20 - 0xFFFF (x 100 ms) [read-only]

Default

N/A

NO (Network Discovery Options)

Set the Advanced Options that affect how a particular device responds to network discoveries—**ND**, **DN** and **FN** commands—and when sending a node identification.

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

0x0 - 0x7 (bit field)

Bit	Meaning
0x01	Append the **DD (Digi Device Identifier)** value to discovery responses and node identification frames.
0x02	Local device sends its own **ND** response when **ND** is issued.
0x04	Append the RSSI of the last hop to discovery responses and node identification frames.

Parameter range

0 - 3

Default

0x0

ND (Network Discover)

Discovers and reports all of the devices it finds on a network. If you send **ND** through a local or remote API frame, each network node returns a separate AT Command Response (0x88) or Remote Command Response (0x97) frame, respectively.

The command reports the following information after a jittered time delay.

SH<CR> (4 bytes)

SL<CR> (4 bytes)

DB<CR> (Contains the detected signal strength of the response in negative dBm units)

NI <CR> (variable, 0-20 bytes plus 0x00 character)

DEVICE_TYPE<CR> (1 byte: **0** = Coordinator, **1** = Router, **2** = End Device)

STATUS<CR> (1 byte: reserved)

PROFILE_ID<CR> (2 bytes)

MANUFACTURER_ID<CR> (2 bytes)

DIGI DEVICE TYPE<CR> (4 bytes. Optionally included based on **NO** settings.)

RSSI OF LAST HOP<CR> (1 byte. Optionally included based on **NO** settings.)

If you send the **ND** command in Command mode, after (**NT***100) ms + overhead time, the command ends by returning a carriage return, represented by <CR>.

The **ND** command accepts an **NI** (Node Identifier) as an argument. For more details, see Directed node discovery.

If the command includes an optional node identifier string parameter, only those devices with a matching **NI** string respond without a random offset delay. If the command does not include a node identifier string parameter, all devices respond with a random offset delay.

The **NT** setting determines the range of the random offset delay.

For more information about options that affect the behavior of the **ND** command Refer to NO (Network Discovery Options) for options which affect the behavior of the **ND** command.

The **ND** command cannot be issued from within MicroPython or over BLE.

WARNING! If the **NT** setting is small relative to the number of devices on the network, responses may be lost due to channel congestion. Regardless of the **NT** setting, because the random offset only mitigates transmission collisions, getting responses from all devices in the network is not guaranteed.

Parameter range

20-byte printable ASCII string (optional)

Default

N/A

DN (Discover Node)

Resolves an **NI** (Node identifier) string to a physical address (case sensitive).

The **DN** command cannot be issued from within MicroPython or over BLE.

The following events occur after **DN** discovers the destination node:

When **DN** is sent in Command mode:

1. The requesting node sets **DL** and **DH** to the address of the device with the matching **NI** string.
2. The requesting node returns **OK** (or ERROR).
3. If the requesting node returns **OK** (node found), it exits Command mode immediately with **DH**/**DL** set to the node that is found so that the next serial input is sent to the node designated by the **DN** parameter.
4. If the requesting node returns **ERROR**, (node not found), it remains in Command mode, allowing you to enter further commands.

When **DN** is sent as a local Local AT Command Request - 0x08:

1. The requesting node returns 0xFFFE followed by its 64-bit extended addresses in an Local AT Command Response - 0x88.
2. If there is no response from a module within (**N?*** 100) milliseconds or you do not specify a parameter (by leaving it blank), the requesting node returns an ERROR message.

Parameter range

20-byte ASCII string

Default

N/A

FN (Find Neighbors)

Discovers and reports all devices found within immediate (1 hop) RF range. **FN** reports the following information for each device it discovers:

MY<CR> (always 0xFFFE)

SH<CR>

SL<CR>

NI<CR> (Variable length)

PARENT_NETWORK ADDRESS<CR> (2 Bytes) (always 0xFFFE)

DEVICE_TYPE<CR> (1 Byte: 0 = Coordinator, 1 = Router, 2 = End Device)

STATUS<CR> (1 Byte: Reserved)

PROFILE_ID<CR> (2 Bytes)

MANUFACTURER_ID<CR> (2 Bytes)

DIGI DEVICE TYPE<CR> (4 Bytes. Optionally included based on NO (Network Discovery Options) settings.)

RSSI OF LAST HOP<CR> (1 Byte. Optionally included based on NO (Network Discovery Options) settings.)

<CR>

If you send the **FN** command in Command mode, after (**NT***100) ms + overhead time, the command ends by returning a carriage return, represented by <CR>.

If you send the **FN** command through a local AT Command (0x08) or remote AT command (0x17) API frame, each response returns as a separate AT Command Response (0x88) or Remote Command Response (0x97) frame, respectively. The data consists of the bytes in the previous list without the carriage return delimiters. The **NI** string ends in a 0x00 null character.

The **FN** command cannot be issued from within MicroPython or over BLE.

FN accepts a NI (Network Identifier) as an argument.

See Find specific neighbor for more details.

Parameter range

0 to 20 ASCII characters (optional)

Default

N/A

DigiMesh Addressing commands

The following commands affect how outgoing DigiMesh transmissions are addressed and configured.

SH (Serial Number High)

Displays the upper 32 bits of the unique IEEE 64-bit address assigned to the XBee in the factory.

This value is read-only and it never changes.

Parameter range

0x0013A200 - 0x0013A2FF [read-only]

Default

Set in the factory

SL (Serial Number Low)

Displays the lower 32 bits of the unique IEEE 64-bit address assigned to the XBee in the factory.

This value is read-only and it never changes.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

Set in the factory

DH (Destination Address High)

Set or read the upper 32 bits of the 64-bit destination address.

When you combine **DH** with **DL**, it defines the 64-bit destination address that the device uses for outgoing data transmissions in Transparent mode (**AP** = **0**) and I/O sampling. This destination address corresponds to the serial number (**SH** + **SL**) of the target device.

Reserved DigiMesh network addresses:

- 0x000000000000FFFF is a broadcast address (**DH** = **0**, **DL** = **0xFFFF**).

Parameter range

0 - 0xFFFFFFFF

Default

0

DL (Destination Address Low)

Set or read the lower 32 bits of the 64-bit destination address.

When you combine **DH** with **DL**, it defines the 64-bit destination address that the device uses for outgoing data transmissions in Transparent mode (**AP** = **0**) and I/O sampling. This destination address corresponds to the serial number (**SH** + **SL**) of the target device.

Reserved DigiMesh network addresses:

- 0x000000000000FFFF is a broadcast address (**DH** = **0**, **DL** = **0xFFFF**).

Parameter range

0 - 0xFFFFFFFF

Default

0xFFFF (broadcast)

RR (Unicast Mac Retries)

Set or read the maximum number of MAC level packet delivery attempts for unicasts. If **RR** is non-zero, the sent unicast packets request an acknowledgment from the recipient. Unicast packets can be retransmitted up to **RR** times if the transmitting device does not receive a successful acknowledgment.

Parameter range

0 - 0xF

Default

0xA (10 retries)

MT (Broadcast Multi-Transmits)

Set or read the number of additional MAC-level broadcast transmissions. All broadcast packets are transmitted **MT**+1 times to increase chances that they are received.

Parameter range

0 - 0xF

Default

3

TO (Transmit Options)

A bitfield that configures the advanced options used for outgoing data transmissions from a device operating in Transparent mode (**AP** = **0**).

When operating in API mode, if the Transmit Options field in the API frame is 0, the **TO** parameter value will be used instead.

Parameter range

0 - 0xFF

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Meaning	Description
0	Disable ACK	Disable MAC acknowledgments (retries) for unicast traffic.
1	Disable RD	Disable Route Discovery on all DigiMesh unicasts.
2	NACK	Enable a NACK messages on all DigiMesh API packets.
3	Trace Route	Enable a Trace Route on all DigiMesh API packets.
4	Secure Session	Send data securely—requires secure session be established with destination. Enabling this bit will reduce maximum payload size by 4 bytes.
6,7	Delivery method	b'00 = <invalid option> b'01 = Point-multipoint (0x40) b'10 = Directed Broadcast (0x80) b'11 = DigiMesh (0xC0)

Default

0

NP (Maximum Packet Payload Bytes)

Reads the maximum number of RF payload bytes that you can typically send in a transmission based on current parameter settings. Some options may impact maximum payload size that are not captured by the **NP** value.

See Maximum payload for more information.

The XBee 3 DigiMesh RF Module firmware returns a fixed number of bytes: 0x49 = 73 bytes without encryption, 65 bytes with encryption.

Note **NP** returns a hexadecimal value. For example, if **NP** returns 0x41, this is equivalent to 65 bytes.

Parameter range

0 - 0xFF [read-only]

Default

N/A

DigiMesh Configuration commands

The following AT commands adjust the advanced communication settings that affect outgoing data transmissions in a DigiMesh network.

AG (Aggregator Support)

The **AG** command sends a broadcast through the network that has the following effects on nodes that receive the broadcast:

- The receiving node establishes a DigiMesh route back to the originating node, if there is space in the routing table.
- The **DH** and **DL** of the receiving node update to the address of the originating node if the **AG** parameter matches the current **DH**/**DL** of the receiving node.
- API-enabled devices with updated **DH** and **DL** send an Aggregate Addressing Update frame (0x8E) out the serial port.

Parameter range

Any 64-bit address

Default

N/A

NH (Network Hops)

Sets or displays the maximum number of hops across the network. This parameter limits the number of hops for both unicasts and broadcasts. For example a RREQ is discarded after **NH** hops occur, preventing the route to a node more than **NH** hops away from being created. Without a route, unicasts will not work to that node. You can use this parameter to calculate the maximum network traversal time.

You must set this parameter to the same value on all nodes in the network.

If BH (Broadcast Hops) = **0**, **NH** is used to set the maximum number of hops across the network when sending a broadcast transmission. **NH** is also used to set the maximum number of hops for broadcast if **BH** > **NH**.

Parameter range

1 - 0x20 (1 - 32 hops)

Default

7

BH (Broadcast Hops)

The maximum transmission hops for broadcast data transmissions.

If you set **BH** greater than NH (Network Hops), the device uses the value of **NH**.

If you set **BH** to **0**, the device uses **NH** as a limit to the maximum number of hops.

When working in API mode, the **Broadcast Radius** field in the API frame is used instead of this configuration.

Parameter range

0 - 0x20

Default

0

MR (Mesh Unicast Retries)

Set or read the maximum number of network packet delivery attempts. If **MR** is non-zero, the packets a device sends request a network acknowledgment, and can be resent up to **MR**+1 times if the device does not receive an acknowledgment.

Changing this value dramatically changes how long a route request takes.

We recommend that you set this value to **1**.

If you set this parameter to **0**, it disables network ACKs. Initially, the device can find routes, but a route will never be repaired if it fails.

Parameter range

0 - 7 mesh unicast retries

Default

1

NN (Network Delay Slots)

Set or read the maximum random number of network delay slots before rebroadcasting a network packet.

One network delay slot is approximately 13 ms.

Parameter range

1 - 0xA network delay slots

Default

3

SE (Source Endpoint)

Sets or displays the application layer source endpoint value used for data transmissions.

This command only affects outgoing transmissions in Transparent mode (**AP** = **0**).

Note Endpoints **0xDC** - **0xEE** are reserved for special use by Digi and should not be used in an application outside of the listed purpose.

The reserved Digi endpoints are:

- 0xE8 - Digi data endpoint
- 0xE6 - Digi device object endpoint
- 0xE5 - Secure Session Server endpoint
- 0xE4 - Secure Session Client endpoint
- 0xE3 - Secure Session SRP authentication endpoint

Parameter range

0 - 0xFF

Default

0xE8

DE (Destination Endpoint)

Sets or displays the application layer destination endpoint used for data transmissions.

This command only affects outgoing transmissions in Transparent mode (**AP** = **0**).

Note Endpoints **0xDC** - **0xEE** are reserved for special use and should not be used in an application outside of the listed purpose.

The reserved Digi endpoints are:

- 0xE8 - Digi data endpoint
- 0xE6 - Digi device object endpoint
- 0xE5 - Secure Session Server endpoint
- 0xE4 - Secure Session Client endpoint
- 0xE3 - Secure Session SRP authentication endpoint

Parameter range

0 - 0xFF

Default

0xE8

CI (Cluster ID)

The application layer cluster ID value. The device uses this value as the cluster ID for all data transmissions in Transparent mode and for all transmissions performed with the Transmit Request - 0x10 in API mode. In API mode, transmissions performed with the Explicit Addressing Command Request - 0x11 ignore this parameter.

If you set this value to **0x12** (loopback Cluster ID), the destination node echoes any transmitted packet back to the source device.

Parameter range

0 - 0xFFFF

Default

0x11 (Transparent data cluster ID)

Diagnostic commands - addressing timeouts

The following AT commands provide the transmission and discovery timeout values.

%H (MAC Unicast One Hop Time)

The MAC unicast one hop time timeout in milliseconds. If you change the MAC parameters it can change this value.

The time to send a unicast between two nodes in the network should not exceed the product of the unicast one hop time (**%H**) and the number of hops between those two nodes.

Parameter range

[read-only]

Default

N/A

%8 (MAC Broadcast One Hop Time)

The MAC broadcast one hop time timeout in milliseconds. If you change MAC parameters, it can change this value.

The time to send a broadcast between two nodes in the network should not exceed the product of the broadcast one hop time (**%8**) and the number of hops between those two nodes.

Parameter range

[read-only]

Default

N/A

Security commands

The following commands enable and control the encryption used for RF transmissions.

EE (Encryption Enable)

Enables or disables Advanced Encryption Standard (AES) encryption. See bit 2 of C8 (Compatibility Options), which controls the encryption mode.

Set this command parameter the same on all devices in a network.

Parameter range

0 - 1

Parameter	Description
0	Encryption Disabled
1	Encryption Enabled

Default

0

KY (AES Encryption Key)

The Link Key used for encryption and decryption. If C8 (Compatibility Options) bit 2 is cleared, encryption/decryption uses the 256 bits of the **KY** value (all 64 ASCII characters of the **KY** value). **C8** bit 2 sets encryption/decryption, and uses the last 32 ASCII characters of the 256-bit **KY** value entered.

This command is write-only and cannot be read. If you attempt to read **KY**, the device returns an **OK** status.

Set this command parameter the same on all devices in a network.

Parameter range

256-bit value (up to 32 hex bytes/64 ASCII bytes)

Default

0

DM (Disable Features)

A bit field mask that you can use to enable or disable DigiMesh features. If disabling device functionality for security purposes, we recommend that you also enable secure remote configuration to prevent features from being re-enabled remotely.

We highly recommend that you set the same **DM** value for bits 0 and 1 on every node on the network, otherwise you may encounter unexpected behavior when attempting to use the DigiMesh diagnostic features.

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Description
0	Disable aggregator updates. When set to 1, the device does not issue or respond to **AG** requests.
1	Disable Trace Route and NACK responses. When set to 1, the device does not generate or respond to Trace Route or NACK requests. If this bit is to be set, we highly recommend that you set the same value on every node on the network, otherwise you may encounter unexpected behavior when attempting to use the DigiMesh diagnostic features.
2	Disable firmware over-the-air (FOTA) updates. When set to 1, the device cannot act as a FOTA client. FOTA File System access is protected with FK (File System Public Key). **Note** Serial firmware updates are always possible via the bootloader.
3	Disable SRP authentication on the client side of the connection.
4	Disable SRP authentication on the server side of the connection.

Parameter range

0 - 0xFF

Default

0

US (OTA Upgrade Server)

Specifies the 64-bit address of the server the device should use for OTA upgrades.

- 0: Accept OTA upgrades from any device
- 0x1-0xFFFFFFFFFFFFFFFE: Only accept OTA upgrades from a server with the given 64-bit address
- 0xFFFFFFFFFFFFFFFF: Reserved

Parameter range

0 - 0xFFFFFFFFFFFFFFFE

Default

0

Secure Session commands

These are the AT commands that enable Secure Session.

SA (Secure Access)

The Secure Access Options bit-field defines the feature set(s) intended to be secure against unauthorized access. The XBee 3 DigiMesh RF Module should establish a secure session in order to access functionality defined by the feature set(s) on the local device.

A password must be set using the Secure Session Salt and Verifier before access is secured.

Parameter range

0 - 0x1F (up to 0xFFFF)

Bit field

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Description
0	Reserved
1	Remote AT Commands When set to **1** and if a password has been set, the device will not respond to insecure Remote AT Command requests (API Frame 0x17) but still can send insecure Remote AT Commands.
2	Serial Data When set to **1**, the device will not emit any serial data that was sent insecurely. This functionality applies to devices that are configured for Transparent mode, but in this instance, only the SRP server would be **AP** = **0**, the client would still have to send the Secure Session Control - 0x2E via API mode. The server will also not emit any 0x90 or 0x91 frames when this bit is set.

Default

0

*S (Secure Session Salt)

The Secure Remote Password (SRP) Salt is a 32-bit number used to create an encrypted password for the XBee 3 DigiMesh RF Module. The ***S** command contains the salt value in the salt/verifier pair used for secure session authentication.

Parameter range

0-FFFFFFFF

Default

0

*V, *W, *X, *Y (Secure Session Verifier)

The secure session verifier is a 128-byte value used together with *S (Secure Session Salt) for secure session authentication. The ***V**, ***W**, ***X**, and ***Y** commands each contain 32 bytes of the secure session verifier: ***V** contains bytes 0 - 31, ***W** bytes 32 - 63, ***X** bytes 54 - 95, and ***Y** bytes 96 - 127.

Parameter range

Each command can be any 32-byte value: 0-FFFFFFFF

Default

0

RF interfacing commands

The following AT commands affect the 2.4 GHz DigiMesh RF interface of the device.

PL (TX Power Level)

Sets or displays the power level at which the device transmits conducted power for DigiMesh traffic.

Note If operating on channel 26 (**CH** = **0x1A**), output power will be capped and cannot exceed 8 dBm regardless of the **PL** setting.

Parameter range

0 - 4

Parameter	XBee non-PRO	XBee 3 PRO
0	-5 dBm	-5 dBm
1	-1 dBm	+3 dBm
2	+2 dBm	+8 dBm
3	+5 dBm	+15 dBm
4	+8 dBm	+19 dBm

Default

4

PP (Output Power in dBm)

Display the operating output power based on the current configuration (channel and **PL** setting). The values returned are in dBm, with negative values represented in two's complement; for example:

-5 dBm = 0xFB.

Parameter range

0 - 0xFF [read-only]

Default

N/A

CA (CCA Threshold)

Defines the Clear Channel Assessment (CCA) threshold. Prior to transmitting a packet, the device performs a CCA to detect energy on the channel. If the device detects energy above the CCA threshold, it will not transmit the packet.

The **CA** parameter is measured in units of -dBm.

Parameter range

0 (disabled), 0x28 - 0x64 (-dBm)

Default

0x0 (CCA disabled)

MAC diagnostics commands

The following commands provide Media Access Control diagnostic information.

DB (Last Packet RSSI)

Reports the RSSI in -dBm of the last received RF data packet. **DB** returns a hexadecimal value for the -dBm measurement.

For example, if **DB** returns 0x60, then the RSSI of the last packet received was -96 dBm.

DB only indicates the signal strength of the last hop. It does not provide an accurate quality measurement for a multihop link.

If the XBee 3 DigiMesh RF Module has been reset and has not yet received a packet, **DB** reports **0**.

This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFF [read-only]

Default

0

EA (MAC ACK Failure Count)

The number of unicast transmissions that time out awaiting a MAC ACK. This can be up to **RR** +1 timeouts per unicast when **RR** > 0.

This count increments whenever a MAC ACK timeout occurs on a MAC-level unicast. When the number reaches **0xFFFF**, the firmware does not count further events.

To reset the counter to any 16-bit unsigned value, append a hexadecimal parameter to the command.

This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFFFF

Default

0x0

EC (CCA Failures)

Sets or displays the number of frames that were blocked and not sent due to CCA failures or receptions in progress. If CCA is disabled (**CA** is **0**), then this count only increments for frames that are blocked due to receive in progress. When this count reaches its maximum value of **0xFFFF**, it stops counting.

You can reset **EC** to **0** (or any other value) at any time to make it easier to track errors. This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFFFF

Default

0x0

BC (Bytes Transmitted)

The number of RF bytes transmitted. The firmware counts every byte of every packet, including MAC/PHY headers and trailers.

You can reset the counter to any 32-bit value by appending a hexadecimal parameter to the command. This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFFFFFFFF

Default

N/A (0 after reset)

GD (Good Packets Received)

This count increments when a device receives a good frame with a valid MAC header on the RF interface. Received MAC ACK packets do not increment this counter. Once the number reaches 0xFFFF, it does not count further events.

To reset the counter to any 16-bit unsigned value, append a hexadecimal parameter to the command.

This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFFFF

Default

N/A (0 after reset)

TR (Transmission Failure Count)

This count increments whenever a MAC transmission attempt exhausts all MAC retries without ever receiving a MAC acknowledgment message from the destination node. Once the number reaches 0xFFFF, it does not count further events.

To reset the counter to any 16-bit value, append a hexadecimal parameter to the command.

This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFFFF

Default

N/A (0 after reset)

UA (Unicasts Attempted Count)

The number of unicast transmissions expecting an acknowledgment (when **RR** > 0).

To reset the counter to any 16-bit value, append a hexadecimal parameter to the command.

UA is a volatile value—that is, the value does not persist across device resets.

Parameter range

0 - 0xFFFF

Default

0

ED (Energy Detect)

Starts an energy detect scan. The device loops through all the available channels and returns the maximal energy on each channel, a comma follows each value, and the list ends with a carriage return. The values returned reflect the energy level that **ED** detects in -dBm units.

ED accepts a parameter value but it will not affect the scan duration or results.

ED cannot be issued from within MicroPython or over BLE.

Parameter range

0 - 0xFF

Default

N/A

Sleep settings commands

The following commands enable and configure the low power sleep modes of the device.

SM (Sleep Mode)

Sets or displays the sleep mode of the device.

Normal mode is always awake. Pin sleep modes allow you to wake the device with the SLEEP_REQUEST line. Asynchronous cyclic mode sleeps for **SP** time and briefly wakes, checking for activity. Sleep support mode is always awake, but can effectively communicate with synchronized cyclic sleep nodes. Synchronized Cyclic Sleep nodes keep the same wake and sleep cycles for all nodes in the network.

Parameter range

0 - 5

0, 1, 4 - 8

Parameter	Description
0	No sleep (always awake)
1	Pin sleep
2	Unused
3	Unused
4	Asynchronous cyclic sleep
5	Asynchronous cyclic sleep with pin wakeup
6	MicroPython sleep (with optional pin wake). For complete details see the *Digi MicroPython Programming Guide*.
7	Synchronous sleep support mode
8	Synchronous cyclic sleep

Default

0

SP (Cyclic Sleep Period)

Sets or displays the device's sleep time. This command defines the amount of time the device sleeps per cycle.

For a node operating as an Indirect Messaging Coordinator, this command defines the amount of time that it will hold an indirect message for an end device. The coordinator will hold the message for (2.5 * **SP**).

Parameter range

0x1 - 0x15F900 (x 10 ms) (4 hours)

Default

0xC8

ST (Cyclic Sleep Wake Time)

Sets or displays the wake time of the device.

For devices in asynchronous cyclic sleep, **ST** defines the amount of time that a device stays awake after it receives RF or serial data.

For devices in synchronous sleep, the minimum wake time is a function of **MT**, **RN**, **NH**, and **NN**. If you increase these values such that **ST** is no longer big enough to get a message through the network during a wake cycle, then **ST** will be increased appropriately.

Parameter range

0x1 - 0x36EE80 (x 1 ms)

Default

0x7D0 (2 seconds)

SN (Number of Sleep Periods)

Set or read the number of sleep periods value. This command controls the number of sleep periods that must elapse between assertions of the ON_SLEEP line during the wake time of Asynchronous or Synchronous Cyclic Sleep. This allows external circuitry to sleep longer than the **SP** time.

Parameter range

1 - 0xFFFF

Default

1

WH (Wake Host Delay)

Sets or displays the wake host timer value. You can use **WH** to give a sleeping host processor sufficient time to power up after the device asserts the ON_SLEEP line.

If you set **WH** to a non-zero value, this timer specifies a time in milliseconds that the device delays after waking from sleep before sending data out the UART or transmitting an I/O sample. If the device receives serial characters, the **WH** timer stops immediately.

Parameter range

0 - 0xFFFF (x 1 ms)

Default

0

SO (Sleep Options)

A bitfield that contains advanced sleep options that do not have dedicated AT commands.

You cannot set bit 0 and bit 1 at the same time.

Parameter range

0 - 0x13E

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Description
0	Sleep coordinator; setting this bit causes a sleep compatible device to always act as sleep coordinator.
1	Non-sleep coordinator; setting this bit causes a device to never act as a sleep coordinator.
2	Enable API sleep status messages.
3	Disable early wake-up for missed syncs.
4	Enable node type equality (disables seniority based on device type).
5	Disable coordinator rapid sync deployment mode.

Default

0

Diagnostic commands - sync sleep status/timing

The following AT commands provide timing and status information for a synchronized cyclically sleeping network (**SM** = **7** or **8**).

SS (Sleep Status)

Queries a number of Boolean values that describe the device's status.

Bit	Description
0	This bit is true when the network is awake and able to receive transmissions.
1	This bit is true if the node currently acts as a network sleep coordinator.
2	This bit is true if the node ever receives a valid sync message after it powers on.
3	This bit is true if the node receives a sync message in the current wake cycle.
4	This bit is true if you alter the sleep settings on the device so that the node nominates itself and sends a sync message with the new settings at the beginning of the next wake cycle.
5	This bit is true if you request that the node nominate itself as the sleep coordinator using the Commissioning Pushbutton or the **CB**2 command.
6	This bit is true if the node is currently in deployment mode.
All other bits	Reserved. Ignore all non-documented bits.

Parameter range

N/A

Default

N/A

OS (Operating Sleep Time)

Reads the current network sleep time that the device is synchronized to, in units of 10 milliseconds. If the device has not been synchronized, then **OS** returns the value of **SP**.

If the device synchronizes with a sleeping router network, **OS** may differ from **SP**.

Parameter range

N/A

Default

N/A

OW (Operating Wake Time)

Reads the current network wake time that a device is synchronized to, in 1 ms units.

If the device has not been synchronized, then **OW** returns the value of **ST**.

If the device synchronizes with a sleeping router network, **OW** may differ from **ST**.

Parameter range

N/A

Default

N/A

MS (Missed Sync Messages)

Reads the number of sleep or wake cycles since the device received a sync message.

Parameter range

N/A

Default

N/A

SQ (Missed Sleep Sync Count)

Counts the number of sleep cycles in which the device does not receive a sleep sync.

Set the value to 0 to reset this value.

When the value reaches 0xFFFF it does not increment anymore.

Parameter range

0 - 0xFFFF

Default

N/A

MicroPython commands

The following commands relate to using MicroPython on the XBee 3 DigiMesh RF Module.

PS (Python Startup)

Sets whether or not the XBee 3 DigiMesh RF Module runs the stored Python code at startup.

Range

0 - 1

Parameter	Description
0	Do not run stored Python code at startup.
1	Run stored Python code at startup.

Default

0

PY (MicroPython Command)

Interact with the XBee 3 DigiMesh RF Module using MicroPython. **PY** is a command with sub-commands. These sub-commands are arguments to **PY**.

PYB (Bundled Code Report)

You can store compiled code in flash using the **os.bundle()** function in the MicroPython REPL; refer to the *Digi MicroPython Programming Guide*. The **PYB** sub-command reports details of the bundled code. In Command mode, it returns two lines of text, for example:

```
bytecode: 619 bytes (hash=0x0900DBCE)
compiled: 2017-05-09T15:49:44
```

The messages are:

- **bytecode**: the size of bytecode stored in flash and its 32-bit hash. A size of **0** indicates that there is no stored code.
- **compiled**: a compilation timestamp. A timestamp of **2000-01-01T00:00:00** indicates that the clock was not set during compilation.

In API mode, **PYB** returns three 32-bit big-endian values:

- bytecode size
- bytecode hash
- timestamp as seconds since 2000-01-01T00:00:00

PYE (Erase Bundled Code)

PYE interrupts any running code, erases any bundled code and then does a soft-reboot on the MicroPython subsystem.

PYV (Version Report)

Report the MicroPython version.

PY^ (Interrupt Program)

Sends **KeyboardInterrupt** to MicroPython. This is useful if there is a runaway MicroPython program and you have filled the stdin buffer. You can enter Command mode (**+++**) and send **ATPY^** to interrupt the program.

Default

N/A

File System commands

To access the file system, enter Command mode and use the following commands. All commands block the AT command processor until completed and only work from Command mode; they are not valid for API mode or MicroPython's **xbee.atcmd()** method. Commands are case-insensitive as are file and directory names. Optional parameters are shown in square brackets (**[]**).

FS (File System)

FS is a command with sub-commands. These sub-commands are arguments to **FS**.

Error responses

If a command succeeds it returns information such as the name of the current working directory or a list of files, or **OK** if there is no information to report. If it fails, you see a detailed error message instead of the typical **ERROR** response for a failing AT command. The response is a named error code and a textual description of the error.

Note The exact content of error messages may change in the future. All errors start with a upper case **E**, followed by one or more uppercase letters and digits, a space, and an description of the error. If writing your own AT command parsing code, you can determine if an **FS** command response is an error by checking if the first letter of the response is upper case **E**.

FS (File System)

When sent without any parameters, **FS** prints a list of supported commands.

FS PWD

Prints the current working directory, which always starts with **/** and defaults to **/flash** at startup.

FS CD *directory*

Changes the current working directory to **directory**. Prints the current working directory or an error if unable to change to **directory**.

FS MD *directory*

Creates the directory **directory**. Prints **OK** if successful or an error if unable to create the requested directory.

FS LS [*directory*]

Lists files and directories in the specified directory. The **directory** parameter is optional and defaults to a period (**.**), which represents the current directory. The list ends with a blank line.

Entries start with zero or more spaces, followed by file size or the string **<DIR>** for directories, then a single space character and the name of the entry. Directory names end with a forward slash (**/**) to differentiate them from files.

```
<DIR> ./
<DIR> ../
<DIR> lib/
    32 test.txt
```

FS PUT filename

Starts a YMODEM receive on the XBee 3 DigiMesh RF Module, storing the received file to **filename** and ignoring the filename that appears in block 0 of the YMODEM transfer. The XBee 3 DigiMesh RF Module sends a prompt (**Receiving file with YMODEM...**) when it is ready to receive, at which point you should initiate a YMODEM send in your terminal emulator.

If the command is incorrect, the reply will be an error as described in Error responses.

FS HASH *filename*

Print a SHA-256 hash of a file to allow for verification against a local copy of the file. On Windows, you can generate a SHA-256 hash of a file with the command **certutil -hashfile test.txt SHA256**. On Mac and Linux use **shasum -b -a 256 test.txt**.

FS GET *filename*

Starts a YMODEM send of filename on the XBee device. When it is ready to send, the XBee 3 DigiMesh RF Module sends a prompt: (**Sending file with YMODEM...**). When the prompt is sent, you should initiate a YMODEM receive in your terminal emulator.

If the command is incorrect, the reply will be an error as described in Error responses.

FS RM *file_or_directory*

Removes the file or empty directory specified by **file_or_directory**. This command fails with an error if **file_or_directory** does not exist, is not empty, refers to the current working directory or one of its parents.

Note Removing a file only reclaims space if the file removed is placed last in the file system. Deleted data that is contiguous with the last deleted file is also reclaimed. Directories are only reclaimed if all directories in that particular block of memory are deleted and found at the end of the file system. Use the **ATFS INFO FULL** command to see where in the file system files and directories are placed.

FS INFO

Report on the size of the filesystem, showing bytes in use, available, marked bad and total. The report ends with a blank line, as with most multi-line AT command output. Example output:

```
204800 used
 695296 free
     0 bad
 900096 total
```

FS INFO FULL

Reports every file and directory in the order they are placed in the file system along with the amount of space they take up individually. Also reports deleted space as well as unused directory slots. Example output:

```
128 /flash./
128 /flash/lib./
128 /flash/directory./
1664 [unused dir slot(s)]
2048 /flash/file1.txt.
2048 [deleted space]
2048 /flash/directory/file2.txt
```

FS FORMAT confirm

Formats the file system, leaving it with a default directory structure. Pass the word **confirm** as the first parameter to confirm the format. The XBee 3 DigiMesh RF Module responds with **Formatting...** when the format starts, and will print **OK** followed by a carriage return when it finishes.

FK (File System Public Key)

Configures the device's File System Public Key.

You must set **FK** locally via Command Mode or 0x08 or 0x09 API frames. You cannot set the public key remotely.

The 65-byte public key is required to verify that the file system that is downloaded over-the-air is a valid XBee 3 file system compatible with the DigiMesh firmware.

For further information, refer to Set the public key on the XBee device.

Parameter range

A valid 65-byte ECDSA public key—all 65-bytes must be entered, including any leading zeros.

Other accepted parameters:

0 = Clear the public key

1 = Returns the upper 48 bytes of the public key

2 = Returns the lower 17 bytes of the public key

Default

0

Note The Default value of **0** indicates that no public key has been set and hence, all file system updates will be rejected.

Bluetooth Low Energy (BLE) commands

The following AT commands are BLE commands.

BT (Bluetooth Enable)

BT enables or disables the Bluetooth functionality.

Note When Bluetooth is enabled, the XBee 3 DigiMesh RF Module cannot be in Sleep mode. If the device is configured to allow Sleep mode and you enable Bluetooth, the XBee 3 DigiMesh RF Module will not enter sleep.

WARNING! RF data loss may be encountered when BLE is enabled due to the PHY switching between RF and BLE. We highly recommended that you enable retries and multi-transmit—via the **RR** and **MT** commands—when BLE is enabled.

Parameter range

Parameter	Description
0	Bluetooth functionality is disabled.
1	Bluetooth functionality is enabled.

Default

0

BL (Bluetooth Address)

BL reports the EUI-48 Bluetooth device address. Due to standard XBee AT Command processing, leading zeroes are not included in the response when in Command mode.

Parameter range

N/A

Default

N/A

BI (Bluetooth Identifier)

A human-friendly name for the device. This is the name that will appear in bluetooth advertisement messages.

If set to default (ASCII space character), the bluetooth indicator will display as **XBee3 DigiMesh**.

If using XBee Mobile, adjustments to the filter options will be needed if this value is populated.

Parameter range

A string of case-sensitive ASCII printable characters from 1 to 22 bytes in length.

Default

0x20 (an ASCII space character)

BP (Bluetooth Power)

Sets the power level for Bluetooth Advertisements. All other BLE transmissions are sent at 8 dBm.

Parameter range

Parameter	Description
0	-20 dBm
1	-10 dBm
2	0 dBm
3	8 dBm

Default

3 = 8 dBm

$S (SRP Salt)

Note You should only use this command if you have already configured a password on the XBee device and the salt corresponds to the password.

The Secure Remote Password (SRP) Salt is a 32-bit number used to create an encrypted password for the XBee 3 DigiMesh RF Module. Use the **$S** command in conjunction with the **$V**, **$W**, **$X**, and **$Y** verifiers. Together, the command and the verifiers authenticate the client for the BLE API Service without storing the XBee password on the XBee 3 DigiMesh RF Module.

Configure the salt in the **$S** command. In the **$V**, **$W**, **$X**, and **$Y** verifiers, you specify the 128-byte verifier value, where each command represents 32 bytes of the total 128-byte verifier value.

Note The XBee 3 DigiMesh RF Module does not allow for **0** to be valid salt. If the value is **0**, SRP is disabled and you are not able to authenticate using Bluetooth.

Parameter range

0 - FFFFFFFF

Default

0

$V, $W, $X, $Y commands (SRP Salt verifier)

Use the **$V**, **$W**, **$X**, and **$Y** verifiers in conjunction with $S (SRP Salt) to create an encrypted password for the XBee 3 DigiMesh RF Module. Together, **$S** and the verifiers authenticate the client for the BLE API Service without storing the XBee password on the XBee device.

Configure the salt with the **$S** command. In the **$V**, **$W**, **$X**, and **$Y** verifiers, you specify the 128-byte verifier value, where each command represents 32 bytes of the total 128-byte verifier value.

Parameter range

0 - FFFFFFFF

Default

0

API configuration commmands

The following commands affect how API mode operates.

AP (API Enable)

Set or read the API mode setting. The device can format the RF packets it receives into API frames and sends them out the serial port.

When you enable API, you must format the serial data as API frames because Transparent operating mode is disabled.

Parameter range

0 - 4

Parameter	Description
0	API disabled (operate in Transparent mode)
1	API enabled
2	API enabled (with escaped control characters)
4	API enabled (operate in Micropython mode)

Default

0

AO (API Options)

Configure the serial output options for received API frames. This parameter is only applicable when the device is operating in API mode (**AP** = **1** or **2**).

Parameter range

0 - 1

Parameter	Description
0	API Rx Indicator - 0x90, this is for standard data frames.
1	API Explicit Rx Indicator - 0x91, this is for Explicit Addressing data frames.

Default

0

AZ (Extended API Options)

Optionally output additional ZCL messages that would normally be masked by the XBee application.

Use this when debugging FOTA updates by enabling client-side messages to be sent out of the serial port.

The bits in this parameter are used to enable different kinds of normally-suppressed output:

Parameter range

0x00 - 0x0A (bitfield)

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Description
0	Reserved
1	Output receive frames for FOTA update commands
2	Reserved
3	Output Extended Modem Status (0x98) frames instead of Modem Status (0x8A) frames when a Secure Session status change occurs

Default

0

UART interface commands

The following AT commands are serial interfacing commands.

BD (UART Baud Rate)

This command configures the serial interface baud rate for communication between the UART port of the device and the host.

The device interprets any value between 0x12C and 0x0EC400 as a custom baud rate. Custom baud rates are not guaranteed and the device attempts to find the closest achievable baud rate. After setting a non-standard baud rate, query **BD** to find the actual operating baud rate before applying changes.

Parameter range

Standard baud rates: 0x0 - 0x0A

Non-standard baud rates: 0x12C - 0x0EC400

Parameter	Description
0x0	1200 b/s
0x1	2400 b/s
0x2	4800 b/s
0x3	9600 b/s
0x4	19200 b/s
0x5	38400 b/s
0x6	57600 b/s
0x7	115200 b/s
0x8	230,400 b/s
0x9	460,800 b/s
0xA	921,600 b/s

Default

0x03 (9600 baud)

NB (Parity)

Set or read the serial parity settings for UART communications.

The device does not actually calculate and check the parity. It only interfaces with devices at the configured parity and stop bit settings for serial error detection.

Parameter range

0 - 2

Parameter	Description
0	No parity
1	Even parity
2	Odd parity

Default

0

SB (Stop Bits)

Sets or displays the number of stop bits for UART communications.

Parameter range

0 - 1

Parameter	Description
0	One stop bit
1	Two stop bits

Default

0

FT (Flow Control Threshold)

Set or display the flow control threshold.

The device de-asserts $\overline{CTS}$ when **FT** bytes are in the UART receive buffer. It re-asserts $\overline{CTS}$ when somewhat less than **FT** bytes are in the UART receive buffer. "Somewhat less than" allows for hysteresis so that $\overline{CTS}$ is not toggling rapidly when close to **FT** bytes are in the UART receive buffer.

Parameter range

0x14 - 0x110 bytes

Default

0xD9

RO (Packetization Timeout)

Set or read the number of character times of inter-character silence required before transmission begins when operating in Transparent mode. A “character time” is the amount of time it takes to send a single ASCII character at the operating baud rate (**BD**).

Set **RO** to 0 to transmit characters as they arrive instead of buffering them into one RF packet.

The **RO** command only applies to Transparent mode, it does not apply to API mode.

Parameter range

0 - 0xFF (x character times)

Default

3

AT Command options

The following commands affect how Command mode operates.

CC (Command Character)

Sets or displays the character value used to break from data mode to Command mode. The command character must be sent three times in succession while observing the minimum guard time (**GT**) of silence before and after this sequence.

The default value (**0x2B**) is the ASCII code for the plus (**+**) character. You must enter it three times within the guard time to enter Command mode. To enter Command mode, there is also a required period of silence before and after the command sequence characters of the Command mode sequence (**GT** + **CC** + **GT**). The period of silence prevents inadvertently entering Command mode. For more information, see Enter Command mode.

Parameter range

0 - 0xFF

Recommended: 0x20 - 0x7F (ASCII)

Default

0x2B (the ASCII plus character: **+**)

CT (Command Mode Timeout)

Sets or displays the Command mode timeout parameter. If the local device enters Command mode and does not receive any valid AT commands within this time period, Command mode silently exits.

Parameter range

2 - 0x1770 (x 100 ms)

Default

0x64 (10 seconds)

GT (Guard Time)

Set the required period of silence before and after the command sequence characters of the Command mode sequence, **GT** + **CC** + **GT**. The period of silence prevents inadvertently entering Command mode if a data stream in Transparent mode includes the **CC** character. For more information, see Enter Command mode.

Parameter range

0x2 - 0x6D3 (x 1 ms)

Default

0x3E8 (one second)

CN (Exit Command mode)

Executable command. **CN** immediately exits Command mode and applies pending changes.

Parameter range

N/A

Default

N/A

UART pin configuration commands

The following commands are related to pin configuration for the UART interface.

D6 (DIO6/RTS Configuration)

Sets or displays the DIO6/$\overline{\text{RTS}}$ configuration (Micro pin 27/SMT pin 29/TH pin 16).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	$\overline{\text{RTS}}$ flow control
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D7 (DIO7/CTS Configuration)

Sets or displays the DIO7/$\overline{\text{CTS}}$ configuration (Micro pin 24/SMT pin 25/TH pin 12).

Parameter range

0, 1, 3 - 7

Parameter	Description
0	Disabled
1	$\overline{\text{CTS}}$ flow control
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high
6	RS-485 enable, low
7	RS-485 enable, high

Default

1

P3 (DIO13/UART_DOUT)

Sets or displays the DIO13/UART_DOUT configuration (Micro pin 3/SMT pin 3/TH pin 2).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	UART DOUT
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P4 (DIO14/UART_DIN Configuration)

Sets or displays the DIO14/UART_DIN configuration (Micro pin 4/SMT pin 4/TH pin 3).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	UART DIN
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

SMT/MMT SPI interface commands

The following commands affect the SPI serial interface on SMT and MMT variants. These commands are not applicable to the through-hole variant of the XBee 3; see **D1** through **D4** and **P2** for through-hole SPI support.

P5 (DIO15/SPI_MISO Configuration)

Sets or displays the DIO15/SPI_MISO configuration (Micro pin 16/SMT pin 17). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MISO
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P6 (DIO16/SPI_MOSI Configuration)

Sets or displays the DIO16/SPI_MOSI configuration (Micro pin 15/SMT pin 16). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MOSI
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P7 (DIO17/SPI_SSEL Configuration)

Sets or displays the DIO17/SPI_$\overline{\text{SSEL}}$ configuration (Micro pin 14/SMT pin 15). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{SSEL}}$
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P8 (DIO18/SPI_CLK Configuration)

Sets or displays the DIO18/SPI_CLK configuration (Micro pin 13/SMT pin 14). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_CLK
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P9 (DIO19/SPI_$\overline{\text{ATTN}}$ Configuration)

Sets or displays the DIO19/SPI_$\overline{\text{ATTN}}$ configuration (Micro pin 11/SMT pin 12). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_ATTN
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

I/O settings commands

The following commands configure the various I/O lines available on the XBee 3 DigiMesh RF Module.

D0 (DIO0/ADC0/Commissioning Configuration)

Sets or displays the DIO0/ADC0/CB configuration (Micro pin 31/SMT pin 33/TH pin 20).

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	Commissioning Pushbutton
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

CB (Commissioning Button)

Use **CB** to simulate Commissioning Pushbutton presses in software.

You can enable a physical commissioning pushbutton with D0 (DIO0/ADC0/Commissioning Configuration).

Set the parameter value to the number of button presses that you want to simulate. For example, send **CB1** to perform the action of pressing the Commissioning Pushbutton once.

Parameter range

1, 2, 4

Parameter	Description
1	Keeps device awake for 30 seconds.
2	Nominate the node as the sleep coordinator for synchronous sleep networks.
4	Restore defaults (equivalent to sending an RE (Restore Defaults)).

Default

N/A

D1 (DIO1/ADC1/TH_SPI_$\overline{\text{ATTN}}$ Configuration)

Sets or displays the DIO1/ADC1/TH_SPI_$\overline{\text{ATTN}}$ configuration (Micro pin 30/SMT pin 32/TH pin 19).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{ATTN}}$ for the through-hole device N/A for the surface-mount device
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D2 (DIO2/ADC2/TH_SPI_CLK Configuration)

Sets or displays the DIO2/ADC2/TH_SPI_CLK configuration (Micro pin 29/SMT pin 31/TH pin 18).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_CLK for through-hole devices N/A for surface-mount devices
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)

Sets or displays the DIO3/ADC3/TH_SPI_SSEL configuration (Micro pin 28/SMT pin 30/TH pin 17).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_SSEL for the through-hole device N/A for surface-mount device
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Parameter	Description
0	Disabled
1	SPI_SSEL for the through-hole device N/A for surface-mount device
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D4 (DIO4/TH_SPI_MOSI Configuration)

Sets or displays the DIO4/TH_SPI_MOSI configuration (Micro pin 23/SMT pin 24/TH pin 11).

Parameter range

SMT/MMT: 0, 3 - 5

TH: 0, 1, 3 - 5

Parameter	Description
0	Disabled
1	SPI_MOSI for the through-hole device N/A for the surface-mount and micro device
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D5 (DIO5/Associate Configuration)

Sets or displays the DIO5/ASSOCIATED_INDICATOR configuration (Micro pin 26/SMT pin 28/TH pin 15).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Associate LED indicator - blinks when associated
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

D8 (DIO8/DTR/SLP_Request Configuration)

Sets or displays the DIO8/DTR/SLP_RQ configuration (Micro pin 9/SMT pin 10/TH pin 9).

Note If **D8** is configured as DTR/Sleep_Request (**1**), the line will be left floating while the device sleeps. Leaving **D8** set to **1** and the corresponding pin not connected to anything external to the device may result in higher sleep current draw.

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	DTR/Sleep_Request (used with pin sleep and cyclic sleep with pin wake)
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

D9 (DIO9/ON_SLEEP Configuration)

Sets or displays the DIO9/ON_SLEEP configuration (Micro pin 25/SMT pin 26/TH pin 13).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	ON/SLEEP indicator
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P0 (DIO10/RSSI/PWM0 Configuration)

Sets or displays the DIO10/RSSI/PWM0 configuration (Micro pin 7/SMT pin 7/TH pin 6).

When configured as RSSI PWM output, the device outputs a PWM signal with a duty cycle equivalent to the dBm of the received packet.

Use RP (RSSI PWM Timer) to configure the timeout.

When configured as PWM output (**2**): you can use **M0** to explicitly control the PWM0 output. When used with Analog line passing, PWM0 corresponds with ADC0.

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	RSSI PWM output
2	PWM0 output. M0 (PWM0 Duty Cycle) or I/O line passing control the value.
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P1 (DIO11/PWM1 Configuration)

Sets or displays the DIO11 configuration (Micro pin 8/SMT pin 8/TH pin 7).

Parameter range

0, 2 - 5

Parameter	Description
0	Disabled
1	N/A
2	PWM1 output. M1 (PWM1 Duty Cycle) or I/O line passing control the value.
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

P2 (DIO12/TH_SPI_MISO Configuration)

Sets or displays the DIO12/TH_SPI_MISO configuration (Micro pin 5/SMT pin 5/TH pin 4).

Parameter range

SMT/MMT: 0, 3 - 5

TH: 0, 1, 3 - 5

Parameter	Description
0	Disabled
1	SPI_MISO for the through-hole device N/A for the surface-mount and micro device
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

PR (Pull-up/Down Resistor Enable)

The bit field that configures the internal pull-up resistor status for the I/O lines.

- If you set a **PR** bit to 1, it enables the pull-up/down resistor
- If you set a **PR** bit to 0, it specifies no internal pull-up/down resistor.

PR and **PD** only affect lines that are configured as digital inputs (**3**) or disabled (**0**).

The following table defines the bit-field map for **PR** and **PD** commands.

Bit	I/O line	Micro pin	Surface-mount pin	Through-hole pin
0	DIO4	23	24	11
1	DIO3	28	30	17
2	DIO2	29	31	18
3	DIO1	30	32	19
4	DIO0	31	33	20
5	DIO6	27	29	16
6	DIO8	9	10	9
7	DIO14	4	4	3
8	DIO5	26	28	15

Bit	I/O line	Micro pin	Surface-mount pin	Through-hole pin
9	DIO9	25	26	13
10	DIO12	5	5	4
11	DIO10	7	7	6
12	DIO11	8	8	7
13	DIO7	24	25	12
14	DIO13	3	3	2
15	DIO15	16	17	N/A
16	DIO16	15	16	N/A
17	DIO17	14	15	N/A
18	DIO18	13	14	N/A
19	DIO19	11	12	N/A

Parameter range

Through-hole: 0 - 0xFFFF

SMT/MMT: 0 - 0xFFFFF

Default

0xFFFF

PD (Pull Up/Down Direction)

The resistor pull direction bit field (1 = pull-up, 0 = pull-down) for corresponding I/O lines that are set by the **PR** command.

See PR (Pull-up/Down Resistor Enable) for the bit mappings.

Parameter range

Through-hole: 0 - 0xFFFF

SMT/MMT: 0 - 0xFFFFF

Default

0xFFFF

IO (Set Digital I/O Lines)

Sets digital output levels. This allows DIO lines setup as outputs to be changed through Command mode.

Parameter range

8-bit bit map; each bit represents the level of an I/O line set up as an output

Default

N/A

M0 (PWM0 Duty Cycle)

The duty cycle of the PWM0 line (Micro pin 7/SMT pin 7).

If IA (I/O Input Address) is set correctly and P0 (DIO10/RSSI/PWM0 Configuration) is configured as PWM0 output, incoming AD0 samples automatically modify the PWM0 value. See PT (PWM Output Timeout).

To configure the duty cycle of PWM0:

1. Enable PWM0 output (**P0** = **2**).
2. Change **M0** to the desired value.
3. Apply settings (use **CN** or **AC**).

The PWM period is 64 µs and there are 0x03FF (1023 decimal) steps within this period. When **M0** = **0** (0% PWM), 0x01FF (50% PWM), 0x03FF (100% PWM), and so forth.

Parameter range

0 - 0x3FF

Default

0

M1 (PWM1 Duty Cycle)

If IA (I/O Input Address) is set correctly and P1 (DIO11/PWM1 Configuration) is configured as PWM1 output, incoming AD0 samples automatically modify the PWM1 value. See PT (PWM Output Timeout).

To configure the duty cycle of PWM1:

1. Enable PWM1 output (**P1** = 2).
2. Change **M1** to the desired value.
3. Apply settings (use **CN** or **AC**).

The PWM period is 64 µs and there are 0x03FF (1023 decimal) steps within this period. When **M0** = **0** (0% PWM), 0x01FF (50% PWM), 0x03FF (100% PWM), and so forth.

Parameter range

0 - 0x3FF

Default

0

RP (RSSI PWM Timer)

The PWM timer expiration in 0.1 seconds. **RP** sets the duration of pulse width modulation (PWM) signal output on the RSSI pin. The signal duty cycle updates with each received packet and shuts off when the timer expires.

When **RP** = **0xFF**, the output is always on.

Parameter range

0 - 0xFF (x 100 ms), 0xFF

Default

0x28 (four seconds)

LT (Associate LED Blink Time)

Set or read the Associate LED blink time. If you use D5 (DIO5/Associate Configuration) to enable the Associate LED functionality (DIO5/Associate pin), this value determines the on and off blink times for the LED when the device has joined the network.

If **LT** = **0**, the device uses the default blink rate of 250 ms.

For all other **LT** values, the firmware measures **LT** in 10 ms increments.

Parameter range

0, 0x14 - 0xFF (x 10 ms)

Default

0

I/O sampling commands

The following commands configure I/O sampling on an originating device. Any I/O sample generated by this device is sent to the address specified by **DH** and **DL**. You must configure at least one I/O line as an input or output for a sample to be generated.

IS (I/O Sample)

Immediately forces an I/O sample to be generated. If you issue the command to the local device, the sample data is sent out the local serial interface. If sent remotely, the sample data is returned as a Local AT Command Response - 0x88.

If the device receives ERROR as a response to an **IS** query, there are no valid I/O lines to sample.

The **IS** command cannot be issued from within MicroPython or over BLE.

Parameter range

N/A

Default

N/A

IR (Sample Rate)

Determines the I/O sample rate used to generate outgoing I/O sample data. When the IR value is greater than 0, the device samples and transmits all enabled digital I/O and ADCs every **IR** milliseconds. I/O Samples transmit to the address specified by **DH** +**DL**.

At least one I/O line must be configured as an input or explicit output for samples to be generated.

Parameter range

0, 0x32 - 0xFFFF (ms)

Default

0

IC (DIO Change Detect)

Set or read the digital I/O pins to monitor for changes in the I/O state.

IC works with the individual pin configuration commands (**D0** - **D9**, **P0** - **P4**). If the device detects a change on an enabled digital I/O pin, it immediately transmits a digital I/O sample to the address specified by **DH** + **DL**. If sleep is enabled, the edge transition must occur during a wake period to trigger a change detect.

The data transmission contains only DIO data.

IC is a bitmask you can use to enable or disable edge detection on individual digital I/O lines. Only DIO0 through DIO14 can be sampled using a Change Detect.

Set unused bits to **0**.

Bit field

Bit	I/O line	Micro pin	Surface-mount pin	Through-hole pin
0	DIO0	31	33	20
1	DIO1	30	32	19
2	DIO2	29	31	18
3	DIO3	28	30	17
4	DIO4	23	24	11
5	DIO5	26	28	15
6	DIO6	27	29	16
7	DIO7	24	25	12
8	DIO8	9	10	9
9	DIO9	25	26	13
10	DIO10	7	7	6
11	DIO11	8	8	7
12	DIO12	5	5	4
13	DIO13	3	3	2
14	DIO14	4	4	3
15	N/A	N/A	N/A	N/A

Parameter range

0 - 0x7FFF

Default

0

AV (Analog Voltage Reference)

The analog voltage reference used for A/D sampling.

ADC lines are 10-bit analog inputs.

Parameter range

0 - 2

Parameter	Description
0	1.25 V reference
1	2.5 V reference
2	VDD reference

Default

0

IF (Sleep Sample Rate)

Set or read the number of sleep cycles that must elapse between periodic I/O samples. This allows the firmware to take I/O samples only during some wake cycles. During those cycles, the firmware takes I/O samples at the rate specified by IR (Sample Rate). In addition, setting **IF** to zero allows I/O samples to occur before the device goes to sleep and occur thereafter every wake cycle specified by **IR**.

To enable periodic sampling, set **IR** to a non-zero value, and enable the analog or digital I/O functionality of at least one device pin. The sample rate is measured in milliseconds.

For more information, see the following commands:

- D0 (DIO0/ADC0/Commissioning Configuration) through D9 (DIO9/ON_SLEEP Configuration)
- P0 (DIO10/RSSI/PWM0 Configuration) through P2 (DIO12/TH_SPI_MISO Configuration)

Parameter range

0 - 0xFFFF (x 1 ms)

Default

1

I/O line passing commands

The following AT commands allow I/O line passing to be enabled and configure the timeout that will be used for each I/O line. Line Passing requires the device to receive an I/O sample from the address specified by **IA** and have an I/O lines configured as outputs that corresponds to inputs in the received I/O sample.

IA (I/O Input Address)

The source address of the device to which outputs are bound.

To disable I/O line passing, set all bytes to **0xFF**.

To allow any I/O packet addressed to this device (including broadcasts) to change the outputs, set **IA** to **0xFFFF**.

Parameter range

0 - 0xFFFF FFFF FFFF FFFF

Default

0xFFFFFFFFFFFFFFFF (I/O line passing disabled)

IU (Send I/O Sample to Serial Port)

Indicates whether or not I/O samples should be sent to the serial port. 0 suppresses output; 1 allows output (only if the device is in API mode).

Parameter range

0 - 1

Parameter	Description
0	Disabled
1	Enabled

Default

1

T0 (D0 Timeout)

Specifies how long pin D0 (DIO0/ADC0/Commissioning Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T1 (D1 Output Timeout)

Specifies how long pin D1 (DIO1/ADC1/TH_SPI_ATTN Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T2 (D2 Output Timeout)

Specifies how long pin D2 (DIO2/ADC2/TH_SPI_CLK Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T3 (D3 Output Timeout)

Specifies how long pin D3 (DIO3/ADC3/TH_SPI_SSEL Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T4 (D4 Output Timeout)

Specifies how long pin D4 (DIO4/TH_SPI_MOSI Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T5 (D5 Output Timeout)

Specifies how long pin D5 (DIO5/Associate Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T6 (D6 Output Timeout)

Specifies how long pin D6 (DIO6/RTS Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T7 (D7 Output Timeout)

Specifies how long pin D7 (DIO7/CTS Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T8 (D8 Timeout)

Specifies how long pin D8 (DIO8/DTR/SLP_Request Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T9 (D9 Timeout)

Specifies how long pin D9 (DIO9/ON_SLEEP Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

Q0 (P0 Timeout)

Specifies how long pin **P0** holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

Q1 (P1 Timeout)

Specifies how long pin P1 holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

Q2 (P2 Timeout)

Specifies how long pin P2 holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

PT (PWM Output Timeout)

Specifies how long both PWM outputs (**P0**, **P1**) output a given PWM signal before it reverts to the configured value (**M0**/**M1**). If set to **0**, there is no timeout. This timeout only affects these pins when they are configured as PWM output.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0xFF

Location commands

The following commands are user-defined parameters used to store the physical location of the deployed device.

LX (Location X—Latitude)

User-defined GPS latitude coordinates of the node that is displayed on Digi Remote Manager and Network Assistant.

Parameter range

0 - 15 ASCII characters

Default

One ASCII space character (0x20)

LY (Location Y—Longitude)

User-defined GPS longitude coordinates of the node that is displayed on Digi Remote Manager and Network Assistant.

Parameter range

0 - 15 ASCII characters

Default

One ASCII space character (0x20)

LZ (Location Z—Elevation)

User-defined GPS elevation of the node that is displayed on Digi Remote Manager and Network Assistant.

Parameter range

0 - 15 ASCII characters

Default

One ASCII space character (0x20)

Diagnostic commands – firmware/hardware Information

The following read-only commands are diagnostics that provide more information about the device.

VR (Firmware Version)

Reads the firmware version on a device.

Parameter range

0x3000 - 0x30FF [read-only]

Default

Set in the firmware

VL (Version Long)

Shows detailed version information including the application build date and time.

Parameter range

N/A

Default

N/A

VH (Bootloader Version)

Reads the bootloader version of the device.

Parameter range

N/A

Default

N/A

HV (Hardware Version)

Display the hardware version number of the device.

Parameter range

0 - 0xFFFF [read-only]

Pre-defined **HV** values for XBee 3 devices:

- 0x41 = XBee 3 Micro (MMT) and Surface Mount (SMT)
- 0x42 = XBee 3 Through Hole (TH)

Default

Set in the factory

%C (Hardware/Software Compatibility)

Specifies what firmware is compatible with this device's hardware. **%C** is compared to the to the "compatibility_number" field of the firmware configuration xml file. Firmware with a compatibility number lower than the value returned by **%C** cannot be loaded onto the board. If an invalid firmware is loaded, the device will not boot until a valid firmware is reloaded.

Parameter range

[read-only]

Default

N/A

R? (Power Variant)

Specifies whether the device is a PRO or Non-PRO variant.

- 0 = PRO (+19 dBm output power)
- 1 = Non-PRO (+8 dBm output power)

Parameter range

0, 1 [read-only]

Default

N/A

%V (Supply Voltage)

Reads the voltage on the Vcc pin in mV.

Parameter range

0 - 0xFFFF (in mV) [read only]

Default

N/A

TP (Temperature)

The current module temperature in degrees Celsius. The temperature is represented in two’s complement, as shown in the following example:

1 °C = 0x0001 and -1°C = 0xFFFF

Parameter range

0 - 0xFFFF (Celsius) [read-only]

Default

N/A

DD (Device Type Identifier)

Stores the Digi device type identifier value. Use this value to differentiate between multiple types of devices (for example, sensors or lights).

This command can optionally be included in network discovery responses by setting bit 1 of **NO**.

Parameter range

0 - 0xFFFFFFFF

Default

0x150000

0x160000

CK (Configuration CRC)

Reads the cyclic redundancy check (CRC) of the current AT command configuration settings to determine if the configuration has changed.

After a firmware update this command may return a different value.

Parameter range

0 - 0xFFFF [read-only]

Default

N/A

%P (Invoke Bootloader)

Forces the device to reset into the bootloader menu.

This command can only be issued locally.

Parameter range

N/A

Default

N/A

Memory access commands

This section details the executable commands that provide memory access to the device.

FR (Software Reset)

Resets the device. The device responds immediately with an **OK** and performs a reset 100 ms later.

If you issue **FR** while the device is in Command mode, the reset effectively exits Command mode.

Parameter range

N/A

Default

N/A

AC (Apply Changes)

This command applies changes to all command parameters configured in Command mode and also applies queued command parameter values set with 0x09 API queued command frames.

Any of the following also applies changes the same as issuing an **AC** command:

- Exiting Command mode with a **CN** command.
- Exiting Command mode via timeout.
- Receiving a 0x08 API command frame.
- Issuing a 0x08 Local AT Command API frame.
- Issuing a remote 0x17 AT Command API frame with option bit 1 set.

Example: Altering the UART baud rate with the **BD** command does not change the operating baud rate until after an **AC** command is received; at this point, the interface immediately changes baud rates.

Parameter range

N/A

Default

N/A

WR (Write)

Immediately writes parameter values to non-volatile flash memory so they persist through a power cycle. Operating network parameters are persistent and do not require a **WR** command for the device to reattach to the network.

Writing parameters to non-volatile memory does not apply the changes immediately. However, since the device uses non-volatile memory to determine initial configuration following reset, the written parameters are applied following a reset.

Note Once you issue a **WR** command, do not send any additional characters to the device until after you receive the **OK** response. Use the **WR** command sparingly; the device's flash supports a limited number of write cycles.

Parameter range

N/A

Default

N/A

RE (Restore Defaults)

Restore device parameters to factory defaults.

Parameter range

N/A

Default

N/A

Custom Default commands

The following commands are used to assign custom defaults to the device. Send RE (Restore Defaults) to restore custom defaults. You must send these commands as local AT commands, they cannot be set using Remote AT Command Request - 0x17.

%F (Set Custom Default)

When **%F** is received, the XBee 3 DigiMesh RF Module takes the next command received and applies it to both the current configuration and the custom defaults, so that when defaults are restored with RE (Restore Defaults) the custom value is used.

Parameter range

N/A

Default

N/A

!C (Clear Custom Defaults)

Clears all custom defaults. This command does not change the current settings, but only changes the defaults so that RE (Restore Defaults) restores settings to the factory values.

Parameter range

N/A

Default

N/A

R1 (Restore Factory Defaults)

Restores factory defaults, ignoring any custom defaults set using %F (Set Custom Default).

Parameter range

N/A

Default

N/A

Operate in API mode

API mode overview

As an alternative to Transparent operating mode, you can use API operating mode. API mode provides a structured interface where data is communicated through the serial interface in organized packets and in a determined order. This enables you to establish complex communication between devices without having to define your own protocol. The API specifies how commands, command responses and device status messages are sent and received from the device using the serial interface or the SPI interface.

We may add new frame types to future versions of the firmware, so we recommend building the ability to filter out additional API frames with unknown frame types into your software interface.

Use the AP command to set the operation mode

Use AP (API Enable) to specify the operation mode:

AP command setting	Description
AP = 0	Transparent operating mode, UART serial line replacement with API modes disabled. This is the default option.
AP = 1	API operation.
AP = 2	API operation with escaped characters (only possible on UART).

The API data frame structure differs depending on what mode you choose.

API frame format

An API frame consists of the following:

- Start delimiter
- Length
- Frame data
- Checksum

API operation (AP parameter = 1)

This is the recommended API mode for most applications. The following table shows the data frame structure when you enable this mode:

Frame fields	Byte	Description
Start delimiter	1	0x7E
Length	2 - 3	Most Significant Byte, Least Significant Byte
Frame data	4 - number (n)	API-specific structure
Checksum	n + 1	1 byte

Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the XBee replies with a radio status frame indicating the nature of the failure.

API operation with escaped characters (AP parameter = 2)

Setting API to 2 allows escaped control characters in the API frame. Due to its increased complexity, we only recommend this API mode in specific circumstances. API 2 may help improve reliability if the serial interface to the device is unstable or malformed frames are frequently being generated.

When operating in API 2, if an unescaped 0x7E byte is observed, it is treated as the start of a new API frame and all data received prior to this delimiter is silently discarded. For more information on using this API mode, see the Escaped Characters and API Mode 2 in the Digi Knowledge base.

API escaped operating mode works similarly to API mode. The only difference is that when working in API escaped mode, the software must escape any payload bytes that match API frame specific data, such as the start-of-frame byte (0x7E). The following table shows the structure of an API frame with escaped characters:

<table>
<tr><th>Frame fields</th><th>Byte</th><th colspan="2">Description</th></tr>
<tr><td>Start delimiter</td><td>1</td><td colspan="2">0x7E</td></tr>
<tr><td>Length</td><td>2 - 3</td><td>Most Significant Byte, Least Significant Byte</td><td rowspan="3">Characters escaped if needed</td></tr>
<tr><td>Frame data</td><td>4 - n</td><td>API-specific structure</td></tr>
<tr><td>Checksum</td><td>n + 1</td><td>1 byte</td></tr>
</table>

Start delimiter field

This field indicates the beginning of a frame. It is always 0x7E. This allows the device to easily detect a new incoming frame.

Escaped characters in API frames

If operating in API mode with escaped characters (**AP** parameter = 2), when sending or receiving a serial data frame, specific data values must be escaped (flagged) so they do not interfere with the data frame sequencing. To escape an interfering data byte, insert 0x7D and follow it with the byte to be escaped (XORed with 0x20).

The following data bytes need to be escaped:

- 0x7E: start delimiter
- 0x7D: escape character
- 0x11: XON
- 0x13: XOFF

To escape a character:

1. Insert 0x7D (escape character).
2. Append it with the byte you want to escape, XORed with 0x20.

In API mode with escaped characters, the length field does not include any escape characters in the frame and the firmware calculates the checksum with non-escaped data.

Example: escape an API frame

To express the following API non-escaped frame in API operating mode with escaped characters:

Start delimiter	Length	Frame type	Frame Data Data	Checksum
7E	00 0F	17	01 00 13 A2 00 40 AD 14 2E FF FE 02 4E 49	6D

You must escape the 0x13 byte:

1. Insert a 0x7D.
2. XOR byte 0x13 with 0x20: $13 \oplus 20 = 33$

The following figure shows the resulting frame. Note that the length and checksum are the same as the non-escaped frame.

Start delimiter	Length	Frame type	Frame Data Data	Checksum
7E	00 0F	17	01 00 7D 33 A2 00 40 AD 14 2E FF FE 02 4E 49	6D

The length field has a two-byte value that specifies the number of bytes in the frame data field. It does not include the checksum field.

Length field

The length field is a two-byte value that specifies the number of bytes contained in the frame data field. It does not include the checksum field.

Frame data

This field contains the information that a device receives or will transmit. The structure of frame data depends on the purpose of the API frame:

Start delimiter	Length		Frame data Frame type	Data							Checksum
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB	API frame type	Data							Single byte

- **Frame type** is the API frame type identifier. It determines the type of API frame and indicates how the Data field organizes the information.
- **Data** contains the data itself. This information and its order depend on the what type of frame that the Frame type field defines.

Multi-byte values are sent big-endian.

Calculate and verify checksums

To calculate the checksum of an API frame:

1. Add all bytes of the packet, except the start delimiter 0x7E and the length (the second and third bytes).
2. Keep only the lowest 8 bits from the result.
3. Subtract this quantity from 0xFF.

To verify the checksum of an API frame:

1. Add all bytes including the checksum; do not include the delimiter and length.
2. If the checksum is correct, the last two digits on the far right of the sum equal 0xFF.

Example

Consider the following sample data packet: **7E 00 0A 01 01 50 01 00 48 65 6C 6C 6F B8**+

Byte(s)	Description
7E	Start delimiter
00 0A	Length bytes
01	API identifier
01	API frame ID
50 01	Destination address low
00	Option byte
48 65 6C 6C 6F	Data packet
B8	Checksum

To calculate the check sum you add all bytes of the packet, excluding the frame delimiter **7E** and the length (the second and third bytes):

7E 00 0A 01 01 50 01 00 48 65 6C 6C 6F B8

Add these hex bytes:

01 + 01 + 50 + 01 + 00 + 48 + 65 + 6C + 6C + 6F = 247

Now take the result of 0x247 and keep only the lowest 8 bits which in this example is 0xC4 (the two far right digits). Subtract 0x47 from 0xFF and you get 0x3B (0xFF - 0xC4 = 0x3B). 0x3B is the checksum for this data packet.

If an API data packet is composed with an incorrect checksum, the XBee 3 DigiMesh RF Module will consider the packet invalid and will ignore the data.

To verify the check sum of an API packet add all bytes including the checksum (do not include the delimiter and length) and if correct, the last two far right digits of the sum will equal FF.

01 + 01 + 50 + 01 + 00 + 48 + 65 + 6C + 6C + 6F + B8 = 2FF

Frame descriptions

The following sections describe the API frames.

Local AT Command Request - 0x08

Response frame: Local AT Command Response - 0x88

Description

This frame type is used to query or set command parameters on the local device. Any parameter that is set with this frame type will apply the change immediately. If you wish to queue multiple parameter changes and apply them later, use the Queue Local AT Command Request - 0x09 instead.

When querying parameter values, this frame behaves identically to Queue Local AT Command Request - 0x09: You can query parameter values by sending this frame with a command but no parameter value field—the two-byte AT command is immediately followed by the frame checksum. When an AT command is queried, a Local AT Command Response - 0x88 frame is populated with the parameter value that is currently set on the device. The Frame ID of the 0x88 response is the same one set by the command in the 0x08 request frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Local AT Command Request - **0x08**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to**0**, the device will not emit a response frame.
5	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
7-n	variable	**Parameter value (optional)**	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the current parameter value and returns the result in the response.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Set the local command parameter

Set the **NI** string of the radio to "**End Device**".

The corresponding Local AT Command Response - 0x88 with a matching Frame ID will indicate whether the parameter change succeeded.

```
7E 00 0E 08 A1 4E 49 45 6E 64 20 44 65 76 69 63 65 38
```

Frame type	Frame ID	AT command	Parameter value
0x08	0xA1	0x4E49	0x456E6420446576696365
Request	*Matches response*	*"NI"*	*"End Device"*

Query local command parameter

Query the temperature of the module—**TP** command.

The corresponding Local AT Command Response - 0x88 with a matching Frame ID will return the temperature value.

```
7E 00 04 08 17 54 50 3C
```

Frame type	Frame ID	AT command	Parameter value
0x08	0x17	0x5450	(omitted)
Request	*Matches response*	*"TP"*	*Query the parameter*

Queue Local AT Command Request - 0x09

Response frame: Local AT Command Response - 0x88

Description

This frame type is used to query or set queued command parameters on the local device. In contrast to Local AT Command Request - 0x08, this frame queues new parameter values and does not apply them until you either:

- Issue a Local AT Command using the 0x08 frame
- Issue an **AC** command—queued or otherwise

When querying parameter values, this frame behaves identically to Local AT Command Request - 0x08: You can query parameter values by sending this frame with a command but no parameter value field—the two-byte AT command is immediately followed by the frame checksum. When an AT command is queried, a Local AT Command Response - 0x88 frame is populated with the parameter value that is currently set on the device. The Frame ID of the 0x88 response is the same one set by the command in the 0x09 request frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Queue Local AT Command Request - **0x09**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to**0**, the device will not emit a response frame.
5	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
7-n	variable	**Parameter value (optional)**	If present, indicates the requested parameter value to set the given register at a later time. If no characters are present, it queries the current parameter value and returns the result in the response.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Queue setting local command parameter

Set the UART baud rate to 115200, but do not apply changes immediately.

The device will continue to operate at the current baud rate until the change is applied with a subsequent **AC** command.

The corresponding Local AT Command Response - 0x88 with a matching Frame ID will indicate whether the parameter change succeeded.

```
7E 00 05 09 53 42 44 07 16
```

Frame type	Frame ID	AT command	Parameter value
0x09	0x53	0x4244	0x07
Request	*Matches response*	*"BD"*	*7 = 115200 baud*

Query local command parameter

Query the temperature of the module (**TP** command).

The corresponding 0x88 - Local AT Command Response frame with a matching Frame ID will return the temperature value.

```
7E 00 04 09 17 54 50 3B
```

Frame type	Frame ID	AT command	Parameter value
0x09	0x17	0x5450	(omitted)
Request	*Matches response*	*"TP"*	*Query the parameter*

Transmit Request - 0x10

Response frame: Extended Transmit Status - 0x8B

Description

This frame type is used to send payload data as an RF packet to a specific destination. This frame type is typically used for transmitting serial data to one or more remote devices.

The endpoints used for these data transmissions are defined by the **SE** and **EP** commands and the cluster ID defined by the **CI** command—excluding 802.15.4. To define the application-layer addressing fields on a per-packet basis, use the Explicit Addressing Command Request - 0x11 instead.

Query the **NP** command to read the maximum number of payload bytes that can be sent.

See Maximum payload for additional information on payload size restrictions.

64-bit addressing

- For broadcast transmissions, set the 64-bit destination address to **0x000000000000FFFF**
- For unicast transmissions, set the 64-bit address field to the address of the desired destination node

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Transmit Request - **0x10**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response frame. If set to **0**, the device will not emit a response frame.
5	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device. Broadcast address is **0x000000000000FFFF**.
13	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
15	8-bit	**Broadcast radius**	Sets the maximum number of hops a broadcast transmission can traverse. This parameter is only used for broadcast transmissions. If set to **0**—recommended—the value of **NH**specifies the broadcast radius.
16	8-bit	**Transmit**	See the Transmit options bit field table below for available

Offset	Size	Frame Field	Description
		options	options. If set to **0**, the value of **TO** specifies the transmit options.
17-n	variable	**Payload data**	Data to be sent to the destination device. Up to **NP** bytes per packet.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Transmit options bit field

The available transmit options vary depending on the protocol being used. Bitfield options can be combined. Set all unused bits to **0**.

DigiMesh

Bit	Meaning	Description
0	Disable ACK [**0x01**]	Disable acknowledgments on all unicasts.
1	Disable route discoveries [**0x02**]	Disable Route Discovery on all DigiMesh unicasts.
2	Unicast NACK [**0x04**]	Enable unicast NACK messages on DigiMesh transmissions When set, a failed transmission will generate a Route Information - 0x8D frame for diagnosis.
3	Unicast trace route [**0x08**]	Enable a unicast Trace Route on DigiMesh transmissions When set, the transmission will generate a Route Information - 0x8D frame.
4	Secure Session Encryption [**0x10**]	Encrypt payload for transmission across a Secure Session Reduces maximum payload size by 4 bytes.
5	Reserved	<set this bit to 0>
6,7	Delivery method	b’00 = <invalid option> b’01 = Point-multipoint [**0x40**] b’10 = Directed Broadcast [**0x80**] b’11 = DigiMesh [**0xC0**]

Examples

Each example is written without escapes (**AP**=**1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

64-bit unicast

Sending a unicast transmission to a device with the 64-bit address of **0013A20012345678** with the serial data "**TxData**". Transmit options are set to **0**, which means the transmission will send using the options set by the **TO** command.

The corresponding Transmit Status - 0x89 response with a matching Frame ID will indicate whether the transmission succeeded.

```
7E 00 14 10 52 00 13 A2 00 12 34 56 78 FF FE 00 00 54 78 44 61 74 61 91
```

Frame type	Frame ID	64-bit dest	Reserved	Bcast radius	Options	RF data
0x10	0x52	0x0013A200 12345678	0xFFFE	0x00	0x00	0x547844617461
Request	*Matches response*	*Destination*	*Unused*	*N/A*	*Will use **TO***	*"TxData"*

64-bit broadcast

Sending a broadcast transmission of the serial data "**Broadcast**" to neighboring devices and suppressing the corresponding response by setting Frame ID to **0**.

```
7E 00 17 10 00 00 00 00 00 00 00 00 FF FF FF FE 01 00 42 72 6F 61 64 63 61 73
74 60
```

Frame type	Frame ID	64-bit dest	Reserved	Bcast radius	Tx Options	RF data
0x10	0x00	0x00000000 0000FFFF	0xFFFE	0x01	0x00	0x42726F616463617374
Request	*Suppress response*	*Broadcast address*	*Unused*	*Single hop broadcast*	*Will use **TO***	*"Broadcast"*

Explicit Addressing Command Request - 0x11

Response frame: Extended Transmit Status - 0x8B

Description

This frame type is used to send payload data as an RF packet to a specific destination using application-layer addressing fields. The behavior of this frame is similar to Transmit Request - 0x10, but with additional fields available for user-defined endpoints, cluster ID, and profile ID.

This frame type is typically used for OTA updates, and serial data transmissions.

Query NP (Maximum Packet Payload Bytes) to read the maximum number of payload bytes that can be sent.

See Maximum payload for additional information on payload size restrictions.

64-bit addressing

- For broadcast transmissions, set the 64-bit destination address to **0x000000000000FFFF**
- For unicast transmissions, set the 64-bit address field to the address of the desired destination node

Reserved endpoints

For serial data transmissions, the **0xE8** endpoint should be used for both source and destination endpoints.

The active Digi endpoints are:

- **0xE8** - Digi Data endpoint
- **0xE6** - Digi Device Object (DDO) endpoint
- **0xE5** - XBee 3 - Secure Session Server endpoint
- **0xE4** - XBee 3 - Secure Session Client endpoint
- **0xE3** - XBee 3 - Secure Session SRP authentication endpoint

Reserved cluster IDs

For serial data transmissions, the **0x0011** cluster ID should be used.

The following cluster IDs can be used on the **0xE8** data endpoint:

- **0x0011**- Transparent data cluster ID
- **0x0012** - Loopback cluster ID:The destination node echoes any transmitted packet back to the source device.

Reserved profile IDs

The Digi profile ID of **0xC105** should be used when sending serial data between XBee devices.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Explicit Addressing Command Request - **0x11**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to**0**, the device will not emit a response frame.
5	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device. Broadcast address is **0x000000000000FFFF**.
13	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
15	8-bit	**Source Endpoint**	Source endpoint for the transmission. Serial data transmissions should use **0xE8**.
16	8-bit	**Destination Endpoint**	Destination endpoint for the transmission. Serial data transmissions should use **0xE8**.
17	16-bit	**Cluster ID**	The Cluster ID that the host uses in the transmission. Serial data transmissions should use **0x11**.
19	16-bit	**Profile ID**	The Profile ID that the host uses in the transmission. Serial data transmissions between XBee devices should use **0xC105**.
21	8-bit	**Broadcast radius**	Sets the maximum number of hops a broadcast transmission can traverse. This parameter is only used for broadcast transmissions. If set to **0** (recommended), the value of **NH** specifies the broadcast radius.
22	8-bit	**Transmit options**	See the Transmit options bit field table below for available options. If set to **0**, the value of **TO** specifies the transmit options.
23-n	variable	**Command data**	Data to be sent to the destination device. Up to **NP** bytes per packet.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Transmit options bit field

The available transmit options vary depending on the protocol being used. Bitfield options can be combined. Set all unused bits to **0**.

DigiMesh

Bit	Meaning	Description
0	Disable ACK [**0x01**]	Disable acknowledgments on all unicasts.
1	Disable route discoveries [**0x02**]	Disable Route Discovery on all DigiMesh unicasts.
2	Unicast NACK [**0x04**]	Enable unicast NACK messages on DigiMesh transmissions When set, a failed transmission will generate a Route Information - 0x8D frame for diagnosis.
3	Unicast trace route [**0x08**]	Enable a unicast Trace Route on DigiMesh transmissions When set, the transmission will generate a Route Information - 0x8D frame.
4	Secure Session Encryption [**0x10**]	Encrypt payload for transmission across a Secure Session Reduces maximum payload size by 4 bytes.
5	Reserved	<set this bit to 0>
6,7	Delivery method	b'00 = <invalid option> b'01 = Point-multipoint [**0x40**] b'10 = Directed Broadcast [**0x80**] b'11 = DigiMesh [**0xC0**]

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

64-bit unicast

Sending a unicast transmission to an XBee device with the 64-bit address of **0013A20012345678** with the serial data "**TxData**". Transmit options are set to **0**, which means the transmission will send using the options set by the **TO** command. This transmission is identical to a Transmit Request - 0x10 using default settings.

The corresponding Extended Transmit Status - 0x8B response with a matching Frame ID will indicate whether the transmission succeeded.

```
7E 00 1A 11 87 00 13 A2 00 12 34 56 78 FF FE E8 E8 00 11 C1 05 00 00 54 78 44
61 74 61 B4
```

Frame type	Frame ID	64-bit dest	Reserved	Source EP	Dest EP	Cluster	Profile	Bcast radius	Tx options	Command data
0x11	0x87	0x0013A200 12345678	0xFFFE	0xE8	0xE8	0x0011	0xC105	0x00	0x00	0x547844617461
Explicit request	*Matches response*	*Destination*	*Unused*	*Digi data*	*Digi data*	*Data*	*Digi profile*	*N/A*	*Use* ***TO***	*"TxData"*

Loopback Packet

Sending a loopback transmission to an device with the 64-bit address of **0013A20012345678** using Cluster ID **0x0012**. To better understand the raw performance, retries and acknowledgements are disabled.

The corresponding Extended Transmit Status - 0x8B response with a matching Frame ID can be used to verify that the transmission was sent.

The destination will not emit a receive frame, instead it will return the transmission back to the sender. The source device will emit the receive frame—the frame type is determined by the value of **AO**—if the packet looped back successfully.

```
7E 00 1A 11 F8 00 13 A2 00 12 34 56 78 FF FE E8 E8 00 12 C1 05 00 01 54 78 44
61 74 61 41
```

Frame type	Frame ID	64-bit dest	Reserved	Source EP	Dest EP	Cluster	Profile	Bcast radius	Tx options	Command data
0x11	0xF8	0x0013A200 12345678	0xFFFE	0xE8	0xE8	0x0012	0xC105	0x00	0x01	0x547844617461
Explicit request	*Matches response*	*Destination*	*Unused*	*Digi data*	*Digi data*	*Data*	*Digi profile*	*N/A*	*Disable retries*	*"TxData"*

Remote AT Command Request - 0x17

Response frame: 0x97 - Remote AT Command Response

Description

This frame type is used to query or set AT command parameters on a remote device.

For parameter changes on the remote device to take effect, you must apply changes, either by setting the **Apply Changes** options bit, or by sending an **AC** command to the remote.

When querying parameter values you can query parameter values by sending this framewith a command but no parameter value field—the two-byte AT command is immediately followed by the frame checksum. When an AT command is queried, a Remote AT Command Response- 0x97 frame is populated with the parameter value that is currently set on the device. The Frame ID of the 0x97 response is the same one set by the command in the 0x17 request frame.

XBee 3 DigiMesh RF Module firmwares support secured remote configuration through a Secure Session. Refer to Secured remote AT commands for information on how to secure your devices against unauthorized remote configuration.

Note Remote AT Command Requests should only be issued as unicast transmissions to avoid potential network disruption. Broadcasts are not acknowledged, so there is no guarantee all devices will receive the request. Responses are returned immediately by all receiving devices, which can cause congestion on a large network.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Remote AT Command Request - **0x17**.
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to **0**, the device will not emit a response frame.
5	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device. When using 16-bit addressing, set this field to **0xFFFFFFFFFFFFFFFF**.
13	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
15	8-bit	**Remote command options**	Bit field of options that apply to the remote AT command request: ■ **Bit 0**: Disable ACK [**0x01**]

Offset	Size	Frame Field	Description
			■ **Bit 1**: Apply changes on remote [**0x02**] • If not set, changes will not applied until the device receives an **AC** command or a subsequent command change is received with this bit set ■ Bit 2: Reserved (set to 0) ■ Bit 3: Reserved (set to 0) ■ **Bit 4**: Send the remote command securely [**0x10**] • Requires a secure session be established with the destination **Note** Option values may be combined. Set all unused bits to 0.
16	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
18-n	variable	**Parameter value (optional)**	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the current parameter value and returns the result in the response.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes—**AP** = **1**—and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Set remote command parameter

Set the **NI** string of a device with the 64-bit address of **0013A20012345678** to "**Remote**" and apply the change immediately.

The corresponding Remote AT Command Response- 0x97 with a matching Frame ID will indicate success.

```
7E 00 15 17 27 00 13 A2 00 12 34 56 78 FF FE 02 4E 49 52 65 6D 6F 74 65 F6
```

Frame type	Frame ID	64-bit dest	Reserved	Command options	AT command	Parameter value
0x17	0x27	0x0013A200 12345678	0xFFFE	0x02	0x4E49	0x52656D6F7465
Request	*Matches response*		*Unused*	*Apply Change*	*"NI"*	*"Remote"*

Queue remote command parameter change

Change the PAN ID of a remote device so it can migrate to a new PAN, since this change would cause network disruption, the change is queued so that it can be made active later with a subsequent **AC** command or written to flash with a queued **WR** command so the change will be active after a power cycle.

The corresponding Remote AT Command Response- 0x97 with a matching Frame ID will indicate success.

```
7E 00 11 17 68 00 13 A2 00 12 34 56 78 FF FE 00 49 44 04 51 D8
```

Frame type	Frame ID	64-bit dest	Reserved	Command options	AT command	Parameter value
0x17	0x68	0x0013A200 12345678	0xFFFE	0x00	0x4944	0x0451
Request	*Matches response*		*Unused*	*Queue Change*	*"ID"*	

Query remote command parameter

Query the temperature of a remote device—**TP** command.

The corresponding Remote AT Command Response- 0x97 with a matching Frame ID will return the temperature value.

```
7E 00 0F 17 FA 00 13 A2 00 12 34 56 78 FF FE 00 54 50 84
```

Frame type	Frame ID	64-bit dest	Reserved	Command options	AT command	Parameter value
0x17	0xFA	0x0013A200 12345678	0xFFFE	0x00	0x5450	(omitted)
Request	*Matches response*		*Unused*	*N/A*	*"TP"*	*Query the parameter*

User Data Relay Input - 0x2D

Response frame: Transmit Status - 0x89

Output frame: User Data Relay Output - 0xAD

Description

This frame type is used to relay user data between local interfaces: MicroPython (internal interface), BLE, or the serial port. Data relayed to the serial port—while in API mode—will be output as a User Data Relay Output - 0xAD frame.

For information and examples on how to relay user data using MicroPython, see Send and receive User Data Relay frames in the MicroPython Programming Guide.

For information and examples on how to relay user data using BLE, see Communicate with a Micropython application in the XBee Mobile SDK user guide.

Use cases

- You can use this frame to send data to an external processor through the XBee UART/SPI via the BLE connection. Use a cellphone to send the frame with UART interface as a target. Data contained within the frame is sent out the UART contained within an Output Frame. The external processor then receives and acts on the frame.
- Use an external processor to output the frame over the UART with the BLE interface as a target. This outputs the data contained in the frame as the Output Frame over the active BLE connection via indication.
- An external processor outputs the Frame over the UART with the Micropython interface as a target. Micropython operates over the data and publishes the data to mqtt topic.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	User Data Relay Input - **0x2D**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to **0**, the device will not emit a response frame.
5	8-bit	**Destination Interface**	The intended interface for the payload data: **0** = Serial port—SPI, or UART when in API mode **1** = BLE **2** = MicroPython
6-n	variable	**Data**	The user data to be relayed
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Error cases

Errors are reported in a Transmit Status - 0x89 frame that corresponds with the Frame ID of the Relay Data frame:

Error code	Error	Description
0x7C	Invalid Interface	The user specified a destination interface that does not exist or is unsupported.
0x7D	Interface not	The destination interface is a valid interface, but is not in a state that

Error code	Error	Description
	accepting frames	can accept data. For example: UART not in API mode, BLE does not have a GATT client connected, or buffer queues are full.

If the message was relayed successfully, no status will be generated.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Relay to MicroPython

A host device needs to pass the message "**Relay Data**" to a MicroPython application running on a local XBee device via the serial port.

A corresponding Transmit Status - 0x89 response with a matching Frame ID will indicate if there was a problem with relaying the data.

If successful, the XBee micropython application can call **relay.receive()** to retrieve the data.

```
7E 00 0D 2D 3D 02 52 65 6C 61 79 20 44 61 74 61 FC
```

Frame type	Frame ID	Destination interface	Data
0x2D	0x3D	0x02	0x52656C61792044617461
Input	*Matches response*	*MicroPython*	*"Relay Data"*

Secure Session Control - 0x2E

Response frame: 0xAE - Secure Session Response

Description

This frame type is used to control a secure session between a client and a server. If the remote node has a password set and you set the frame to login, this will establish a secure session that will allow secured messages to be passed between the server and client.

This frame is also used for clients to log out of an existing secure session.

Secure Sessions are end-to-end connections. If a login attempt is addressed to a broadcast address, the attempt will fail with an invalid value—status **0xA**—error.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Secure Session Control - **0x2E**
4	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device. Set to a broadcast address (**0x000000000000FFFF**) to affect all active incoming sessions.
12	8-bit	**Secure Session options**	Bit field of options that alter the session behavior: ■ **Bit 0**: Client-side control: • [**0x00**] = **Login** - Log in to a server as a client. ◦ If this bit is clear, the local device will act as a client and initiate SRP authentication with the target server. • [**0x01**] = **Logou**t - Log out of an existing session as a client. ◦ If this bit is set, the local device will attempt to end an existing client-side session with the target server. ◦ When set, all other options, the timeout field, and password will be ignored. ■ **Bit 1**: Server-side control: • [**0x02**] = **Terminate Session** - If this bit is set, the server will end active incoming session(s). ◦ The address field can be set to a specific node or the broadcast address can be used to end all incoming sessions. ◦ Use Extended Modem Status - 0x98 frames to manage multiple incoming sessions. ■ **Bit 2**: Timeout type: • [**0x00**] = **Fixed timeout** - The session terminates after the timeout period has elapsed. • [**0x04**] = **Inter-packet timeout** - The timeout is refreshed every time a secure transmission occurs between client and server. **Note** Option values may be combined. Set all unused bits to 0.
13	16-bit	**Timeout**	Timeout value for the secure session in units of $\frac{1}{10}$th second. Accepts up to **0x4650** (30 minutes). A session with a timeout of **0x0000** is considered a yielding

Offset	Size	Frame Field	Description
			session. Yielding sessions will never time out, but if a server receives a request to start a session when it has the maximum incoming sessions, the oldest yielding session will be ended by the server to make room for the new session. Sessions with non-zero timeouts will never be ended in this way.
15-n	variable	**Password**	The password set on the remote node—up to 64 ASCII characters. Will be ignored if this frame is a logout or server termination frame.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte—between length and checksum.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Secure Session Client - Login with fixed timeout

A change is needed to be made on a device that is secured against unauthorized configuration changes. A gateway that is authorized to make the change logs into the remote node for 5 minutes as a client using the following frame:

The corresponding Secure Session Response - 0xAE will indicate whether the login attempt succeeded.

```
7E 00 14 2E 00 13 A2 00 12 34 56 78 00 0B B8 50 41 53 53 57 4F 52 44 D2
```

Frame type	64-bit dest	Session options	Timeout	Password
0x2E	0x0013A200 12345678	0x00	0x02B8	0x50415353574F5244D2
Request		*Login Fixed*	*5 minutes*	*"PASSWORD"*

Secure Session Client - Login for streaming data

A large stream of data needs to be sent to a gateway that is secured against receiving unauthorized data. Because the data stream, and the gateway's ability to process the data is unknown, a Secure Session using a 60 second inter-packet timeout is established. The sending device logs into the gateway as a client using the following frame:

The corresponding Secure Session Response - 0xAE will indicate whether the login attempt succeeded.

```
7E 00 13 2E 00 00 00 00 00 00 00 00 04 02 58 52 6F 73 33 62 75 64 D1
```

Frame type	64-bit dest	Session options	Timeout	Password
0x2E	0x00000000	0x04	0x0258	0x526F7333627564

Frame type	64-bit dest	Session options	Timeout	Password
	00000000			
Request	*Zigbee coordinator*	*Login* *Inter-packet*	*60 seconds*	*"Ros3bud"*

Local AT Command Response - 0x88

Request frames:

- Local AT Command Request - 0x08
- Queue Local AT Command Request - 0x09

Description

This frame type is emitted in response to a local AT Command request. Some commands send back multiple response frames; for example, ND (Network Discover). Refer to individual AT command descriptions for details on API response behavior.

This frame is only emitted if the Frame ID in the request is non-zero.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Local AT Command Response - **0x88**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.
5	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
7	8-bit	**Command status**	Status code for the host's request: **0** = OK **1** = ERROR **2** = Invalid command **3** = Invalid parameter
8-n	variable	**Command data (optional)**	If the host requested a command parameter change, this field will be omitted. If the host queried a command by omitting the parameter value in the request, this field will return the value currently set on the device.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Set local command parameter

Host set the NI string of the local device to "**End Device**" using a 0x08 request frame.

The corresponding Local AT Command Response - 0x88 with a matching Frame ID is emitted as a response:

```
7E 00 05 88 01 4E 49 00 DF
```

Frame type	Frame ID	AT command	Command Status	Command data
0x88	0xA1	0x4E49	0x00	(omitted)
Response	*Matches request*	*"NI"*	*Success*	*Parameter changes return no data*

Query local command parameter

Host queries the temperature of the local device—TP command—using a 0x08 request frame.

The corresponding Local AT Command Response - 0x88 with a matching Frame ID is emitted with the temperature value as a response:

```
7E 00 07 88 01 54 50 00 FF FE D5
```

Frame type	Frame ID	AT command	Command Status	Command data
0x88	0x17	0x5450	0x00	0xFFFE
Response	*Matches request*	*"TP"*	*Success*	*-2 °C*

Modem Status - 0x8A

Description

This frame type is emitted in response to specific conditions. The status field of this frame indicates the device behavior.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Modem Status - **0x8A**
4	8-bit	**Modem status**	Complete list of modem statuses: **0x00** = Hardware reset or power up **0x01** = Watchdog timer reset **0x02** = Joined network **0x03** = Left network **0x06** = Coordinator started **0x07** = Network security key was updated **0x0B** = Network woke up **0x0C** = Network went to sleep **0x0D** = Voltage supply limit exceeded **0x0E** = Remote Manager connected **0x0F** = Remote Manager disconnected **0x11** = Modem configuration changed while join in progress **0x12** = Access fault **0x13** = Fatal error **0x3B** = Secure session successfully established **0x3C** = Secure session ended **0x3D** = Secure session authentication failed **0x3E** = Coordinator detected a PAN ID conflict but took no action **0x3F** = Coordinator changed PAN ID due to a conflict **0x32** = BLE Connect **0x33** = BLE Disconnect **0x34** = Bandmask configuration failed **0x35** = Cellular component update started **0x36** = Cellular component update failed **0x37** = Cellular component update completed **0x38** = XBee firmware update started **0x39** = XBee firmware update failed **0x3A** = XBee firmware update applying **0x40** = Router PAN ID was changed by coordinator due to a conflict

Offset	Size	Frame Field	Description
			0x42 = Network Watchdog timeout expired **0x80** through **0xFF** = Stack error Refer to the tables below for a filtered list of status codes that are appropriate for specific devices.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Modem status codes

Statuses for specific modem types are listed here.

XBee DigiMesh

0x00 = Hardware reset or power up

0x01 = Watchdog timer reset

0x0B = Network woke up

0x0C = Network went to sleep

0x0D = Voltage supply limit exceeded—see **Over-voltage detection** in the *XBee 3 RF Module Hardware Reference Manual*.

0x3B = XBee 3 - Secure session successfully established

0x3C = XBee 3 - Secure session ended

0x3D = XBee 3 - Secure session authentication failed

0x32 = XBee 3 - BLE Connect

0x33 = XBee 3 - BLE Disconnect

0x34 = XBee 3 - No Secure Session Connection

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Boot status

When a device powers up, it returns the following API frame:

```
7E 00 02 8A 00 75
```

Frame type	Modem Status
0x8A	0x00
Status	*Hardware Reset*

Extended Transmit Status - 0x8B

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type is emitted when a network transmission request completes. The status field of this frame indicates whether the request succeeded or failed and the reason. This frame type provides additional networking details about the transmission.

This frame is only emitted if the Frame ID in the request is non-zero.

Note Broadcast transmissions are not acknowledged and always return a status of **0x00**, even if the delivery failed.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Transmit Status - **0x8B**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.
5	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
7	8-bit	**Transmit retry count**	The number of application transmission retries that occur.
8	8-bit	**Delivery status**	Complete list of delivery statuses: **0x00** = Success **0x01** = MAC ACK failure **0x02** = CCA/LBT failure **0x03** = Indirect message unrequested / no spectrum available **0x21** = Network ACK failure **0x25** = Route not found **0x31** = Internal resource error **0x32** = Resource error lack of free buffers, timers, etc. **0x74** = Data payload too large **0x75** = Indirect message unrequested Refer to the tables below for a filtered list of status codes that are

Offset	Size	Frame Field	Description
			appropriate for specific devices.
9	8-bit	**Discovery status**	Complete list of delivery statuses: **0x00** = No discovery overhead **0x02** = Route discovery
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Delivery status codes

Protocol-specific status codes follow

XBee DigiMesh

0x00 = Success

0x01 = MAC ACK Failure

0x02 = CCA/LBT Failure

0x03 = No 868 MHz spectrum available

0x21 = Network ACK Failure

0x25 = Route not found

0x31 = Internal resource error

0x32 = Internal error

0x34 = XBee 3 - No Secure Session Connection

0x35 = Encryption Failure

0x74 = Data payload too large

0x75 = Indirect message unrequested

Route Information - 0x8D

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type contains the DigiMesh routing information for a remote device on the network. This route information can be used to diagnose marginal links between devices across multiple hops.

This frame type is emitted in response to a DigiMesh unicast transmission request which has Trace Routing or NACK enabled. See Trace route option and NACK messages for more information.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Route Information - **0x8D**
4	8-bit	**Source event**	Event that caused the route information to be generated: **0x11** = NACK **0x12** = Trace route
5	8-bit	**Data length**	The number of bytes that follow, excluding the checksum. If the length increases, new items have been added to the end of the list for future revisions.
6	32-bit	**Timestamp**	System timer value on the node generating the Route Information Packet.The timestamp is in microseconds. Only use this value for relative time measurements because the time stamp count restarts approximately every hour.
10	8-bit	**ACK timeout count**	The number of MAC ACK timeouts that occur.
11	8-bit	**TX blocked count**	The number of times the transmission was blocked due to reception in progress.
12	8-bit	**Reserved**	Not used.
14	64-bit	**Destination address**	The 64-bit IEEE address of the final destination node of this network-level transmission.
21	64-bit	**Source**	The 64-bit IEEE address of the source node of this network-level

Offset	Size	Frame Field	Description
		address	transmission.
29	64-bit	**Responder address**	The 64-bit IEEE address of the node that generates this Route Information packet after it sends (or attempts to send) the data packet to the next hop (the Receiver node).
37	64-bit	**Receiver address**	The 64-bit IEEE address of the node that the device sends (or attempts to send) the data packet.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Routing information

The following example represents a possible Route Information Packet. A device emits this frame when it performs a trace route enabled transmission from one device—serial number 0x0013A200 4052AAAA—to another—serial number 0x0013A200 4052DDDD—across a DigiMesh network.

This particular frame indicates that the network successfully forwards the transmission from one device—serial number 0x0013A200 4052BBBB—to another device—serial number 0x0013A200 4052CCCC.

```
7E 00 2A 8D 12 27 6B EB CA 93 00 00 00 00 13 A2 00 40 52 DD DD 00 13 A2 00 40
52 AA AA 00 13 A2 00 40 52 BB BB 00 13 A2 00 40 52 CC CC 4E
```

Frame type	Source event	Data length	Timestamp	ACK timeout	TX Blocked	Reserved	Dest address	Source address	Responder address	Receiver address
0x8D	0x12	0x27	0x6BEBCA93	0x00	0x00	0x00	0x0013A200 4052DDDD	0x0013A200 4052AAAA	0x0013A200 4052BBBB	0x0013A200 4052CCCC
Route	*Trace Route*		*~30 minutes*	*No retries this hop*	*No error*	*N/A*	*Destination*	*Source*	*Node that sent this information*	*Next hop*

Aggregate Addressing Update - 0x8E

Description

This frame type is emitted on devices that update it addressing information in response to a network aggregator issuing an addressing update. A network aggregator is defined by a device on the network who has had the AG (Aggregator Support) command issued. A device on the network who's current **DH** and **DL** matches the address provided in the **AG** command request will update **DH** and **DL** and emit this frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Aggregate Addressing Update - **0x8E**
4	8-bit	**Reserved**	Reserved for future functionality. This field returns 0.
5	64-bit	**New address**	Address to which **DH** and **DL** are being set.
13	64-bit	**Old address**	Address to which **DH** and **DL** were previously set.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Aggregate address update

In the following example, a device with destination address (**DH**/**DL**) of 0x0013A200 4052AAAA updates its destination address to 0x0013A200 4052BBBB.

```
7E 00 12 8E 00 00 13 A2 00 40 52 BB BB 00 13 A2 00 40 52 AA AA 19
```

Frame type	Reserved	New address	Old address
0x8E	0x00	0x0013A200 4052BBBB	0x0013A200 4052AAAA
Update	*N/A*	*What **DH**/**DL** is now set to*	*What **DH**/**DL** was set to*

Transmit Status - 0x89

Request frames:

- TX Request: 64-bit address frame - 0x00
- TX Request: 16-bit address - 0x01
- User Data Relay Input - 0x2D

Description

This frame type is emitted when a transmit request completes. The status field of this frame indicates whether the request succeeded or failed and the reason.

This frame is only emitted if the Frame ID in the request is non-zero.

Note Broadcast transmissions are not acknowledged and always return a status of **0x00**, even if the delivery failed.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Transmit Status - **0x89**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.
5	8-bit	**Delivery status**	Complete list of delivery statuses: **0x00** = Success **0x01** = No ACK received **0x02** = CCA failure **0x03** = Indirect message unrequested **0x04** = Transceiver was unable to complete the transmission **0x21** = Network ACK failure **0x22** = Not joined to network **0x2C** = Invalid frame values (check the phone number) **0x31** = Internal error

Offset	Size	Frame Field	Description
			0x32 = Resource error - lack of free buffers, timers, etc. **0x34** = No Secure Session Connection **0x35** = Encryption Failure **0x74** = Message too long **0x76** = Socket closed unexpectedly **0x78** = Invalid UDP port **0x79** = Invalid TCP port **0x7A** = Invalid host address **0x7B** = Invalid data mode **0x7C** = Invalid interface. See User Data Relay Input - 0x2D. **0x7D** = Interface not accepting frames. See User Data Relay Input - 0x2D. **0x7E** = A modem update is in progress. Try again after the update is complete. **0x80** = Connection refused **0x81** = Socket connection lost **0x82** = No server **0x83** = Socket closed **0x84** = Unknown server **0x85** = Unknown error **0x86** = Invalid TLS configuration (missing file, and so forth) **0x87** = Socket not connected **0x88** = Socket not bound Refer to the tables below for a filtered list of status codes that are appropriate for specific devices.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Delivery status codes

Protocol-specific status codes follow

XBee 3 DigiMesh

This frame type is only used for indicating errors in sending a User Data Relay request

0x7C = Invalid interface. See User Data Relay Input - 0x2D.

0x7D = Interface not accepting frames. See User Data Relay Input - 0x2D.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Successful transmission

Host sent a unicast transmission to a remote device using a TX Request: 64-bit address frame - 0x00 frame.

The corresponding 0x89 Transmit Status with a matching Frame ID is emitted as a response to the request:

```
7E 00 03 89 52 00 24
```

Frame type	Frame ID	Delivery status
0x89	0x52	0x00
Response	*Matches request*	*Success*

Receive Packet - 0x90

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type is emitted when a device configured with standard API output—AO (API Options) = **0**—receives an RF data packet.

Typically this frame is emitted as a result of a device on the network sending serial data using the Transmit Request - 0x10 or Explicit Addressing Command Request - 0x11 addressed either as a broadcast or unicast transmission.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Receive Packet - **0x90**
4	64-bit	**64-bit source address**	The sender's 64-bit address.
12	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
14	8-bit	**Receive options**	Bit field of options that apply to the received message: ■ **Bit 0**: Packet was Acknowledged [**0x01**] ■ **Bit 1**: Packet was sent as a broadcast [**0x02**] ■ **Bit 2**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] ■ **Bit 3**: Reserved ■ **Bit 4**: Packet was sent across a secure session [0x10] ■ **Bit 5**: Reserved ■ **Bit 6**: Reserved ■ **Bit 6, 7**: DigiMesh delivery method • b'00 = <invalid option> • b'01 = Point-multipoint [**0x40**] • b'10 = Directed Broadcast [**0x80**] • b'11 = DigiMesh [**0xC0**]

Offset	Size	Frame Field	Description
			Note Option values may be combined.
15-n	variable	**Received data**	The RF payload data that the device receives.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

64-bit unicast

A device with the 64-bit address of **0013A20041AEB54E** sent a unicast transmission to a specific device with the payload of "**TxData**". The following frame is emitted if the destination is configured with **AO** = **0**.

```
7E 00 12 90 00 13 A2 00 41 AE B5 4E FF FE C1 54 78 44 61 74 61 C4
```

Frame type	64-bit source	Reserved	Rx options	Received data
0x90	0x0013A200 41AEB54E	0xFFFE	0xC1	0x547844617461
Output		*Unused*	*ACK was sent in DigiMesh mode*	*"TxData"*

Explicit Receive Indicator - 0x91

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type is emitted when a device configured with explicit API output—AO (API Options) bit1 set—receives a packet.

Typically this frame is emitted as a result of a device on the network sending serial data using the Transmit Request - 0x10 or Explicit Addressing Command Request - 0x11 addressed either as a broadcast or unicast transmission.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Explicit Receive Indicator - **0x91**
4	64-bit	**64-bit source address**	The sender's 64-bit address.
12	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
14	8-bit	**Source endpoint**	Endpoint of the source that initiated transmission.
15	8-bit	**Destination endpoint**	Endpoint of the destination that the message is addressed to.
16	16-bit	**Cluster ID**	The Cluster ID that the frame is addressed to.
18	16-bit	**Profile ID**	The Profile ID that the fame is addressed to.
20	8-bit	**Receive options**	Bit field of options that apply to the received message for packets sent using Digi endpoints (0xDC-0xEE): ■ **Bit 0**: Packet was Acknowledged [**0x01**] ■ **Bit 1**: Packet was sent as a broadcast [**0x02**] ■ **Bit 2**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] ■ **Bit 4**: Packet was sent across a secure session [**0x10**]

Offset	Size	Frame Field	Description
			■ **Bit 5**: Reserved ■ **Bit 6**: Reserved ■ **Bit 6, 7**: DigiMesh delivery method • b’00 = <invalid option> • b’01 = Point-multipoint [**0x40**] • b’10 = Directed Broadcast [**0x80**] • b’11 = DigiMesh [**0xC0**] **Note** Option values may be combined.
21-n	variable	**Received data**	The RF payload data that the device receives.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

64-bit unicast

A device with the 64-bit address of **0013A20087654321** sent a unicast transmission to a specific device with the payload of "**TxData**". The following frame is emitted if the destination is configured with **AO** > **1**.

```
7E 00 18 91 00 13 A2 00 41 AE B5 4E FF FE E8 E8 00 11 C1 05 C1 54 78 44 61 74
61 1C
```

Frame type	64-bit source	Reserved	Source EP	Dest EP	Cluster	Profile	Rx options	Received data
0x91	0x0013A200 41AEB54E	0x87BD	0xE8	0xE8	0x0011	0xC105	0xC1	0x54784461746 1
Explicit output		*Unused*	*Digi data*	*Digi data*	*Data*	*Digi profile*	*ACK was sent in DigiMesh network*	*"TxData"*

I/O Sample Indicator - 0x92

Description

This frame type is emitted when a device configured with standard API output—AO (API Options) = **0**—receives an I/O sample frame from a remote device. Only devices running in API mode will send I/O samples out the serial port.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	I/O Sample Indicator - **0x92**
4	64-bit	**64-bit source address**	The sender's 64-bit IEEE address.
12	16-bit	**Reserved**	Unused, but typically **0XFFFE**.
14	8-bit	**Receive options**	Bit field of options that apply to the received message: ▪ **Bit 0**: Packet was Acknowledged [**0x01**] ▪ **Bit 1**: Packet was sent as a broadcast [**0x02**] Note Option values may be combined.
15	8-bit	**Number of samples**	The number of sample sets included in the payload. This field typically reports 1 sample.
16	16-bit	**Digital sample mask**	Bit field that indicates which I/O lines on the remote are configured as digital inputs or outputs, if any: **bit 0**: DIO0 **bit 1**: DIO1 **bit 2**: DIO2 **bit 3**: DIO3 **bit 4**: DIO4 **bit 5**: DIO5 **bit 6**: DIO6 **bit 7**: DIO7 **bit 8**: DIO8 **bit 9**: DIO9 **bit 10**: DIO10 **bit 11**: DIO11

Offset	Size	Frame Field	Description
			bit 12: DIO12 **bit 13**: DIO13 **bit 14**: DIO14 bit 15: N/A For example, a digital channel mask of **0x002F** means DIO **0**, **1**, **2**, **3**, and **5** are enabled as digital I/O.
18	8-bit	**Analog sample mask**	Bit field that indicates which I/O lines on the remote are configured as analog input, if any: **bit 0**: AD0 **bit 1**: AD1 **bit 2**: AD2 **bit 3**: AD3 **bit 7**: Supply Voltage (enabled with **V+** command)
19	16-bit	**Digital samples (if included)**	If the sample set includes any digital I/O lines (**Digital channel mask** > **0**), this field contain samples for all enabled digital I/O lines. If no digital lines are configured as inputs or outputs, this field will be omitted. DIO lines that do not have sampling enabled return 0. Bits in this field are arranged the same as they are in the Digital channel mask field.
22	16-bit variable	**Analog samples (if included)**	If the sample set includes any analog I/O lines (Analog channel mask > 0), each enabled analog input returns a 16-bit value indicating the ADC measurement of that input. Analog samples are ordered sequentially from AD0 to AD3.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

I/O sample

A device with the 64-bit address of **0013A20012345678** is configured to periodically send I/O sample data to a particular device. The device is configured with DIO3, DIO4, and DIO5 configured as digital I/O, and AD1 and AD2 configured as an analog input.

The destination will emit the following frame:

```
7E 00 16 92 00 13 A2 00 12 34 56 78 FF FE C1 01 00 38 06 00 28 02 25 00 F8 E8
```

Frame type	64-bit source	Reserved	Rx options	Num samples	Digital channel mask	Analog channel mask	Digital samples	Analog sample 1	Analog sample 2
0x92	0x0013A20	0x87AC	0xC1	0x01	0x0038	0x06	0x0028	0x0225	0x00F8

Frame type	64-bit source	Reserved	Rx options	Num samples	Digital channel mask	Analog channel mask	Digital samples	Analog sample 1	Analog sample 2
	0 12345678								
Sample		*Unused*	*ACK was sent in mesh network*	*Single sample (typical)*	*b'00 **111**000 DIO3, DIO4, and DIO5 enabled*	*b'0**11**0 AD1 and AD2 enabled*	*b'00 **101**000 DIO3 and DIO5 are HIGH; DI04 is LOW*	*AD1 data*	*AD2 data*

Node Identification Indicator - 0x95

Description

This frame type is emitted when a node identification broadcast is received. The node identification indicator contains information about the identifying device, such as address, identifier string (**NI**), and other relevant data.

A node identifies itself to the network under these conditions:

- The commissioning button is pressed once.
- A **CB 1** command is issued.

See ND (Network Discover) for information on the payload formatting.

See NO (Network Discovery Options) for configuration options that modify the output of this frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Node Identification Indicator - **0x95**
4	64-bit	**64-bit source address**	The sender's 64-bit address.
12	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
14	8-bit	**Options**	Bit field of options that apply to the received message: ■ Bit 0: Reserved ■ **Bit 1**: Packet was sent as a broadcast [**0x02**] ■ **Bit 2**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] ■ Bit 4: Reserved ■ Bit 5: Reserved ■ **Bit 6, 7**: DigiMesh delivery method • b’00 = <invalid option> • b’01 = Point-multipoint [**0x40**] • b’10 = Directed Broadcast [**0x80**] • b’11 = DigiMesh [**0xC0**] **Note** Option values may be combined.

Offset	Size	Frame Field	Description
15	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
17	64-bit	**64-bit remote address**	The 64-bit address of the device that sent the Node Identification.
25	variable (2-byte minimum)	**Node identification string**	Node identification string on the remote device set by NI (Network Identifier). The identification string is terminated with a NULL byte (0x00).
27+NI	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
29+NI	8-bit	**Network device type**	What type of network device the remote identifies as: 0 = Coordinator 1 = Router 2 = End Device
30+NI	8-bit	**Source event**	The event that caused the node identification broadcast to be sent. 0 = Reserved 1 = Frame sent by node identification pushbutton event—see D0 (DIO0/ADC0/Commissioning Configuration).
31+NI	16-bit	**Digi Profile ID**	The Digi application Profile ID—**0xC105**.
33+NI	16-bit	**Digi Manufacturer ID**	The Digi Manufacturer ID—**0x101E**.
35+NI	32-bit	**Device type identifier (optional)**	The user-defined device type on the remote device set by DD (Device Type Identifier). Only included if the receiving device has the appropriate NO (Network Discovery Options) bit set.
EOF-1	8-bit	**RSSI (optional)**	The RSSI of the last hop that relayed the message. Only included if the receiving device has the appropriate NO (Network Discovery Options) bit set.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte—between length and checksum.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Identify remote device

A technician is replacing a DigiMesh device in the field and needs to have the its entry removed from a cloud server's database. The technician pushes the commissioning button on the old device once to send an identification broadcast. The server can use the broadcast to identify which device is being replaced and perform the necessary action.

When the node identification broadcast is sent, every device that receives the message will flash the association LED and emit the following information frame:

```
7E 00 27 95 00 13 A2 00 12 34 56 78 FF FE C2 FF FE 00 13 A2 00 12 34 56 78 4C
48 37 35 00 FF FE 01 01 C1 05 10 1E 00 14 00 08 0D
```

Frame type	64-bit source	16-bit source	Options	16-bit remote	64-bit remote	NI String	Parent	Device type	Event	Profile ID	MFG ID
0x95	0x0013 A200 12345678	0xFFFE	0xC2	0xFFFE	0x0013 A200 12345678	0x4C483735 00	0xFFFE	0x01	0x01	0xC105	0x101E
Identification		*Unknown*	*DigiMesh broadcast*	*Unknown*		*"LH75" + null*	*Unknown*	*Router*	*Button press*	*Digi*	*Digi*

Remote AT Command Response- 0x97

Request frame: Remote AT Command Request - 0x17

Description

This frame type is emitted in response to a Remote AT Command Request - 0x17. Some commands send back multiple response frames; for example, the **ND** command. Refer to individual AT command descriptions for details on API response behavior.

This frame is only emitted if the Frame ID in the request is non-zero.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Remote AT Command Response - **0x97**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.
5	64-bit	**64-bit source address**	The sender's 64-bit address.
13	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
15	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
17	8-bit	**Command status**	Status code for the host's request: **0x00** = OK **0x01** = ERROR **0x02** = Invalid command **0x03** = Invalid parameter **0x04** = Transmission failure **0x0C** = Encryption error
18-n	variable	**Parameter value (optional)**	If the host requested a command parameter change, this field will be omitted. If the host queried a command by omitting the parameter value in the request, this field will return the value currently set on the device.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Set remote command parameter

Host set the **NI** string of a remote device to "**Remote**" using a Remote AT Command Request - 0x17.

The corresponding 0x97 Remote AT Command Response with a matching Frame ID is emitted as a response:

```
7E 00 0F 97 27 00 13 A2 00 12 34 56 78 12 7E 4E 49 00 51
```

Frame type	Frame ID	64-bit source	Reserved	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0x127E	0x4E49	0x00	(omitted)
Response	*Matches request*		*Unused*	*"NI"*	*Success*	*Parameter changes return no data*

Transmission failure

Host queued the the PAN ID change of a remote device using a Remote AT Command Request - 0x17. Due to existing network congestion, the host will retry any failed attempts.

The corresponding 0x97 Remote AT Command Response with a matching Frame ID is emitted as a response:

```
7E 00 0F 97 27 00 13 A2 00 12 34 56 78 FF FE 49 44 04 EA
```

Frame type	Frame ID	64-bit source	Reserved	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0xFFFE	0x4944	0x04	(omitted)
Response	*Matches request*		*Unused*	*"ID"*	*Transmission failure*	*Parameter changes return no data*

Query remote command parameter

Query the temperature of a remote device—TP (Temperature).

The corresponding 0x97 Remote AT Command Response with a matching Frame ID is emitted with the temperature value as a response:

```
7E 00 11 97 27 00 13 A2 00 12 34 56 78 FF FE 54 50 00 00 2F A8
```

Frame type	Frame ID	64-bit source	Reserved	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0x0013A200 12345678	0x4944	0x00	0x002F
Response	*Matches request*		*Unused*	*"TP"*	*Success*	*+47 °C*

Extended Modem Status - 0x98

Description

This frame type can be used to manage and troubleshoot Secure Session connections. To enable extended modes statuses set AZ (Extended API Options) bit 3.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Extended Modem Status - **0x98**
4	8-bit	**Status code**	Refer to the tables below for appropriate status codes
n	variable	**Status data (optional)**	Additional fields that provide information about the status
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Secure Session status codes

When AZ (Extended API Options) is configured to output extended secure session statuses, whenever Secure Session API Frames are emitted, the extended modem status will provide additional details about the event.

Status code	Description	Status data	Size	Description
0x3B	A Secure Session was established with this node	Address	64-bit	The address of the client in the session.
		Options	8-bit	Session options set by the client.
		Timeout	16-bit	Session timeout set by the client.

Status code	Description	Status data	Size	Description
0x3C	A Secure Session ended	Address	64-bit	The address of the other node in this session.
		Reason	8-bit	The reason the session was ended: **0x00** - Session was terminated by the other node **0x01** - Session Timed out **0x02** - Received a transmission with an invalid encryption counter **0x03** - Encryption counter overflow - the maximum number of transmissions for a single session has been reached **0x04** - Remote node out of memory
0x3D	A Secure Session authentication attempt failed	Address	64-bit	Address of the client node.
		Error	8-bit	Error that caused the authentication to fail. See Secure Session Response - 0xAE for a list of error statuses.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Secure Session established

A device has established a secure session with the local node that has AZ (Extended API Options) configured to output extended secure session information. The following frame is emitted that announces the secure session establishment.

```
7E 00 0D 98 3B 00 13 A2 00 12 34 56 78 00 46 50 CD
```

Frame type	Status code	Status data
0x98	0x3B	■ 0x0013A20012345678 ■ 0x00 ■ 0x4650
Extended status	*Secure Session established*	■ Address ■ Options ■ Timeout (30 min)

Zigbee Verbose join messages

The following example shows a successful association of a device that has configured to enable Verbose Join messages. The device is operating in Transparent mode—**AP** = **0**—to allow a human-friendly way to troubleshoot association issues, if set for API mode—**AP** = **1**—equivalent 0x98 Extended Modem Status frames would be emitted.

Message	Description
V AI -SearchingforParent:FF	...search has started
V Scanning:03FFF800	...channels 11 through 25 are enabled by the SC setting for the Active Search.
V BeaconRsp:0000000000000042A6010B949AC8FF	■ **ZS** = 0x00 ■ extendedPanId = 00000000000042A6 ■ allowingJoin 0x01 (yes) ■ radiochannel 0x0B ■ panid 0x949A ■ rssi 0xC8 ■ lqi = 0xFF
V Reject ID	...beacon response's extendedPanId does not match this radio's ID setting of 3151
V BeaconRsp:0200000000000002AB010C55D2B2DB	■ **ZS** = 0x02 ■ extendedPanId = 0x00000000000002AB ■ allowingJoin = 0x01 (yes) ■ radiochannel = 0x0C ■ panid = 0x55D2 ■ rssi = 0xB2 ■ lqi = 0xDB
V Reject ZS	...beacon response's **ZS** does not match this radio's **ZS** setting
V BeaconRsp:000000000000003151010EE29FDFFF	
V BeaconSaved:0E05E29F0000000000003151	...this beacon response is acceptable as a candidate for association
V Joining:0E05E29F0000000000003151	...sending association request
V StackStatus: joined, network up 0290	...we are joined, the network is up, we can send and transmit
V Joined unsecured network:	
V AI -AssociationSucceeded:00	

User Data Relay Output - 0xAD

Input frame: User Data Relay Input - 0x2D

Description

This frame type is emitted when user data is relayed to the serial port from a local interface: MicroPython (internal interface), BLE, or the serial port.

For information and examples on how to relay user data using MicroPython, see Send and receive User Data Relay frames in the *MicroPython Programming Guide*.

for information and examples on how to relay user data using BLE, see Communicate with a Micropython application in the *XBee Mobile SDK user guide*.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	User Data Relay Output - **0xAD**
4	8-bit	**Source Interface**	The intended interface for the payload data: **0** = Serial port—SPI, or UART when in API mode **1** = BLE **2** = MicroPython
5-n	variable	**Data**	The user data to be relayed
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Error cases

Errors are reported in a Transmit Status - 0x89 frame that corresponds with the Frame ID of the Relay Data frame:

Error code	Error	Description
0x7C	Invalid Interface	The user specified a destination interface that does not exist or is unsupported.
0x7D	Interface not accepting frames	The destination interface is a valid interface, but is not in a state that can accept data. For example: UART not in API mode, BLE does not have a GATT client connected, or buffer queues are full.

If the message was relayed successfully, no status will be generated.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Relay from Bluetooth (BLE)

A mobile phone sends a serial data message to the XBee device's BLE interface. The message is flagged to be sent out of the serial port of the XBee device. The following frame outputs the relayed data:

```
7E 00 0C AD 01 52 65 6C 61 79 20 44 61 74 61 BA
```

Frame type	Source interface	Data
0xAD	0x01	0x52656C61792044617461
Output	*Bluetooth*	*"Relay Data"*

Secure Session Response - 0xAE

Request frame: Secure Session Control - 0x2E

Description

This frame type is output as a response to a Secure Session Control - 0x2E attempt. It indicates whether the Secure Session operation was successful or not.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Secure Session Response - **0xAE**
4	8-bit	Response type	The type of response to correlate with the preceding request: **0x00** - Login response **0x01** - Logout response **0x02** - Server Termination
5	64-bit	**64-bit source address**	The 64-bit IEEE address of the responding device.
13	8-bit	**Status**	Typical statuses: **0x00** - SRP operation was successful **0x01** - Invalid Password - SRP verification failed due to mismatched **M1** and **M2** values **0x02** - Session request was rejected as there are too many active sessions on the server already **0x03** - Session options or timeout are invalid **0x05** - Timed out waiting for the other node to respond **0x06** - Could not allocate memory needed for authentication

Offset	Size	Frame Field	Description
			0x07 - A request to terminate a session in progress has been made **0x08** - There is no password set on the server **0x09** - There was no initial response from the server **0x0A** - Data within the frame is not valid or formatted incorrectly Atypical statuses: **0x80** - Server received a packet that was intended for a client or vice-versa **0x81** - Received an SRP packet we were not expecting **0x82** - Offset for a split value (A/B) came out of order **0x83** - Unrecognized or invalid SRP frame type **0x84** - Authentication protocol version is not supported **0xFF** - An undefined error occurred
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Secure Session Login attempt

A client attempted to log into a Secure Session server.

The following Secure Session Response - 0xAE is emitted as a response:

```
7E 00 0B AE 00 00 13 A2 00 12 34 56 78 00 88
```

Frame type	Response type	64-bit source	Status
0x2E	0x00	0x0013A200 12345678	0x00
Response	*Login*		*success*

OTA firmware/file system upgrades

Overview

The XBee 3 DigiMesh RF Module supports two kinds of over-the-air upgrades:

- Firmware upgrades: upgrading the firmware or bootloader code on a device remotely.
- File System upgrades: placing or replacing the entire file system on a remote device.

An OTA upgrade is performed using two XBee3 RF modules: The **client module** is the module being upgraded, and the **server module** is connected to an external processor (the **OTA upgrade server**) and used to send the upgrade to the client. XCTU and Network Manager are capable of acting as an OTA upgrade server, and are the recommended method for distributing OTA upgrades. See Create an OTA upgrade server for more information on the OTA upgrade protocol.

Firmware over-the-air upgrades

A firmware OTA upgrade upgrades either just the application firmware or both the application firmware and the bootloader firmware on a device. OTA firmware upgrades must be to a different version, re-installing the same version as what is already installed is not supported.

Note Performing an OTA upgrade will erase any file system or bundled MicroPython code on the target device, even if the OTA upgrade does not complete.

File system over-the-air upgrades

A file system OTA upgrade uses the same protocol as a firmware OTA upgrade, but instead of changing the device firmware it installs a new image to the target module's file system. This method does not allow writing individual files, only copying an entire file system image at once. See OTA file system upgrades for more information on creating and sending file system images.

Scheduled upgrades

When a client has finished downloading the data for an OTA upgrade, it sends a request to the server asking when to apply the upgrade. The server can instruct the client to upgrade immediately, to wait a specified amount of time before upgrading, or to wait for a further command from the server to upgrade. If instructed to wait, the device will keep the downloaded upgrade for the specified time and then apply it. If a client looses track of time—for example, due to power loss—it will attempt to re-send the request for an upgrade time to the server and resume waiting. If the device does not receive a response to this request after a number of attempts, it applies the upgrade immediately.

Note Sleeping devices do not count time towards the upgrade while asleep. The delay for a scheduled upgrade on a sleeping end device should be calculated only considering the time that device will be awake.

Different OTA upgrade server tools have varying levels of support for scheduled upgrades. See the documentation for the OTA upgrade server you are using, or see Create an OTA upgrade server for information on how to implement scheduled upgrades on a server.

Create an OTA upgrade server

ZCL firmware upgrade cluster specification

The process, format, and commands used for OTA firmware upgrades are based on the ZCL OTA Upgrade cluster from the ZCL specification. The specification used is in Zigbee document 07-5123-06. Chapter 2 describes the general format of ZCL commands and chapter 11 describes the OTA upgrade cluster in detail. The specification contains a complete description of the OTA upgrade process, and you should reference it when creating an OTA upgrade server. This guide focuses on differences and examples specific to the XBee 3 DigiMesh RF Module. Where relevant, we refer to the ZCL specification document by section, for example (ZCL Spec §11.2.1).

Differences from the ZCL specification

The OTA upgrade process differs from what is described in the ZCL specification in the following ways:

- Setting/querying OTA cluster attributes and parameters (ZCL Spec §11.10, §11.11) is not supported.
- The WAIT_FOR_DATA status in an Image Block Response Command (ZCL Spec §11.13.8) is not supported.
- Devices will not automatically discover an OTA upgrade server upon joining a network (ZCL Spec §11.8). To specify an OTA server set US (OTA Upgrade Server), or leave it at its default value to accept OTA upgrades from any server.
- Clients do not automatically query the server for an available upgrade. The only way to start an OTA upgrade is by sending an Image Notify command from the server.

OTA files

Use an OTA file to perform an OTA upgrade. The OTA file format consists of an OTA header describing what is present in the file followed by one or more sub-elements containing the upgrade data. The OTA file format is described in the ZCL Spec §11.4.

The OTA file is included alongside other firmware files in each release. The file with the .ota extension contains the application firmware update, and the file with the .otb extension contains updates for both the firmware and the bootloader. The recommended bootloader version is listed in each firmware release's XML file—if the target device has an older version, we strongly recommend that you perform the OTA update using the .otb file. Updating a device with the same or newer bootloader version as the recommended version will not change the bootloader, but will update the application.

OTA header

The OTA header contains information about the upgrade data contained in the file. An OTA server needs to parse this file in order to get information that will be requested by a file. The OTA header format is (ZCL Spec §11.4.2):

Offset	Length	Name	Description
0	4	OTA upgrade file identifier	Unique identifier for an OTA file - will always be 0x0BEEF11E.
4	2	OTA header	Version for the OTA header format - The OTA header version

Offset	Length	Name	Description
		version	supported by XBee 3 firmwares is 0x0100.
6	2	OTA header length	The length in bytes of this OTA header.
8	2	OTA header field control	Indicates what optional fields are present.
10	2	Manufacturer code	The manufacturer code for the image.
12	2	Image type	One of two values: ■ 0x0000 for a firmware upgrade ■ 0x0100 for a file system upgrade
14	4	File version	Contains the version information for this upgrade. See File version definition for more information on how to interpret this field. **Note** It is important to parse this value from the OTA file itself instead of inferring it from the file name, as the software compatibility number is not included elsewhere.
18	2	Zigbee stack version	This field is not used for and can be ignored.
20	32	OTA header string	A human-readable string to identify the OTA file.
52	4	Total image size	The total size of the OTA file, including the OTA header. **Note** This field contains incorrect information in most older firmware files and should not be used in the update process. The total size of the file should be determined using an external method.

Note All fields—except for the OTA header string—are in little endian byte order. Optional fields may be present at the end of the OTA header, they have been omitted here as they are not used in the XBee 3 upgrade process.

File version definition

The file version is a 32-bit integer—sent in little-endian byte order—containing information on a firmware version. It is divided into two fields:

- The most significant byte corresponds to the compatibility number field in the firmware's XML file—see %C (Hardware/Software Compatibility)—for a description of the compatibility number's effect on loading firmware.
- The remaining three bytes indicate the firmware version as reported by **VR**.

For example, a file version of **0x0100100A** indicates that the software compatibility number is **1** and the version number is **100A**. **0x0200300B** indicates that the software compatibility number is **2** and the version is **300B**.

Sub-elements

All data after the OTA header is organized into sub-elements. Most OTA files will contain a single sub-element: the upgrade image. Sub-elements are arranged as tag-length-value triplets, as shown in the table below.

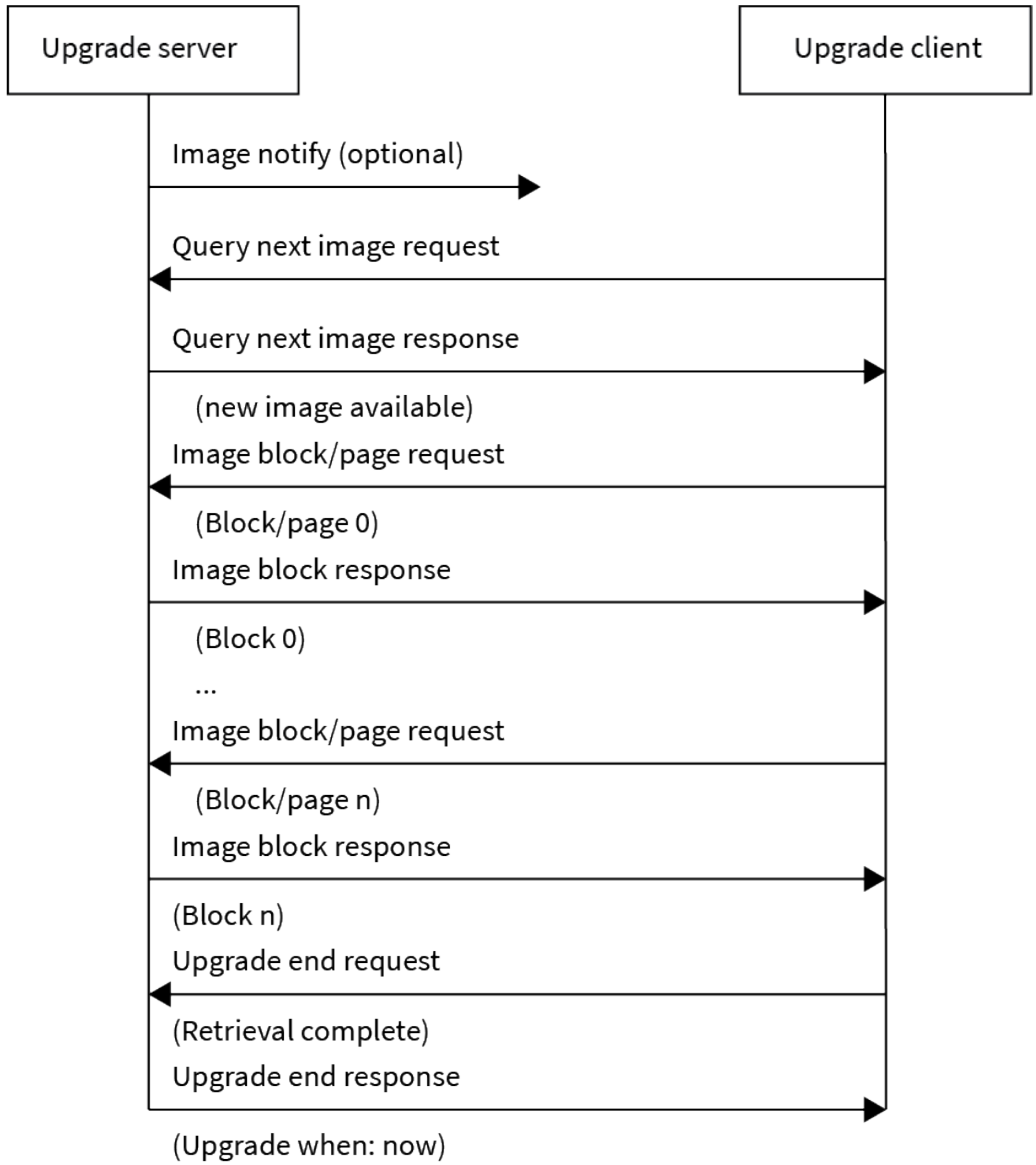
Offset	Length	Field name	Description
0	2	Sub-element tag	The tag for the sub-element, in little-endian format. This is usually 0x0000 for 'upgrade image'—this is the case for both firmware upgrades and file system upgrades.
2	4	Sub-element length	The length of the sub-element data (n) in little-endian format.
6	n	Sub-element data	The data to be transferred. This is either the contents of a .gbl firmware image or a signed file system image.

OTA upgrade process

The OTA upgrade process is performed by sending OTA commands between the client and server. OTA commands are sent as explicitly addressed packets, as described in OTA commands.

To initiate an OTA upgrade, the upgrade server sends an Image Notify Command, either to a single device or as a broadcast. After that initial transmission, the OTA process is driven by the client—or clients, if the Image Notify command is sent as a broadcast and accepted by multiple clients. The client sends requests to the server to request the image information, download it, and request when to upgrade. If the client does not receive a response from the server, it retries its request a few times before aborting the upgrade. The requests sent by the client are designed so that the server does not have to store any state related to a client's upgrade in progress—it only needs to send the image notify and respond to requests as they come in. The server can still observe these requests to track the state of an upgrade if desired, however—for example, to report download progress.

The following diagram shows the sequence of transmissions for an OTA upgrade:



OTA commands

All OTA commands are sent as explicitly addressed packets with the following address information:

- **Source/destination endpoint:** 0xE8
- **Cluster ID:** 0x0019
- **Profile ID:** 0xC105

The first three payload bytes of the command indicate what the command is and the structure of the remaining data in the command. All integer values in OTA commands are represented using little-endian byte order.

Image Notify command

(see ZCL Spec §11.13.3)

The Image Notify command is sent by the server to alert clients that an upgrade is available and prompt them to begin the upgrade. This command can be sent either as a broadcast or as a unicast:

- If sent as a unicast, the client will respond with a Query Next Image Request if the Image Notify contains valid information, and with a default response otherwise.
- If sent as a broadcast, all receiving clients will examine any optional fields included and respond only if the information indicates an image compatible with that device. On large networks, the query jitter parameter can be used to make only a percentage of those receiving the command respond at a time.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	When sending this command, value to set depends on whether the command will be sent as a broadcast or a unicast: ■ if sending a unicast: set this field to 0x09 (server-to-client command). ■ if sending a broadcast: set this field to 0x19 (server-to-client command, Default Response disabled).
1	1	Sequence number	Any sequence number can be used for the Image Notify
2	1	Command ID	0x00 for Image Notify
3	1	**Payload type**	Indicates which fields are present: 0: No optional fields (Query Jitter only) 1: Query Jitter, Manufacturer Code 2: Query Jitter, Manufacturer Code, Image Type 3: Query Jitter, Manufacturer Code, Image Type, File Version
4	1	**Query jitter**	A number, 0-100, must be set to 100 for a unicast. If less than 100 for a broadcast, then each receiving device will generate a random number and only respond to this command if that generated number is less than the query jitter.

Offset	Length	Field Name	Description
5	2	**Manufacturer code**	Optional. The Manufacturer code for the available image, parsed from the OTA file header.
7	2	**Image type**	Optional. The image type of the available image, parsed from the OTA file header.
9	4	**New file version**	Optional. The version parsed from the available image's OTA file header.

Example

To send this command from a server device, use the following Explicit Addressing Command Request - 0x11:

```
7E 00 21 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 09 01 00 03 64 1E 10 00 00 0A 20 00 01 18
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame control	Sequence number	Command ID	Payload type	Query jitter	Manufacturer code	Image type	New file version
Data	09	01	00	03	64	1E 10	00 00	0A 20 00 01
Value	0x09	0x01	0x00	0x03	0x64 (100)	0x101E	0x0000	0x0100200A
Description			Image Notify	All fields present	Client will always respond	Digi's manufacturer code	Firmware upgrade	Must match value in the OTA file header. 0x01: Software compatibility number 0x00200A: Application version

Additional error cases

If a client receives a unicast Image Notify command that includes any optional fields—Manufacturer ID, Image Type, New File Version—and those fields do not match what the client is expecting, it will send a default response to the server. See Default Response command for more information on possible error cases.

Query Next Image Request command

(See ZCL Spec §11.13.4)

The Query Next Image Request command is sent by the client to ask for information on any available OTA Upgrade. It is sent in response to an Image Notify from the server.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Will be set to 0x01, indicating a client to server command.
1	1	Sequence number	Sequence number chosen by the client.
2	1	Command ID	0x01 for Query Next Image Request.
3	1	**Field control**	Indicates which optional fields are present.
4	2	**Manufacturer code**	Manufacturer code of the client.
6	2	**Image type**	Image type that the client is requesting: ■ 0x0000 for a firmware upgrade ■ 0x0100 for a file system upgrade
8	4	**Current file version**	Firmware version that is currently running on the client. See File version definition for more information on how to interpret this field. **Note** The compatibility number reported in the current file version field refers to the installed firmware's compatibility number, which may be different from the **%C** value of the device.
12	2	**Hardware version**	Optional. Hardware version of the client.

Example

This is an example Explicit Rx Indicator (0x91) frame containing a Query Next Image Request that could be received by a server:

```
7E 00 1E 91 00 13 A2 00 55 66 77 88 FF FE E8 E8 00 19 C1 05 01 01 02 01 00 1E
10 00 00 06 20 00 01 F9
```

The payload portion of the API frame (starting at offset 21) is shown below:

	Frame control	Sequence number	Command ID	Field control	Manufacturer code	Image type	Current version
Data	01	02	01	00	1E 10	00 00	06 20 00 01

	Frame control	Sequence number	Command ID	Field control	Manufacturer code	Image type	Current version
Value	0x01	0x02	0x01	0x00	0x101E	0x0000	0x01002006
Description			Query Next Image Request	HW version not present	Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x002006: Application version

Query Next Image Response command

(See ZCL Spec §11.13.5)

The Query Next Image Response command should be sent by the server when it receives a Query Next Image request.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Should be set to 0x19, indicating a server-to-client command.
1	1	Sequence number	Must match the sequence number of the request that prompted this response.
2	1	Command ID	0x02 for Query Next Image Response.
3	1	**Status**	One of three values: ■ 0x00 (SUCCESS): An image is available ■ 0x98 (NO_IMAGE_AVAILABLE): No upgrade image is available ■ 0x7E (NOT_AUTHORIZED): This server isn't authorized to perform an upgrade Remaining fields are only included if this field contains 0x00 (SUCCESS).
4	2	**Manufacturer code**	The Manufacturer code for the available image, parsed from the OTA file header. Must match the manufacturing code from the Query Next Image request that prompted this response.
6	2	**Image type**	The Image for the available image, parsed from the OTA file header. Must match the manufacturing code from the Query Next Image request that prompted this response.
8	4	**File version**	The version parsed from the available image's OTA file header.
12	4	**Image size**	The size in bytes of the image that will be sent over the air. This should be the size of the OTA file. **Note** This field is handled differently if the client has a firmware version older than 300A. See Does the download include the OTA header?.

Example

An OTA server could respond to the Query Next Image Request example in the previous section using the following Explicit Addressing Command Request - 0x11:

```
7E 00 24 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 19 02 02
00 1E 10 00 00 0A 20 00 01 3A 90 05 00 9D
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame Control	Sequence Number	Command ID	Status	Manufacturer Code	Image Type	File Version	Image Size
Data	19	02	02	00	1E 10	00 00	0A 20 00 01	3A 90 05 00
Value	0x19	0x02	0x02	0x00 (SUCCESS)	0x101E	0x0000	0x0100200A	0x0005903A
Description					Digi's manufacturer code	Firmware upgrade	Must match value in the OTA file header. 0x01: Software compatibility number 0x00200A: Application version	

This indicates that the server has version 0x0100200A available for the client to upgrade to, and that the file's size is 0x0005903A (364,6042) bytes.

Image Block Request command

(See ZCL Spec §11.13.6)

The client sends Image Block Request commands to the server to download the upgrade image data. The client will send requests until it has downloaded the entire image, as determined by the image size given in the Query Next Image Response from the server.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Will be set to 0x01, indicating a client to server command.
1	1	Sequence number	Sequence number chosen by the client.
2	1	Command ID	0x03 for Image Block Request.
3	1	**Field control**	Indicates which optional fields are present. No optional fields are currently used by the XBee 3 DigiMesh RF Module.
4	2	**Manufacturer code**	The manufacturer code of the image being downloaded.
6	2	**Image type**	The image type of the image being downloaded.
8	4	**File version**	The version number of the file being downloaded.
12	4	**File offset**	The offset at which to begin the data, from the start of the OTA file. **Note** This field is handled differently if the client has a firmware version older than 300A. See Does the download include the OTA header?
13	1	**Maximum data size**	The maximum number of bytes of image data the server may include in its response.

Note Optional fields have been omitted here as they are not used by the XBee 3 DigiMesh RF Module.

Example

This is an example Explicit Receive Indicator - 0x91 containing an Image Block Request that could be received by a server:

```
7E 00 25 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 01 12 03
00 1E 10 00 00 0A 20 00 01 34 12 00 00 63 CA
```

The payload portion of the API frame (starting at offset 21) is shown below:

	Frame control	Sequence number	Command ID	Field control	Manufacturer code	Image type	Current version	File offset	Maximum data size
Data	01	12	03	00	1E 10	00 00	0A 20 00 01	34 12 00 00	63
Value	0x01	0x12	0x01	0x00	0x101E	0x0000	0x0100200A	0x00001234	0x63
Description			Image Block Request	No optional fields present	Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x00200A: Application version		

The client is requesting up to 0x63 bytes of data, starting from offset 0x1234.

Image Block Response command

(See ZCL Spec §11.13.8)

The Image Block Response is generated by the OTA server to send the data asked for in an Image Block Request.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Should be set to 0x19 indicating a server-to-client command.
1	1	Sequence number	Must match the sequence number of the request that prompted this response.
2	1	Command ID	0x05 for Image Block Response.
3	1	**Status**	This field has one of two values, and determines the structure of the remaining fields: ■ 0x00 (SUCCESS): Image data is available. The remaining fields must be included. ■ 0x95 (ABORT): Instructs the client to abort the download. The remaining fields must not be included. **Note** The 0x97 (WAIT_FOR_DATA) status (see ZCL Spec §11.13.8.1) is not supported.
4	2	**Manufacturer code**	The Manufacturer code for the available image, parsed from the OTA file header. Must match the manufacturing code from the request that prompted this response.
6	2	**Image type**	The Image for the available image, parsed from the OTA file header. Must match the manufacturing code from the request that prompted this response.
8	4	**File version**	The version parsed from the available image's OTA file header. Must match the version number from the request that prompted this response.
12	4	**File offset**	The offset into the OTA file where the data begins. Must match the offset from the request that prompted this response. **Note** This field is handled differently if the client has a firmware version older than 300A. See Does the download include the OTA header?
16	1	**Data size**	The number of bytes of data included in this block. This can be any number less than or equal to the maximum data size value in the request that prompted this response.

Offset	Length	Field Name	Description
17	n	**Image data**	Image data starting from the given offset. The length of this field is determined by the value in the preceding field (Data Size).

Example

An OTA server could respond to the Image Block Request example in the previous section using the following Explicit Addressing Command Request - 0x11:

```
7E 00 28 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 19 12 05
00 1E 10 00 00 0A 20 00 01 34 12 00 00 03 69 6D 67 D3
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame control	Sequence number	Command ID	Status	Manufacturer code	Image type	File version	File offset	Data size	Image data
Data	19	12	05	00	1E 10	00 00	0A 20 00 01	34 12 00 00	03	69 6d 67
Value	0x19	0x12	0x05	0x00 (SUCCESS)	0x101E	0x0000	0x 0100 200A	0x00001234	0x03	69 6d 67
Description			Image Block Response		Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x00200A: Application version			

This response contains three bytes of data starting at offset 0x1234. The data size value in this example is very small—three bytes—for simplicity; since any size less than or equal to the client's requested maximum is allowed this is a valid frame, but smaller image blocks will increase the time the OTA upgrade takes.

Upgrade End Request command

(See ZCL Spec §11.13.9)

The Upgrade End Request command is sent by the client when it finishes a download, whether successfully or not.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Will be set to 0x01, indicating a client to server command.
1	1	Sequence number	Sequence number chosen by the client.
2	1	Command ID	0x06 for Upgrade End Request.
3	1	**Status**	One of four values indicating the status of the download. ■ 0x00 (SUCCESS): The client successfully downloaded and verified the image. ■ 0x96 (INVALID_IMAGE): The client aborted the download because the downloaded image was invalid or corrupted. ■ 0x95 (ABORT): The client aborted the download for another reason. ■ 0x99 (REQUIRE_MORE_IMAGE): The download completed, but additional files are needed for the upgrade. This status is not used by the XBee 3 DigiMesh RF Module. The value of this field determines what response the server should send. If the status is 0x00 (SUCCESS), the server should respond with an Upgrade End Response command. Otherwise, the server should respond with a Default Response command with the SUCCESS status.
4	2	**Manufacturer code**	The manufacturer code of the image being downloaded.
6	2	**Image type**	The image type of the image being downloaded.
8	4	**File version**	The version of the image being downloaded

Exampe

This is an example Explicit Receive Indicator - 0x91 containing an Upgrade End Request that could be received by a server:

```
7E 00 1E 91 00 13 A2 00 55 66 77 88 FF FE E8 E8 00 19 C1 05 01 01 95 06 00 1E
10 00 00 0A 20 00 01 5D
```

The payload portion of the API frame (starting at offset 21) is shown below:

	Frame control	Sequence number	Command ID	Status	Manufacturer code	Image type	File version
Data	01	95	06	00	1E 10	00 00	0A 20 00 01
Value	0x01	0x95	0x06	0x00 (SUCCESS)	0x101E	0x0000	0x0100200A
Description			Upgrade End Request		Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x00200A: Application version

The client has completed the download of version 0x0100200A. The server should respond with an Upgrade End Response command.

Upgrade End Response command

(See ZCL Spec §11.13.9.6)

The Upgrade End Response command is sent by the server when it receives an Upgrade End Request with the SUCCESS status. This command instructs the device to perform the upgrade, and can be used to schedule an upgrade for a later time. An Upgrade End Response can also be sent without a request from a client if the client is waiting for an upgrade—scheduled by a previous Upgrade End Response—to change the time to wait for that upgrade.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Should be set to 0x19 indicating a server-to-client command.
1	1	Sequence number	If this command is sent in response to an Upgrade End request, the sequence number should match the one from that request.
2	1	Command ID	0x07 for Upgrade End Response.
3	2	**Manufacturer code**	The Manufacturer code for the available image, parsed from the OTA file header. Must match the manufacturer code from the request that prompted this response.
5	2	**Image type**	The Image for the available image, parsed from the OTA file header. Must match the image type from the request that prompted this response.
7	4	**File version**	The version parsed from the available image's OTA file header. Must match the version number from the request that prompted this response.
11	4	**Current time**	The current time, used for scheduled upgrades. See Schedule an upgrade for more information.
15	4	**Upgrade time**	The scheduled upgrade time, used for scheduled upgrades. See Schedule an upgrade for more information.

If the upgrade should be performed immediately and not scheduled for a later time, the Current Time and Upgrade Time fields should be set to the same value less than 0xFFFFFFFF.

Example

An OTA server could respond to the Image Block Request example in the previous section using the following Explicit Addressing Command Request - 0x11:

```
7E 00 27 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 19 95 07
1E 10 00 00 0A 20 00 01 00 00 00 00 00 00 00 00 D4
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame control	Sequence number	Command ID	Manufacturer code	Image type	File version	Current time	Upgrade time
Data	19	95	07	1E 10	00 00	0A 20 00 01	00 00 00	00 00 00 00
Value	0x19	0x95	0x07	0x101E	0x0000	0x0100200A	0x00000000	0x00000000
Description			Upgrade End Response	Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x00200A: Application version		

With the current time and upgrade time both set to 0, the device will reboot and install the upgrade immediately.

Default Response command

(See ZCL Spec §2.5.12)

A Default Response command is sent when a response is needed but there is no other command frame suited to the response.

During the OTA Upgrade process, the client will send a default response with an error status if it receives an invalid command from the server. The only time the server needs to send a default response is when it receives an Upgrade End Request with an error status; the server responds with a default response with status 0x00 (SUCCESS) status to indicate that the request was received.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	If command is sent by the client: 0x10 If command is sent by the server: 0x18
1	1	Sequence number	Must match the sequence number of the command that prompted this Default Response.
2	1	Command ID	0x0B for Default Response.
3	1	**(Source) command identifier**	The command ID of the command that prompted this Default Response.
4	1	**Status code**	A status code indicating success or an error. A full list of status codes, see ZCL Spec §2.6.3.

Error messages sent by the client

The client will send a default response to the server when an error occurs. The significance of the status code in this message depends on what server command prompted the default response. The **Handling Error Cases** section of each command's section in the ZCL specification contains detailed information on what errors a command can produce. Some errors that can be sent by the client are listed below:

<table>
<tr><th>Source Command Identifier</th><th>Status</th><th>Description</th></tr>
<tr><td rowspan="3">0x00 (Image Notify)</td><td>0x80 (MALFORMED_ COMMAND)</td><td>Either one of the errors form ZCL Spec §11.13.3.5.1, or manufacturer code or image type is not valid.</td></tr>
<tr><td>0x70 (REQUEST_ DENIED)</td><td>OTA Upgrades have been disabled on this device.</td></tr>
<tr><td>0x8A (DUPLICATE_ EXISTS)</td><td>The new version is not valid:
■ For firmware upgrades, the new firmware version must be different than what is installed on the device. Upgrades to the same version are not supported.</td></tr>
</table>

Source Command Identifier	Status	Description
		■ For file system upgrades, the version indicates what firmware version the image supports. It must match the currently installed firmware. Make sure the firmware version in the Image Notify is being parsed from the OTA header in the upgrade image.
	0x85 (INVALID_FIELD)	Firmware is incompatible with the client's **%C (Hardware Compatibility)** value.
0x02 (Query Next Image Response)	0x80 (MALFORMED_COMMAND)	The format of the command is invalid (see ZCL Spec §11.13.5.5).
	0x89 (INSUFFICIENT_SPACE)	The image is too large for the client to store.
0x05 (Image Block Response)	0x80 (MALFORMED_COMMAND)	The format of the command is invalid (See ZCL Spec §11.13.8.5).
0x07 (Upgrade End Response)	0x80 (MALFORMED_COMMAND)	The format of the command is invalid (See ZCL Spec §11.13.9.9).

Example

After unicasting an Image Notify command to a client, the server may receive the following Explicit Receive Indicator - 0x91 frame containing a Default Response:

```
7E 00 17 91 00 13 A2 00 55 66 77 88 FF FE E8 E8 00 19 C1 05 01 10 0C 0B 00 8A
A1
```

The payload portion of the API frame (starting at offset 21) is shown below:

	Frame control	Sequence number	Command ID	Source command identifier	Status
Data	10	0C	0B	00	8A
Value	0x10	0x0C	0x0C	0x00	0x8A (DUPLICATE_EXISTS)
Description			Default Response	Image Notify	

The source command identifier field indicates that the error is in response to an image notify, and the sequence number will match that of the Image Notify command sent by the server. According to the table above, a DUPLICATE_EXISTS status for an Image Notify means that the firmware version is invalid—the device is already running the firmware version that the server is trying to send.

When the server needs to send a default response, it can do so using an Explicit Addressing Command Request - 0x11. For example, to send a Default Response with a SUCCESS status in response to an Upgrade End Request:

```
7E 00 19 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 18 41 0B
06 00 78
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame control	Sequence number	Command ID	Source command identifier	Status
Data	18	41	0B	06	00
Value	0x18	0x41	0x0B	0x06	0x00 (SUCCESS)
Description			Default Response	Upgrade End Response	

Handling unrecognized commands

If the server receives a command with an unrecognized command ID, it should respond with a default response with status 0x81 (UNSUP_CLUSTER_COMMAND).

Schedule an upgrade

The current time and upgrade time fields of the Upgrade End Response command can be used to schedule an upgrade for some time in the future. The time can for the upgrade can be scheduled in several ways:

Current time value	Upgrade time value	Effect
0x00000000-0xFFFFFFFE	Equal to current time	The device will upgrade immediately.
0x00000000	0x00000001-0xFFFFFFFE	Delayed upgrade: the device will upgrade after the number of seconds indicated by the upgrade time value.
0x00000001-0xFFFFFFFE (Current time in seconds since midnight Jan 1, 2000)	Any value greater than current time and less than 0xFFFFFFFF (Intended upgrade time in seconds since midnight Jan 1, 2000)	Scheduled upgrade: the device will determine how long to wait by subtracting current time from upgrade time, and wait that long before upgrading.
Any	0xFFFFFFFF	Prompted upgrade: The device will not upgrade, and will wait indefinitely to receive another Upgrade End Response with the server. The second upgrade end response can schedule an upgrade with any of the above methods.

Note When performing a scheduled upgrade, we recommend that the OTA upgrade server continue to monitor for and respond to OTA commands until after the time the upgrade is meant to be applied. If the client loses power while waiting to apply a scheduled upgrade, it will send another Upgrade End Request to the server when it regains power in an attempt to resume the schedule. If the client does not receive a response from the server after a few tries, it applies the upgrade without confirmation from the server.

Scheduled upgrades on sleeping devices

To schedule an upgrade, an XBee 3 DigiMesh RF Module makes use of internal software timers, which only count time while the device is awake. So a sleeping device takes significantly longer to apply the scheduled upgrade than a non-sleeping device. Consider this limitation when scheduling an upgrade on a sleeping device.

Formula for estimating when a sleeping device will apply an upgrade

upgrade_delay = number of seconds the upgrade was scheduled for (**upgradeTime**- **currentTime** fields in the Upgrade End Response frame) sleep_time = amount of time the device is estimated to be asleep (**SP** for an asynchronous sleeping device)

wake_time = amount of time the device is estimated to be awake (**ST** for an asynchronous sleeping device)

total_time = sleep_time + wake_time

expected_upgrade_delay = upgrade_delay * (total_time / wake_time)

Asynchronous cyclic sleep scheduled upgrades

A device that is configured for asynchronous cyclic sleep will only be awake for a few milliseconds at a time, therefore we do not recommend that you schedule an upgrade for a sleeping node with this configuration. However, if the device is configured to always stay awake for **ST** time then the scheduled upgrade can be estimated by using the above formula—where wake_time = **ST** and sleep_time = **SP**. You can configure a device to always stay awake for **ST** by setting **SO** bit 8 to one—for example, **SO** = 0x80).

Synchronous cyclic sleep scheduled upgrades

A scheduled upgrade on a sleeping device in a synchronous sleeping XBee 3 DigiMesh network can be estimated using the above formula for estimating when the device will apply the upgrade. Use **OS**for sleep_time and **OW**for wake_time.

Pin sleep scheduled upgrades

Since the device only counts time while it is awake, scheduling an upgrade on a pin sleeping device may be unpredictable. However, if a pin sleeping device has predictable sleep patterns it is possible to estimate when a scheduled upgrade will be applied. The sleep estimate formula can be applied to a pin sleeping device to estimate when it will apply the upgrade.

Aggressively sleeping devices

If a device is asynchronously sleeping, and keeping it awake for all of **ST** time is undesired, then we recommend performing a scheduled upgrade in the following manner:

1. Configure the sleeping node for indirect messaging:
   a. Configure the sleeping device with the following parameters:
      - **CE** = **4** (indirect message polling)
      - **DH, DL** should be set to match **SH**, **SL** of the OTA server device

   b. Make sure that **ST** and **SP** of the sleeping device and OTA server radio match.

   c. Set all of the transmit option fields of the API frames sent to the OTA server device to **0x40**.
2. Download the firmware/file system image to the sleeping device as described in this section.
   a. When sending the Upgrade End Response frame set the **upgradeTime** to **0xFFFFFFFF**—instructing the sleeping device to wait for another upgrade end request before applying the upgrade.
3. Wait for the desired amount of time to pass.
4. When the time to have the sleeping device apply its upgrade has arrived, send a second Upgrade End Response to the sleeping device with the **currentTime** and **upgradeTime** fields both set to **0x0000**. This causes the sleeping device to apply the upgrade immediately.

Considerations for older firmware versions

Some changes need to be made to this OTA upgrade process for some previous versions of the software.

All versions older than 300A

- When the firmware is sent over the air it must be sent without including the OTA header and sub-element tags. See Does the download include the OTA header?
- These older versions will not retry requests; if a packet from the server is dropped, you may need to restart the upgrade.

Does the download include the OTA header?

Most OTA files consist of an OTA header, a sub-element tag, and a single sub-element: The upgrade image. For firmware versions 300A and newer, the entire OTA file is sent to the client during an OTA Upgrade. However, for versions older than 300A, only the contents of the file's single sub-element should be sent—not the OTA header or the sub-element tag. This affects several fields in the upgrade process.

When dealing with these two methods it is useful to know the **image offset** of the OTA file—that is, the offset at which the upgrade image data actually begins. This can be calculated by taking the size of the OTA header—which can be parsed from near the beginning of the OTA file—and adding six bytes for the sub-element header: two bytes for the tag, four bytes for the length.

Command	Field	Value when sending without header (pre-300A)	Value when sending with header (300A and later)	Notes
Query Next Image Response	Image size	The size of the upgrade image parsed from the first sub-element tag's length value, or the total size of the OTA file minus the image offset.	The total size of the OTA file.	In either case, this is the total number of bytes that the client needs to download. This value should never be determined by reading the **Total Image Size** field from the OTA header, as that field contains incorrect information on most older firmware files.
Image Block	File offset	This refers to the offset from the start of the upgrade image data—add the image offset to this value to get the offset into the OTA file.	This refers to the offset into the OTA file.	

Note For compatibility with older OTA upgrade servers, newer firmware versions support both methods for a firmware upgrade. File system upgrades only support the method corresponding to the installed firmware version, as described above. We recommend using the newer method where possible to ensure compatibility with future releases.

OTA file system upgrades

After a FOTA update, all file system data and bundled MicroPython code is erased. To continue running code, a new file system needs to be sent to the device after the firmware update is complete. This section contains information on how to update the file system of remote devices over the air.

OTA file system update process

Since OTA file system updates are signed, remote devices must be configured so that they can validate incoming updates. To set up a network for OTA file system updates:

1. Generate a public/private Elliptic Curve Digital Signature Algorithm (ECDSA) signing key pair.
2. Using the generated public key, set FK (File System Public Key) on all devices that will receive OTA file system updates.

Note You cannot set **FK** remotely. You must either set **FK** before the XBee 3 DigiMesh RF Module is deployed, or else serial access to the device is needed to set it.

To perform an OTA file system update:

1. On a local device, create a copy of the file system that you want to send over the air.
2. Create an OTA file system image, signed using the private key generated previously.
3. Perform an OTA update using the created OTA file.

Note The local device used to create the file system image must have the same firmware version installed as the target device or the file system will be rejected. Use VR (Firmware Version) to check the version number on both the staging and target devices.

You can perform all of these steps automatically through XCTU or manually using other tools.

OTA file system updates using XCTU

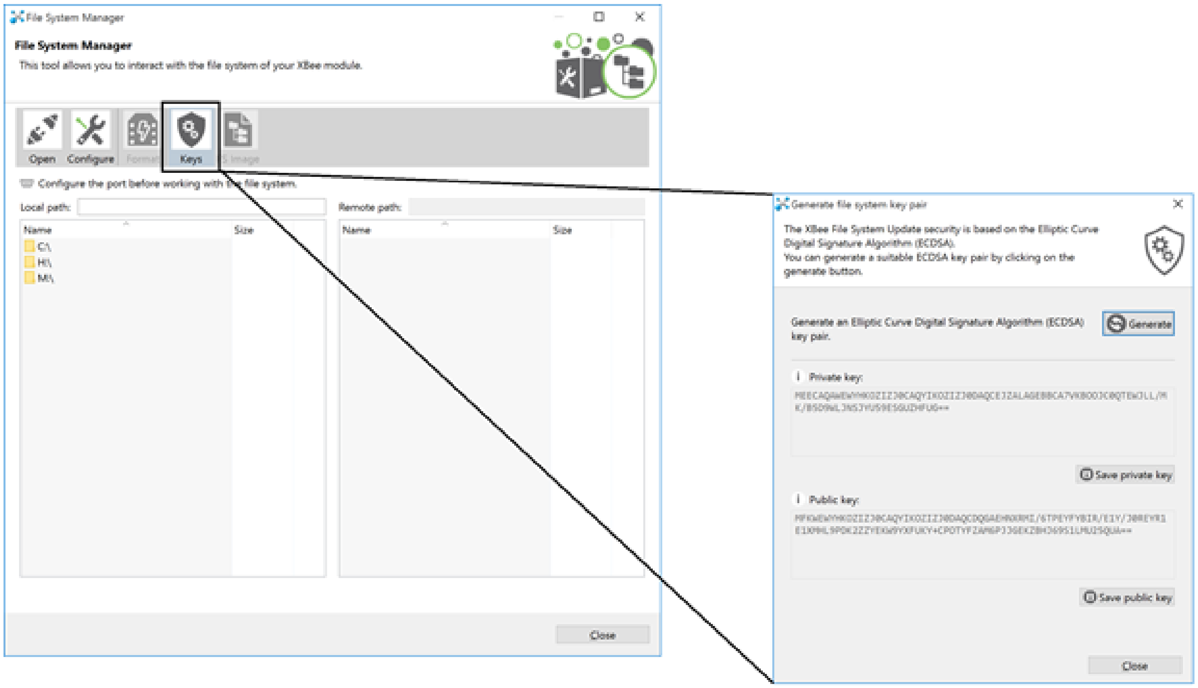
Use the following steps to perform a file system update OTA using XCTU:

1. Generate a public/private key pair
2. Set the public key on the XBee device
3. Create the OTA file system image
4. Perform the OTA file system update

Generate a public/private key pair

XCTU provides an ECDSA key pair generator that you can use to store a public/private key pair in .pem files. To access the **Generate file system key pair** dialog:

1. Open the **File System Manager** dialog box.
2. Click **Keys** as shown below.



3. Click **Generate** in the **Generate file system key pair** dialog.
4. Save both the keys in a safe location and close the dialog box.

Set the public key on the XBee device

1. Open the configuration view of the target device in XCTU and go to the **File System** category.
2. In the **File System Public Key** row, click **Configure**.

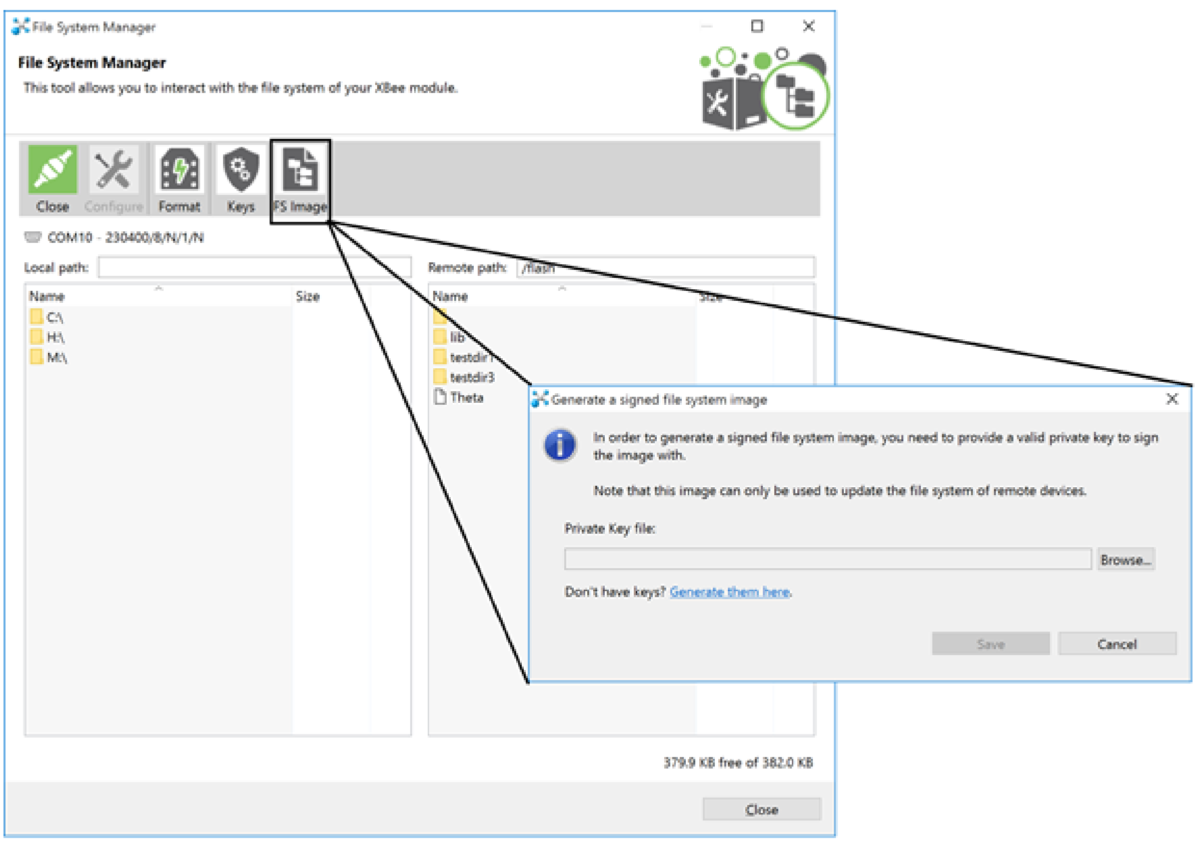
3. In the **Configure File System Public Key** dialog box, click **Browse** and choose the .pem file that you saved the public key into. Once this is done, the HEX value of the public key is visible under the **Public key** section on the dialog box as shown.
4. Click **OK** to ensure that the key gets written into the device.

Note This can be only be done locally. XBee firmware **DOES NOT** support remotely setting the file system public key at this time.

Create the OTA file system image

To create the OTA file system image:

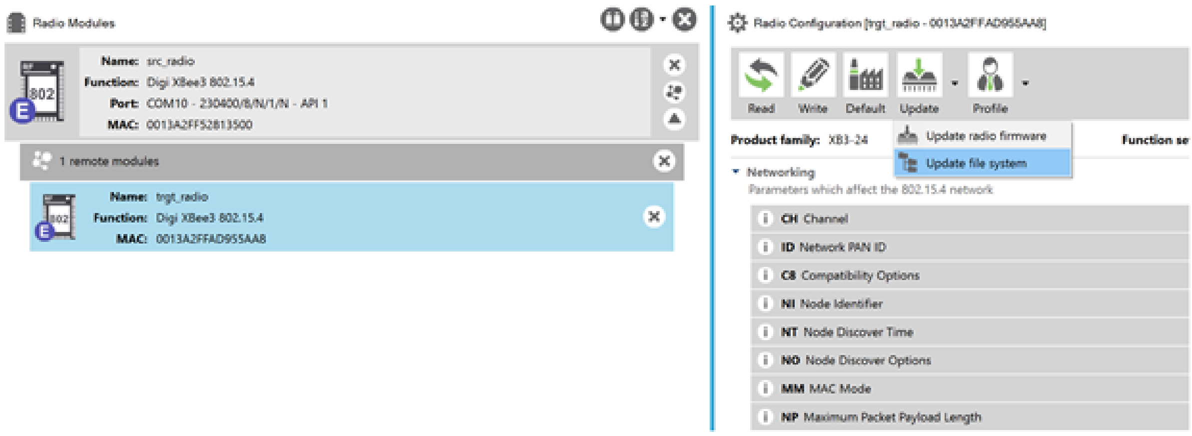
1. Open the **File System Manager** dialog box.
2. Open a connection on the device that you want to generate the OTA file system image from.
3. Click **FS Image**.
4. In the **Generate a signed file system image** window that displays, click **Browse** and choose the .pem file that the private key was stored in.
5. Once the path shows up on the **Private Key file** field, click **Save** to assign the .fs.ota an appropriate file name and location.
6. Save the file.



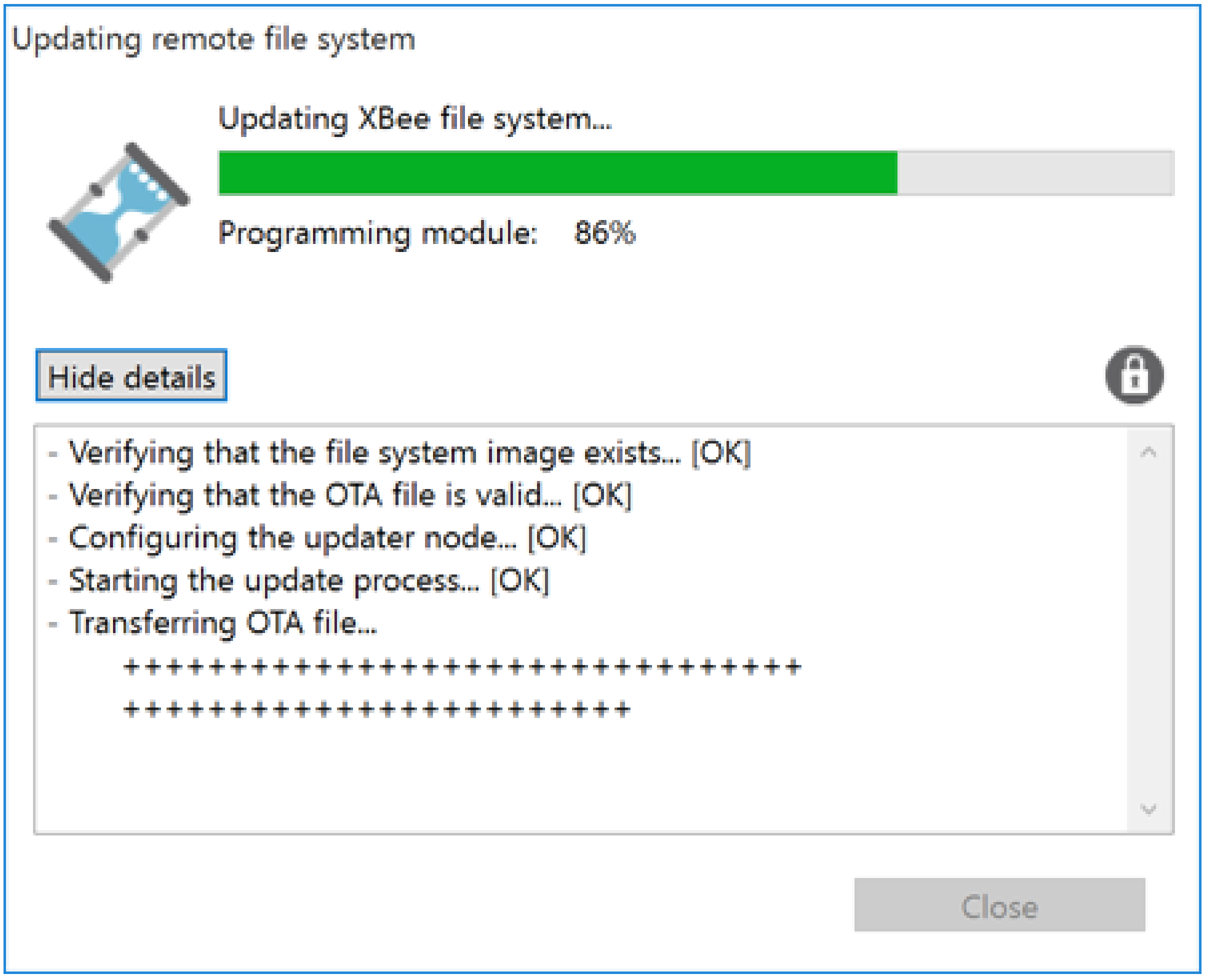
You will be prompted with a **File system image successfully saved** dialog box if the file was successfully generated.

Perform the OTA file system update

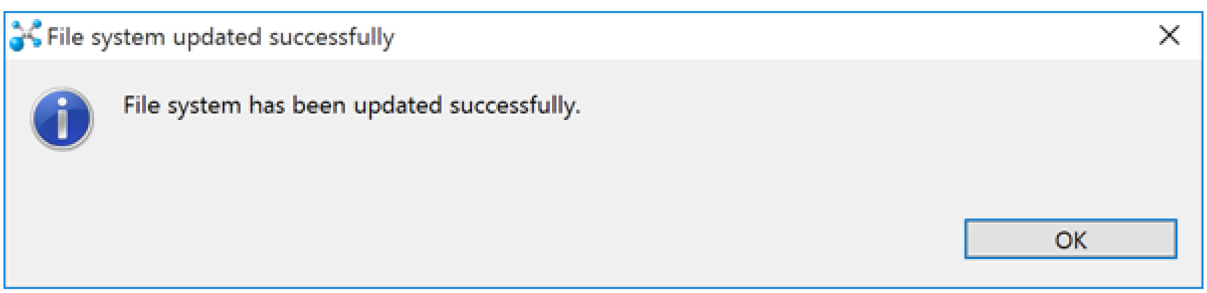
1. To add the target device, click **Discover radios in the same network** from the source device.
2. Enter Configuration mode on the remote device.
3. Click the down arrow next to the **Update** button and choose **Update File System**.



4. Choose the OTA file system image (.fs.ota) that the target node needs to be updated to.
5. Click **Open**.



Once the file system image is completely transferred and mounted on the remote device, XCTU informs you that the file system has been updated successfully.



OTA file system updates: OEM

Use the following steps to perform a file system update OTA using OEM tools:

1. Generate a public/private key pair
2. Set the public key on the XBee 3 device

3. Create the OTA file system image
4. Perform the OTA file system update

Generate a public/private key pair

Generate ECDSA signing keys using secp256r1 curve parameters (also known as prime256v1 or NIST P-256).

To generate a public/private key pair using OpenSSL, run the following command:

```
openssl ecparam -name prime256v1 -genkey -outform pem -out keypair.pem
```

To extract the private key from the key pair generated above:

```
openssl pkcs8 -topk8 -inform pem -in pair.pem -outform pem -nocrypt -out
private.pem
```

To extract the public key from the key pair generated above:

```
openssl ec -in keypair.pem -pubout -out public.pem
```

Set the public key on the XBee 3 device

The public keys generated by XCTU and OpenSSL are stored in *.pem files. These files need to be parsed to get the value to use when setting **FK**. To parse a public key file, run:

```
openssl asn1parse -in public.pem -dump
```

The command will produce something like the following output:

```
0:d=0  hl=2 l=  89 cons: SEQUENCE
  2:d=1  hl=2 l=  19 cons: SEQUENCE
  4:d=2  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
 13:d=2  hl=2 l=   8 prim: OBJECT            :prime256v1
 23:d=1  hl=2 l=  66 prim: BIT STRING
    0000 - 00 04 95 50 aa 55 b6 f5-5d 99 4d d8 15 d1 71 57   ...P.U..].M...qW
    0010 - 51 80 d5 14 ec 1f 6a 15-51 a2 c4 b8 0f 77 10 8a   Q.....j.Q....w..
    0020 - 33 a3 80 07 47 40 14 8b-5c a7 4c 78 02 fc 4d 82   3...G@..\.Lx..M.
    0030 - 90 4b 39 98 62 a1 1d 97-6e 78 fb 54 62 06 d2 41   .K9.b...nx.Tb..A
    0040 - c7 3b
```

The public key should be 65 bytes long - it is the BIT STRING value at the end, with the leading 00 omitted; in this case:

```
049550aa55b6f55d994dd815d171575180d514ec1f6a1551a2c4b80f77108a33a380074740148b5
ca74c7802fc4d82904b399862a11d976e78fb546206d241c73b
```

Create the OTA file system image

You can create a file system image outside of XCTU using any utility that can perform ECDSA signing. These instructions show how to do so using OpenSSL. To create an OTA file system image, use the following steps.

Create a staged file system

In order to create a usable file system image, first create a 'staged' copy of the file system you want to send on a local device.

Use the **FS** command or MicroPython to load all of the files that you want to send onto the local staging device.

Note The staging device must have the same firmware version installed as the target device or the file system will be rejected. Use the **VR** command to check the version number on both the staging and target devices.

Download the file system image

Run the command **ATFS GET /sys/xbfs.bin** to download an image of the file system from the staging device. The file is transferred using the YMODEM protocol. See File system for more information on downloading files using **FS GET**.

Pad the file system image

The file system image must be a multiple of 2048 bytes long before it is signed. Using hex editing software, add 0xFF bytes to the end of the downloaded image until size of the file is a multiple of 2048 (0x800 in hex).

Calculate the image signature

Once the image has been padded to a multiple of 2048 bytes, it is ready to be signed. The ECDSA signature should be calculated using SHA256 as the hash algorithm.

Assuming a public/private key pair has been generated as described in Generate a public/private key pair, that the private key is named private.pem, and that the padded image is named **xbfs.bin**; this can be done using OpenSSL with the following command:

```
openssl dgst -sha256 -sign private.pem -binary -out sig.bin xbfs.bin
```

sig.bin will contain the signature for the image.

Append the calculated signature to the image

The signature should be between 70 and 72 bytes, and it should be appended to the padded image.

Create the OTA file

Put the image into an OTA file that follows the format specified in ZigBee Document 095264r23. The file should consist of:

- An OTA header
- An upgrade image sub-element tag
- The padded, signed image data

The OTA file must begin with an OTA header. See The OTA header for information on the format of the header. The image type should be **0x0100** for a file system image upgrade.

The sub-element tag should come before the image data. The sub-element tag follows the format described in section **6.3.3** of ZigBee Document 095264r23. It consists of 6 bytes: the first 2 bytes are the tag id and should be set to **0x0000**. The next 4 bytes contain the length of the file system image in little-endian format.

Perform the OTA file system update

The process for performing an OTA file system update is the same as the process for performing a FOTA upgrade, as described in Over-the-air firmware/file system upgrade process for DigiMesh 2.4.

Note that the data that goes in the image blocks starts at the beginning of the image data, after the OTA header and sub-element tag.