Digi XBee® 3 Cellular LTE-M/NB-IoT

Smart Modem

User Guide

Revision history—90002258

Revision	Date	Description
AH	July 2023	Updated default value for K1 and K2 AT commands.
AG	June 2023	Updated CP (Carrier Profile). Updated PY (MicroPython Command) to include **PYR (Soft Reset)**. Updated SM (Sleep Mode).
AF	April 2023	Updated USB direct mode documentation.
AE	January 2023	Added Digi Remote Manager device requests section: Device Requests in Remote Manager.
AD	November 2022	Updated to include a link to the Digi MicroPython Programming Guide. See PY (MicroPython Command). Updated HM (Health Metrics). Updated Network connection issues. Updated DT (Cellular Network Time). Added PSM information. See Modem fails to go into PSM dormant state.

Trademarks and copyright

Disclaimers

Information in this document is subject to change without notice and does not represent a commitment on the part of Digi International. Digi provides this document "as is," without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

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 Cellular LTE-M/NB-IoT Global Smart Modem User Guide, 90002258 I) in the subject line of your email.

Contents

XBee connection examples

Get started with MicroPython

Get started with Bluetooth® Low Energy

Get started with Digi Remote Manager

Examples: IOT protocols with transparent mode

Update the firmware

Technical specifications

Hardware

Antenna recommendations

Design recommendations

Cellular connection process

Modes

Sleep modes

Power saving features and design recommendations

Serial communication

SPI operation

File system

Socket behavior

Extended Socket frames

Transport Layer Security (TLS)

AT commands

Operate in API mode

API frames

File system API frames

Regulatory firmware

Troubleshooting

Regulatory Information

Deprecated kit (PN XK3-C-A2-UT-U or XK3-C-N1-UT-E)

Digi XBee® 3 Cellular LTE-M/NB-IoT Global Smart Modem User Guide

The XBee Smart Modem provides OEMs with a simple way to integrate low-power cellular connectivity into their devices.

Features include:

- FCC certified and carrier end-device certified
- Excellent coverage and building penetration
- Manage and configure with XCTU and Digi Remote Manager®
- Available with Digi provided SIM cards and data plans
- Digital I/O support
- Analog input support
- API and Transparent mode
- Command mode
- Bypass to the raw cellular modem
- SMS: Some carriers do not support SMS on LTE-M and/or NB-IoT. Check with your carrier for details.
- TCP/UDP (up to six sockets)
- TLS (up to six sockets)
- Incoming connections
- MicroPython
  - On-module programmability to add local intelligence
  - Many examples in the *Digi MicroPython Programming Guide*
  - AT commands for managing run-time behavior
- Low power modes
- LTE power save mode (PSM)
- Deep sleep mode
- Pin sleep support
- Cyclic sleep support
- Airplane mode support
- Digi TrustFence secure boot
- Multi-network capability (Verizon, AT&T)

Applicable firmware and hardware

This manual supports the following firmware:

- 114xx and above

Note This manual uses the placeholder value "xx" in the firmware versions listed above, as the manual documents the released features as of the time of its writing. Digi International periodically releases new firmware containing bug fixes and new features. As new firmware is released and distributor stock is refreshed, the new firmware will gradually become available without the need to update. However, no guarantees can be made that a specific version of the firmware will be populated on any given XBee as delivered. If a specific revision is desired, it is the user's responsibility to ensure that version is loaded onto all XBees purchased.

The documentation supports the following hardware.

- XB3-C-A2-UT-001

Safety instructions

Safety instructions

XBee adapter, gateways, and routers

- The XBee Adapter, Gateway, or Router products 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 Adapter, Gateway, or Router products 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 Adapter, Gateway, or Router product. Do not remove the product covers or modify the Gateway or Router in any way. Modifications may exclude the product from any warranty and can cause the gateway or router to operate outside of regulatory compliance for a given country, leading to the possible illegal operation of the product.
- Use industry standard ESD protection when handling the XBee Adapter, Gateway, or Router product.
- Take care while handling to avoid electrical damage to the PCB and components.
- Do not expose the XBee Adapter, Gateway, or Router products to water or moisture.
- Use this product with the antennas specified in the XBee Adapter, Gateway, or Router product user guides.
- The end user must be told how to remove power from the XBee Adapter, Gateway, or Router product 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):
  - orvosi eszközök
  - nukleáris alkalmazások
  - 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.

Las instrucciones de seguridad

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.

SIM cards

The XBee Smart Modem requires a 4FF nano-SIM card, which is the size normally used in most Smart phones. The SIM interface supports both 1.8 V and 3.3 V SIM types.

NB-IoT network

NB-IoT network is supported by XBee 3 Cellular. Note the following:

- NB-IoT does not support roaming. You cannot roam between networks.
- For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.
- For NB-IoT, SMS support is dependent on the network. Contact your network provider for details.

- Digi Remote Manager® requires TCP and will not work with NB-IoT unless the network supports TCP.
- The SIM card in the device determines whether the device supports NB-IoT, LTE-M, or both.

Get started with the XBee Smart Modem

This section describes how to connect the hardware in the XBee, and provides some examples you can use to communicate with the device.

You should perform all of the steps below in the order shown.

1. Identify the kit contents
2. Determine cellular service and acquire a SIM card
3. Connect the hardware
4. Install and upgrade XCTU
5. Use one of the following methods to verify your cellular connection. You must have a SIM card installed.
   - Connect to the Echo server
   - Connect to the ELIZA server
   - Connect to the Daytime server

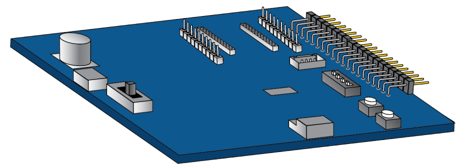
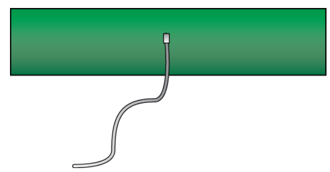
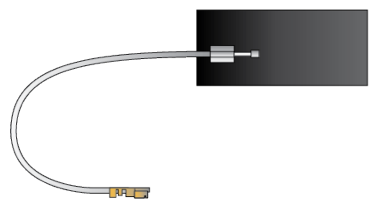
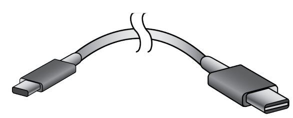
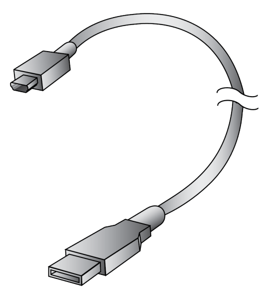
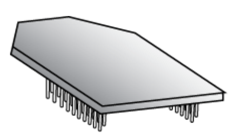
Optional steps

You can review the information in these steps for more XBee connection examples and examples of how to use MicroPython.

1. Review additional connection examples to help you learn how to use the device. See XBee connection examples.
2. Review introductory MicroPython examples. You can use MicroPython to enhance the intelligence of the XBee to enable you to do edge-computing by adding business logic in MicroPython, rather than using external components.
   - Example: hello world
   - Example: Turn on an LED

Identify the kit contents

The Developer's kit includes the following:

Item	Description
One XBIB-CU-TH board	
One cellular antenna with a U.FL connector	
One Bluetooth Low Energy (BLE) antenna	
One USB-C cable **Note** This cable is used to power the development board.	
One Micro USB cable **Note** This cable is used only with USB Direct mode. This cable will not power the development board.	
One XBee Smart Modem **Note** The XBee Smart Modem comes attached to the board in ESD wrap.	

Item	Description
One SIM card **Note** NB-IoT kits do not include a SIM card. Contact your NB-IoT mobile carrier provider to obtain a SIM card and service. See Determine cellular service and acquire a SIM card.	

Determine cellular service and acquire a SIM card

You need cellular service to use your XBee. Depending on the device that you purchased, your kit may not include a SIM card.

Note If your kit came with a SIM card, you can skip this section. If you are interested in purchasing a Cellular Bundled Service plan from Digi, see Cellular service.

If your kit does not include a SIM card, the following sections below explain how to purchase a SIM card in the US and Europe.

US customers

In the US, Digi XBee® 3 Cellular LTE-M/NB-IoT works with AT&T, Verizon, and T-Mobile. You must purchase a SIM card before you can connect the hardware. Contact Digi Sales at www.digi.com/contactus for information about obtaining a SIM card and activating cellular service.

After you have purchased your SIM card, you must get the APN from the carrier. You will need this information when you get service. See Configure your module for cellular connectivity.

European customers

If you are using the LTE-M/NB-IoT European kit, you must purchase a SIM card before you can connect the hardware. Contact your mobile carrier provider to obtain a SIM card and service.

- Vodafone: www.vodafone.com
- Deutsche Telekom: www.telekom.com/en

After you have purchased your SIM card, you can get the APN (if needed by your carrier), network bands, and supported channels from your carrier. You will need this information when configuring the device from the SIM card and service you have selected. See Configure your module for cellular connectivity.

Ensure that you choose a carrier and plan that supports the technologies and bands supported by the LTE Cat 1 smart modem. Your carrier may require you to enter the APN when configuring the smart modem.

Cellular service

Digi now offers Cellular Bundled Service plans, where you can choose to purchase a subscription for cell service, and/or a Digi Remote Manager package.

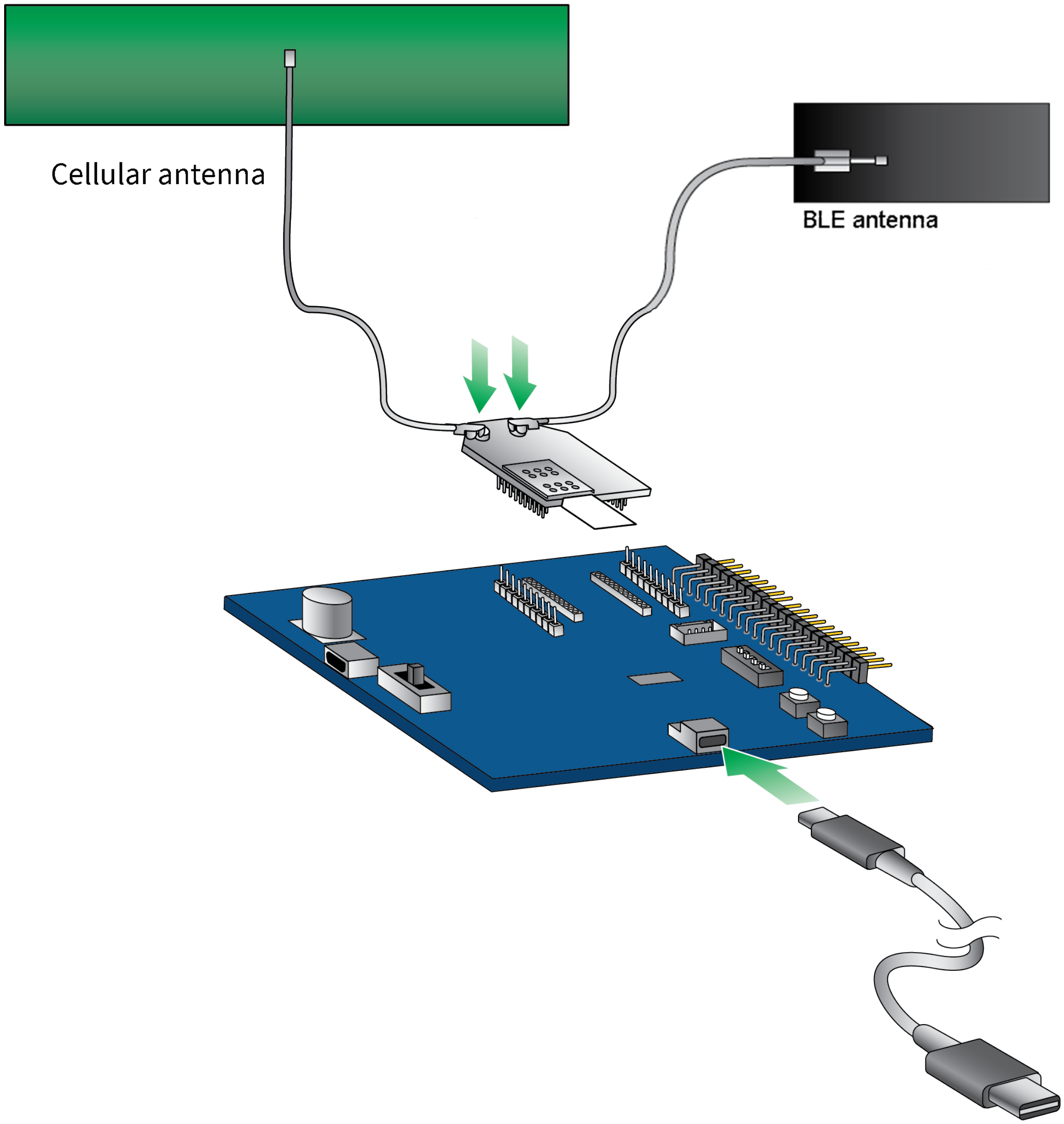
To shop online, go to: shop.digi.com

Note The Digi Cellular Bundled Service plan is not offered for NB-IoT. Contact Digi for more information about Digi Cellular Bundled Service.

To learn more, or obtain the plan that is right for your needs, contact us:

- By phone: 1-877-890-4014 (USA/toll free) or +1-952-912-3456 (International). Select the **Wireless Plan Support** or **Activation** option in the menu.
- By email: Data.Plan.QuoteDesk@digi.com.

Connect the hardware



1. The XBee Smart Modem should already be plugged into the development board. For more information about development boards, see Development boards.
2. If a SIM card is included with the kit, the card is inserted into the XBee. If a SIM card is not included, install the SIM card into the XBee before attaching the XBee device to the board.

Note Some kits do not include a SIM card. Contact your mobile carrier provider to obtain a SIM card and service. See Determine cellular service and acquire a SIM card.

WARNING! Never insert or remove the SIM card while the device is powered!

3. Connect the antennas.
   a. Connect the cellular antenna.
   b. Connect the BLE antenna if you are using BLE functionality. If you are not, you do not have to connect the BLE antenna.

Note Align the U.FL connectors carefully, then firmly press straight down to seat the connector. You should hear a snap when the antenna attaches correctly. Caution should be used when connecting or removing the U.FL. Digi recommends using a U.FL removal tool.

4. Connect the USB-C cable from a PC to the USB port on the development board. The computer searches for a driver, which can take a few minutes to install.

Note The USB-C cable must be plugged into a port that will supply a minimum of 1 Amp of current for the device to work as expected.

Install and upgrade XCTU

XBee Configuration and Test Utility (XCTU) is a multi-platform program developed by Digi 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.

XCTU does not work directly over an SPI interface.

You can use XCTU to update the device firmware, and if needed, XCTU will attempt to update your cellular firmware. Firmware is the program code stored in the device's persistent memory that provides the control program for the device.

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

Note If you are on a macOS computer and encounter problems installing XCTU, see Correct a macOS Java error.

Step 1: Install and upgrade XCTU

You can use XCTU to update the device firmware.

1. To use XCTU, you may need to install FTDI Virtual COM port (VCP) drivers onto your computer. Click here to download the drivers for your operating system.
2. Upgrade XCTU to the latest version. This step is required.

Step 2: Add a device to XCTU

You must add a device to XCTU before you can update the device's firmware or configure the device from XCTU.

Add a device to XCTU

These instructions show you how to add the XBee to XCTU.

If XCTU does not find your serial port, see Cannot find the serial port for the device and Enable Virtual COM port (VCP) on the driver.

1. Launch XCTU .

Note XCTU's **Update the radio module firmware** dialog box may open and will not allow you to continue until you click **Update** or **Cancel** on the dialog.

2. Click **Help** > **Check for XCTU Updates** to ensure you are using the latest version of XCTU.
3. Click the **Discover radio modules** button in the upper left side of the XCTU screen.
4. In the **Discover radio devices** dialog, select the serial ports where you want to look for XBee modules, and click **Next**.
5. In the **Set port parameters** window, maintain the default values and click **Finish**.
6. As XCTU locates radio modules, they appear in the **Discovering radio modules** dialog box.
7. Select the device(s) you want to add and click **Add selected devices**.

If your module could not be found, XCTU displays the **Could not find any radio module** dialog providing possible reasons why the module could not be added.

Update the device and cellular firmware using XCTU

You should use XCTU to update the device firmware on your XBee 3 to the most recent version. This ensures that you can take advantage of all the latest fixes and features. XCTU will update the device firmware, and if needed, XCTU will attempt to update your cellular modem firmware. Upgrading the cellular modem component firmware requires USB Direct.

Update the device and cellular firmware using XCTU and USB Direct access.

Configure your module for cellular connectivity

Note LTE-M is configured by default. You can skip this section if you are using LTE-M.

If you are using an NB-IoT kit, you must configure the device to use NB-IoT.

US customers

Note Some carriers require an APN. If the carrier does not require an APN, you should not change the APN from the default

1. Launch XCTU .
2. Click the **Configuration working modes** button
3. Select an XBee module from the **Radio Modules** list.
4. Set the APN using the AN command. You should get the APN from your carrier when you purchased your SIM card. See Determine cellular service and acquire a SIM card.
5. To set the APN, in the **AN** field, type the APN value from your carrier and click the **Write** button .

European customers

Note Some carriers require an APN. If the carrier does not require an APN, you should not change the APN from the default

1. Launch XCTU .
2. Click the **Configuration working modes** button .
3. Select an XBee module from the **Radio Modules** list.
4. Set the APN using the AN command. You should get the APN from your carrier when you purchased your SIM card. See Determine cellular service and acquire a SIM card.

   To set the APN, in the **AN** field, type the APN value from your carrier and click the **Write** button .
5. Set the N# parameter to 3. In the **N#** field, select **NB-IoT Only [3]** and click the **Write** button .
6. To set the **CP**, in the **CP** field, select **No Profile (1)** and click the **Write** button .
7. Reset the module with either the reset button or issue the **FR** command.
8. Wait for a connection. You may wait for up to 5-6 minutes.
   - **If you have a connection**: This process is complete. The LED on the development board blinks when the XBee is registered to the cellular network. See Check for cellular registration and connection.
   - **If you do not have a connection**: If the LED remains solid, registration has not occurred properly.
     a. Repeat steps 1-5 to make sure you have correctly configured NB-IoT.
     b. If you still do not have a connection, contact your carrier to confirm that the carrier has correctly configured the service.
        - If the carrier makes a change to the service, reset the module and wait 5-6 minutes.
        - If the carrier does not make a change to the service, then contact Digi support.

Note If you are having trouble attachment to the network, see Verify that radio channels match your carrier.

Check for cellular registration and connection

The cellular network registration and address assignment must occur successfully. To verify the network connection, you can view the LED on the development board or check the status of the relevant commands in XCTU.

Registration typically takes 5 - 6 minutes the first time a device is connected to the network.

Before you begin

- A working SIM card is required. See Determine cellular service and acquire a SIM card.
- Make sure you have added the device to XCTU. See Add a device to XCTU.

- Make sure you are in an area with adequate cellular network reception.
- Verify that the antennas are connected properly to the device.

View LED action

The LED on the development board blinks when the XBee is registered to the cellular network; see Associate LED functionality. If the LED remains solid, registration has not occurred properly.

View commands in XCTU

1. Launch XCTU .
2. Click the **Configuration working mode** button.
3. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
4. Verify the status of your network connection using the following commands:
   - **AI (Association Indication)** reads **0** when the device successfully registers to the cellular network and the LED is blinking. If it reads **23** it is connecting to the Internet; **22** means it is registering to the cellular network.
   - **MY (Module IP Address)** should display a valid IP address. If it reads **0.0.0.0**, it has not registered yet.

Hints

- To search for an AT command in XCTU, use the search box .
- To read a command's value, click the **Read** button next to the command.

XBee connection examples

The following examples provide some additional scenarios you can try to get familiar with the XBee. These examples are focused on inter-operating with a host processor to drive the XBee.

If you are interested in using the intelligence built into the XBee, see Get started with MicroPython.

Note Some carriers restrict your internet access. If access is restricted, running some of these examples may not be possible. Check with your carrier provider to determine whether internet access is restricted.

Connect to the Echo server

This server echoes back the messages you type.

You may use TCP or UDP, depending on the protocols supported by your network carrier.

Note For help with debugging, see Debugging.

The following table explains the AT commands that you use in this example.

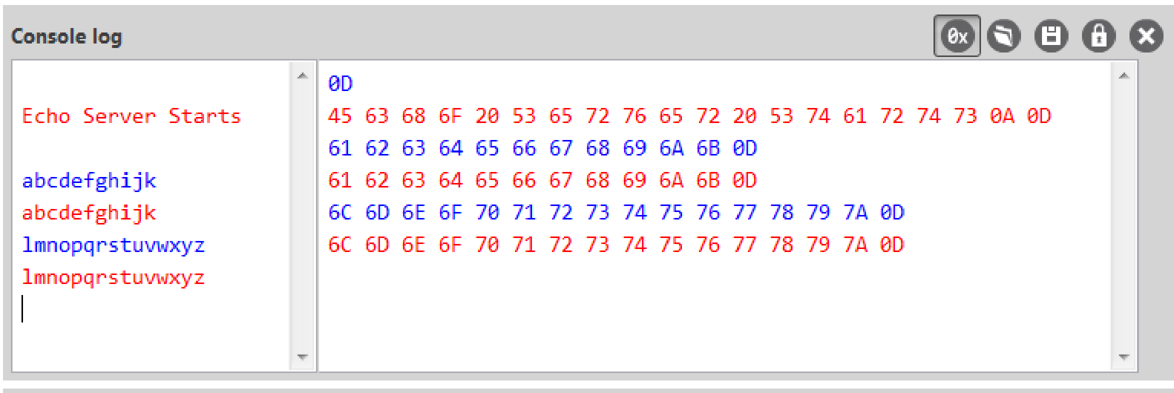
At command	Value	Description
IP (IP Protocol)	1	TCP: Set the expected transmission mode to TCP communications.
	0	UDP: Set the expected transmission mode to UDP communications.
TD (Text Delimiter)	D (0x0D)	The text delimiter to be used for Transparent mode, as an ASCII hex code. No information is sent until this character is entered, unless the maximum number of characters has been reached. Set to **0** to disable text delimiter checking. Set to **D** for a carriage return.
DL (Destination Address)	52.43.121.77	The target IP address of the echo server. **Note** Some carriers may require whitelisted IP addresses. If this IP is not whitelisted by your carrier you will not be able to run this example.
DE (Destination Port)	2329 (0x2329)	TCP: The target port number of the TCP echo server. This port in decimal is 9001.
	2711 (0x2711)	UDP: The target port number of the UDP echo server. This port in decimal is 10001.

To communicate with the Echo server:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in Connect the hardware.
2. Open XCTU and Add a device to XCTU.
3. Click the **Configuration working mode** button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
5. To switch to TCP communication, in the **IP** field, select 1 and click the **Write** button .
6. To enable the XBee to recognize carriage return as a message delimiter, in the **TD** field, type **D** and click the **Write** button.
7. To enter the destination address of the echo server, in the **DL** field, type **52.43.121.77** and click the **Write** button.
8. To enter the destination IP port number, in the **DE** field, type **2329** and click the **Write** button.

Note XCTU does not follow the standard hexadecimal numbering convention. The leading 0x is not needed in XCTU.

9. Click the **Consoles working mode** button on the toolbar to open a serial console to the device. For instructions on using the Console, see the AT console topic in the *XCTU User Guide*.
10. Click the **Open** button to open a serial connection to the device.
11. Click in the left pane of the **Console log**, then type in the Console to talk to the echo server. The following screenshot provides an example of this chat.



Connect to the ELIZA server

You can use the XBee to chat with the ELIZA Therapist Bot. ELIZA is an artificial intelligence (AI) bot that emulates a therapist and can perform simple conversations.

Note For help with debugging, see Debugging.

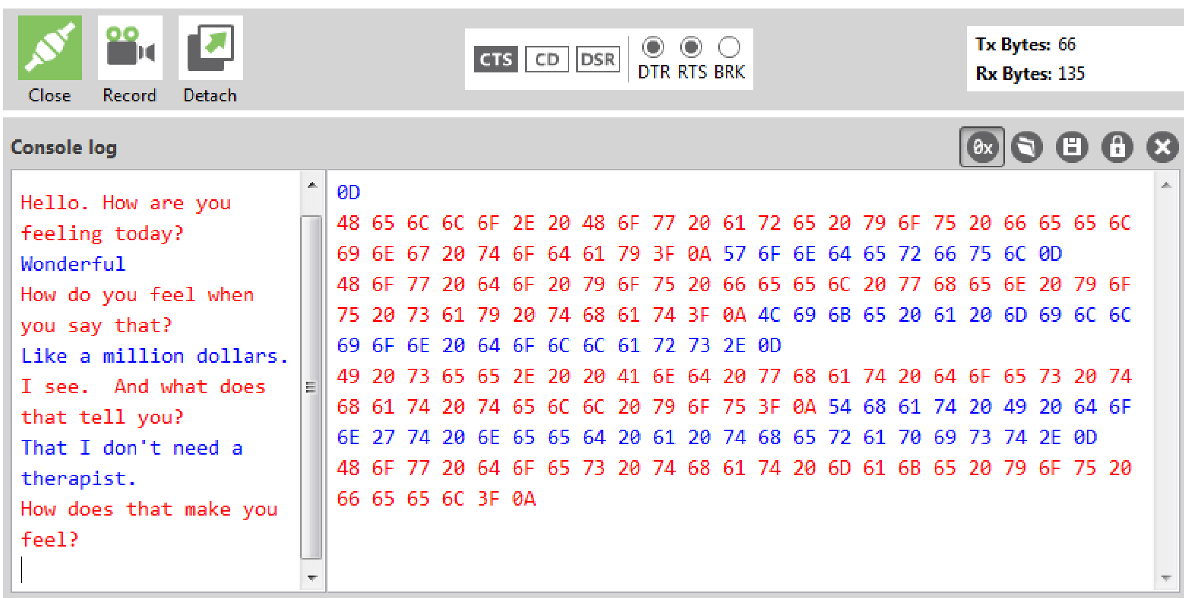
The following table explains the AT commands that you use in this example.

At command	Value	Description
IP (IP Protocol)	1	Set the expected transmission mode to TCP communications.
DL (Destination Address)	52.43.121.77	The target IP address of the ELIZA server. **Note** Some carriers may require whitelisted IP addresses. If this IP is not whitelisted by your carrier you will not be able to run this example.
DE (Destination Port)	2328 (0x2328)	The target port number of the ELIZA server.

To communicate with the ELIZA Therapist Bot:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in Connect the hardware.
2. Open XCTU and Add a device to XCTU.

3. Click the **Configuration working mode** button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
5. To switch to TCP communication, in the **IP** field, select 1 and click the **Write** button .
6. To enter the destination address of the ELIZA Therapist Bot, in the **DL** field, type **52.43.121.77** and click the **Write** button.
7. To enter the destination IP port number, in the **DE** field, type **2328** and click the **Write** button.
8. Click the **Consoles working mode** button on the toolbar to open a serial console to the device. For instructions on using the Console, see the AT console topic in the *XCTU User Guide*.
9. Click the **Open** button to open a serial connection to the device.
10. Click in the left pane of the **Console log**, then type in the Console to talk to the ELIZA Therapist Bot. The following screenshot provides an example of this chat with the user's text in blue.



Connect to the Daytime server

The Daytime server reports the current Coordinated Universal Time (UTC) value responding to any user input.

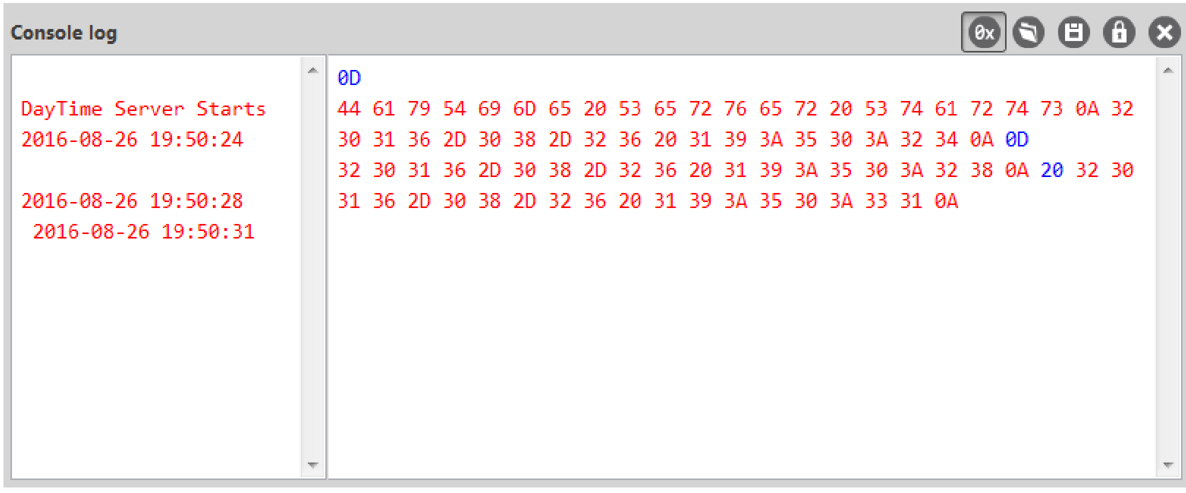
Note For help with debugging, see Debugging.

The following table explains the AT commands that you use in this example.

At command	Value	Description
IP (IP Protocol)	1	Set the expected transmission mode to TCP communications.
DL (Destination Address)	52.43.121.77	The target IP of the Daytime server. **Note** Some carriers may require whitelisted IP addresses. If this IP is not whitelisted by your carrier you will not be able to run this example.
DE (Destination Port)	232A (0x232A)	The target port number of the Daytime server.
TD (Text Delimiter)	0	The text delimiter to be used for Transparent mode, as an ASCII hex code. No information is sent until this character is entered, unless the maximum number of characters has been reached. Set to zero to disable text delimiter checking.

To communicate with the Daytime server:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in Connect the hardware.
2. Open XCTU and Add a device to XCTU.
3. Click the **Configuration working mode** button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
5. To switch to TCP communication, in the **IP** field, select 1 and click the **Write** button .
6. To enter the destination address of the daytime server, in the **DL** field, type **52.43.121.77** and click the **Write** button.
7. To enter the destination IP port number, in the **DE** field, type **232A** and click the **Write** button.
8. To disable text delimiter checking, in the **TD** field, type **0** and click the **Write** button.
9. Click the **Consoles working mode** button on the toolbar to open a serial console to the device. For instructions on using the Console, see the AT console topic in the *XCTU User Guide*.
10. Click the **Open** button Open to open a serial connection to the device.
11. Click in the left pane of the **Console log**, then type in the Console to query the Daytime server. The following screenshot provides an example of this chat.



Perform a (GET) HTTP request

You can use the XBee to perform a GET Hypertext Transfer Protocol (HTTP) request using XCTU. HTTP is an application-layer protocol that runs over TCP. This example uses httpbin.org/ as the target website that responds to the HTTP request.

Note For help with debugging, see Debugging.

To perform a GET request:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in Connect the hardware.
2. Open XCTU and Add a device to XCTU.
3. Click the **Configuration working mode** button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
5. To enter the destination address of the target website, in the **DL** field, type **httpbin.org** and click the **Write** button .
6. To enter the HTTP request port number, in the **DE** field, type **50** and click the **Write** button. Hexadecimal **50** is 80 in decimal.
7. To switch to TCP communication, in the **IP** field, select **1** and click the **Write** button.
8. To move into Transparent mode, in the **AP** field, select **0** and click the **Write** button.
9. Wait for the **AI** (Association Indication) value to change to **0** (Connected to the Internet).
10. Click the **Consoles working mode** button on the toolbar.
11. From the AT console, click the **Add new packet button** in the Send packets dialog. The **Add new packet** dialog appears.
12. Enter the name of the data packet.
13. Type the following data in the **ASCII** input tab:

    GET /ip HTTP/1.1

Host: httpbin.org

14. Click the **HEX** input tab and add **0A** (zero A) after each **0D** (zero D), and add an additional **0D 0A** at the end of the message body. For example, copy and past the following text into the **HEX** input tab:

    47 45 54 20 2F 69 70 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F 73 74 3A 20 68 74 74 70 62 69 6E 2E 6F 72 67 0D 0A 0D 0A

Note The HTTP protocol requires an empty line (a line with nothing preceding the CRLF) to terminate the request.

15. Click **Add packet**.
16. Click the **Open** button.
17. Click **Send selected packet**.
18. A GET HTTP response from httpbin.org appears in the Console log.

Connect to a TCP/IP address

The XBee Smart Modem can send and receive TCP messages while in Transparent mode; see Transparent operating mode.

Note You can use this example as a template for sending and receiving data to or from any TCP/IP server.

Note For help with debugging, see Debugging.

The following table explains the AT commands that you use in this example.

Command	Value	Description
IP (IP Protocol)	1	Set the expected transmission mode to TCP communication.
DL (Destination IP Address)	<Target IP address>	The target IP address that you send and receive from. For example, a data logging server's IP address that you want to send measurements to.
DE (Destination Port)	<Target port number>	The target port number that the device sends the transmission to. This is represented as a hexadecimal value.

To connect to a TCP/IP address:

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in Connect the hardware.
2. Open XCTU and Add a device to XCTU.
3. Click the **Configuration working mode** button.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.

5. In the **IP** field, select 1 and click the **Write** button .
6. In the **DL** field, type the <**target IP address**> and click the **Write** button. The target IP address is the IP address that you send and receive from.
7. In the **DE** field, type the <**target port number**>, converted to hexadecimal, and click the **Write** button.
8. Exit Command mode.

After exiting Command mode, any UART data sent to the device is sent to the destination IP address and port number after the RO (Packetization Timeout) occurs.

Debugging

If you experience problems with the settings in the examples, you can load the default settings in XCTU.

Note If you load the default settings, you will need to reapply any configuration settings that you have previously made.

1. On the Configuration toolbar, click the **Default** button to load the default values established by the firmware, and click **Yes** to confirm.
2. Factory settings are loaded but not written to the device. To write them, click the **Write** button on the toolbar.

Software libraries

One way to communicate with the XBee device is by using a software library. The libraries available for use with the XBee Smart Modem include:

- XBee Java library
- XBee Python library
- XBee ANSI C 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.

Get started with MicroPython

This section provides an overview and simple examples of how to use MicroPython with the XBee Smart Modem. You can use MicroPython to enhance the intelligence of the XBee to enable you to do edge-computing by adding business logic in MicroPython, rather than using external components.

Note For in-depth information and more complex code examples, refer to the *Digi MicroPython Programming Guide*.

About MicroPython

MicroPython is an open-source programming language based on Python 3, with much of the same syntax and functionality, but modified to fit on small devices with limited hardware resources, such as microcontrollers, or in this case, a cellular modem.

Why use MicroPython

MicroPython enables on-board intelligence for simple sensor or actuator applications using digital and analog I/O. MicroPython can help manage battery life. Cryptic readings can be transformed into useful data, excess transmissions can be intelligently filtered out, modern sensors and actuators can be employed directly, and logic can glue inputs and outputs together in an intelligent way.

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

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

MicroPython on the XBee Smart Modem

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

Note MicroPython does not work with SPI.

The examples in this guide assume:

- You have XCTU on your computer. See Install and upgrade XCTU.
- You have a terminal program installed on your computer. We recommend using the Use the MicroPython Terminal in XCTU. This requires XCTU 6.3.7 or higher.
- You have an XBee Smart Modem installed in an appropriate development board, such as an XBIB-U-DEV.

Note Most examples in this guide require the XBIB-U-DEV board.

- The XBee Smart Modem is connected to the computer via a USB cable and XCTU recognizes it.
- The board is powered by an appropriate power supply: 12 VDC.

Use XCTU to enter the MicroPython environment

To use the XBee Smart Modem in the MicroPython environment:

1. Use XCTU to add the device(s); see Install and upgrade XCTU and Add a device to XCTU.
2. The XBee Smart Modem 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.
4. Put the XBee Smart Modem into MicroPython mode, in the **AP** field select **MicroPython REPL [4]** and click the **Write** button .
5. Note what COM port(s) the XBee Smart Modem is using, because you will need this information when you use terminal communication. The **Radio Modules** information panel lists the COM port in use.

Use the MicroPython Terminal in XCTU

You can use the MicroPython Terminal to communicate with the XBee Smart Modem when it is in MicroPython mode.[1] This requires XCTU 6.3.7 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 opens.
2. Click **Open**. If you have not already added devices to XCTU:
   a. In the **Select the Serial/USB port** area, click the COM port that the device uses.
   b. Verify that the baud rate and other settings are correct.
3. Click **OK**. The **Open** icon changes to **Close**, indicating that the device is properly connected.
4. Press **Ctrl**+**B** to get the MicroPython version banner and prompt.

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

Troubleshooting

If you receive **No such port: 'Port is already in use by other applications.'** in the **MicroPython Terminal** close any other console sessions open inside XCTU and close any other serial terminal programs connected to the device, then retry the MicroPython connection in XCTU.

If the device seems unresponsive, try pressing **Ctrl+C** to end any running programs.

You can use the **+++** escape sequence and look for an **OK** for confirmation that you have the correct baud rate.

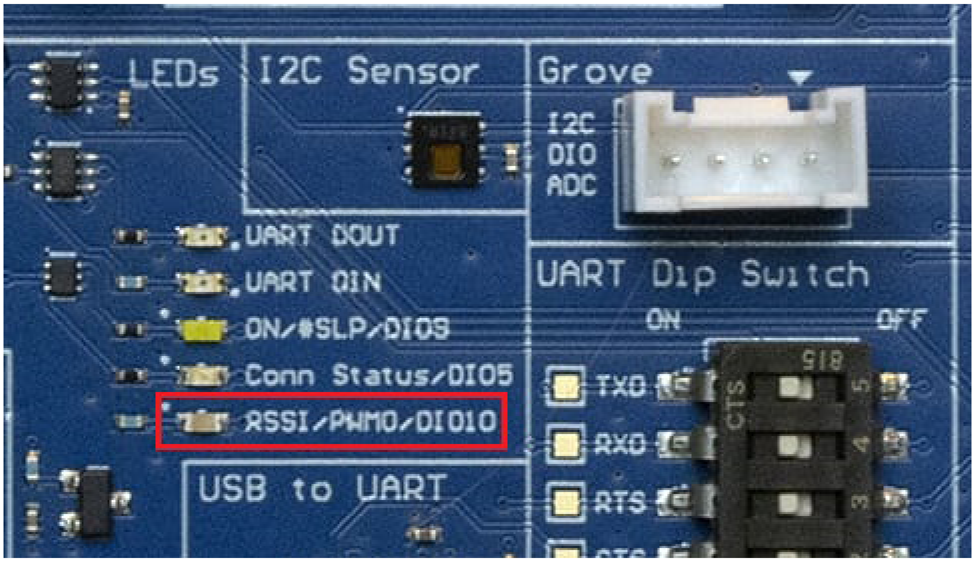
Example: hello world

Before you begin, you must have previously added a device in XCTU. See Add a device to XCTU.

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!**.

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

Example: Turn on an LED



1. Note the **DIO10** LED on the XBIB board. The following image highlights it in a red box. The LED is normally off.
2. At the MicroPython **>>>** prompt, type the commands below, pressing **Enter** after each one. After entering the last line of code, the LED illuminates. Anything after a **#** symbol is a comment, and you do not need to type it.

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.

```
from machine import Pin
led = Pin("D10", Pin.OUT, value=1)  # Makes a pin object set to output 1.
```

3. To turn it off, type the following and press **Enter**:

```
led.value(0)
```

You have successfully controlled an LED on the board using basic I/O.

Example: Debug the secondary UART

This sample code is handy for debugging the secondary UART. It simply relays data between the primary and secondary UARTs.

```
from machine import UART
import sys, time

def uart_init():
    u = UART(1)
    u.write('Testing from XBee\n')
    return u

def uart_relay(u):
```

```
        while True:
            uart_data = u.read(-1)
            if uart_data:
                sys.stdout.buffer.write(uart_data)
            stdin_data = sys.stdin.buffer.read(-1)
            if stdin_data:
                u.write(stdin_data)

            time.sleep_ms(5)

    u = uart_init()
    uart_relay(u)
```

You only need to call **uart_init()** once.

Call **uart_relay()** to pass data between the UARTs.

Send **Ctrl-C** to exit relay mode.

When done, call **u.close()** to close the secondary UART.

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 Smart Modem. If you use Microsoft Windows, follow the instructions for Tera Term, if you use Linux, follow the instructions for picocom. To download these programs:

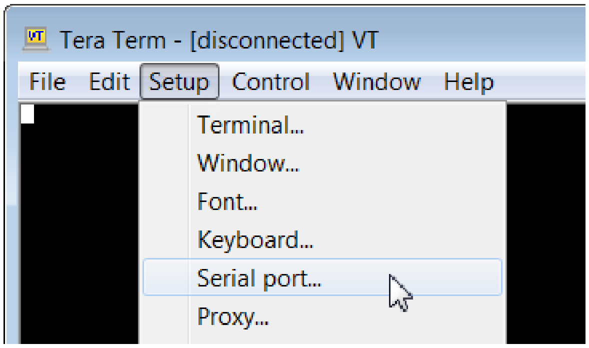
- Tera Term for Windows. See https://ttssh2.osdn.jp/index.html.en.
- PuTTY for Windows
- Picocom for Linux. See https://developer.ridgerun.com/wiki/index.php/Setting_up_Picocom_-_Ubuntu and for the source code and in-depth information https://github.com/npat-efault/picocom.

Tera Term for Windows

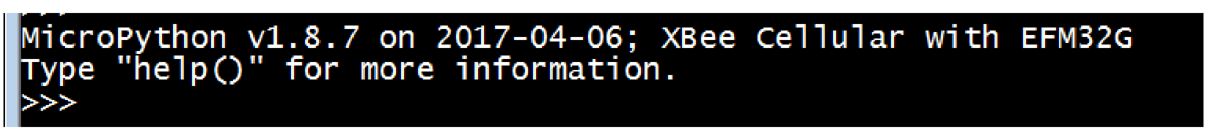
With the XBee Smart Modem in MicroPython mode (**AP** = **4**), you can access the MicroPython prompt using a terminal.

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 Smart Modem 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 b/s.
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 Smart Modem is connected on.
   - **Baud rate**: 9600
   - **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 **CR** 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.



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

Use picocom in Linux

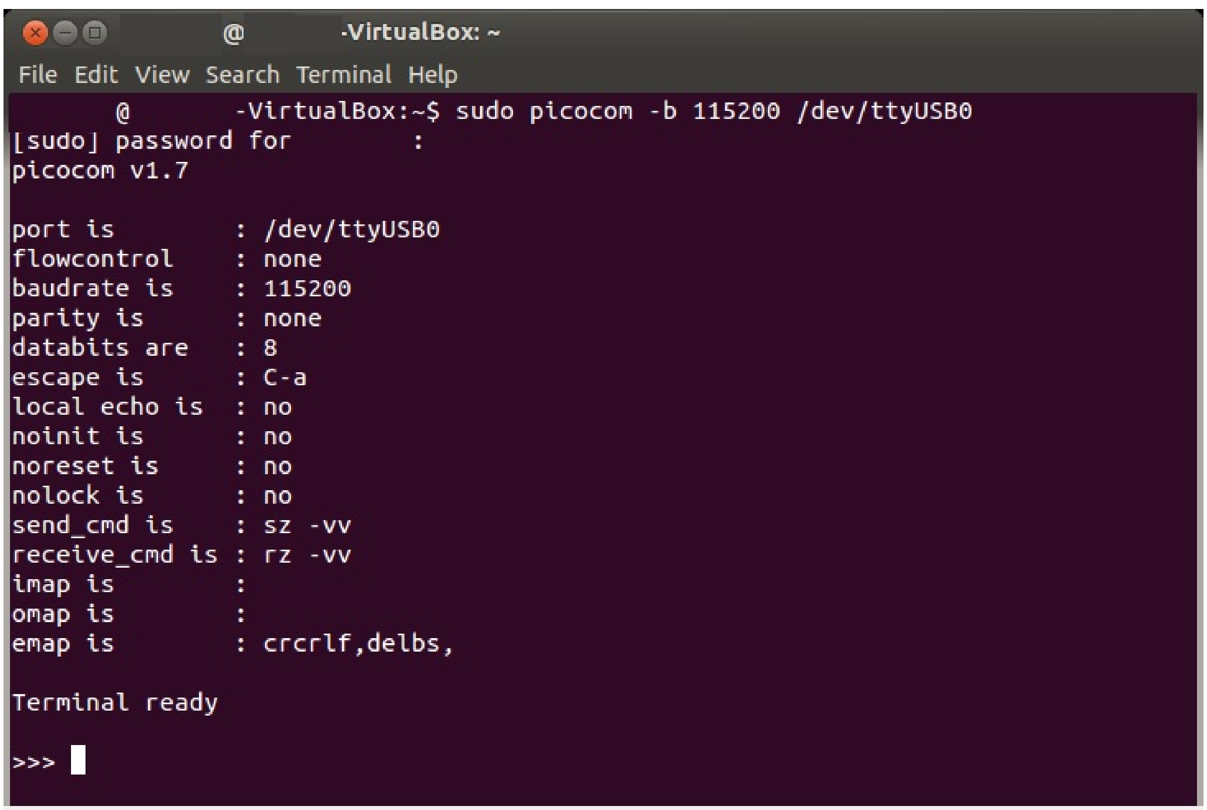
With the XBee Smart Modem in MicroPython mode (**AP** = **4**), you can access the MicroPython prompt using a terminal.

Note The user must have read and write permission for the serial port the XBee Smart Modem is connected to in order to communicate with the device.

1. Open a terminal in Linux and type **picocom -b 9600 /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 Smart Modem, 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 Smart Modem.
3. Check the directory again and you should see one additional device, which is the XBee Smart Modem.
4. In this case, replace **/dev/ttyUSB0** at the top with **/dev/ttyUSB<number>**, where **<number>** is the new number that appeared.
5. It should connect and show Terminal ready.



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

Get started with Bluetooth® Low Energy

BLE (**Bluetooth**® Low Energy) is an RF protocol that enables you to connect your XBee (server) device to another (client) device. he latest Digi XBee products include a dual-mode radio that allows the device to communicate through the BLE interface and the RF/Cellular network at the same time.

The XBee acts as a BLE GATT server and allows client devices, such as a cellphone or a third-party BLE device such as the Nordic nRF and SiLabs BGM, to configure the XBee or transfer data with the User Data Relay frame using the XBee API BLE Service.

The XBee does not support modifying the XBee's GATT database. This means that the XBee cannot be configured to appear as something else, such as a temperature sensor.

On XBee 3 Cellular firmware ending in x16 or newer

The XBee supports the following BLE features:

- BLE pairing and bonding support for GATT client connections.
- Ability to authenticate and communicate as a BLE client to other XBee3 devices using the Digi BLE service.

On XBee 3 Cellular firmware ending in x15 or newer

The XBee supports the following BLE features:

- GATT server supporting the XBee API BLE Service and the standard Device Information Service. Note that support for these features was added in firmware ending in x10.
- Configurable BLE device name in advertisements using the ATBI command.
- Configurable BLE advertisement payload, such as beacons, using MicroPython.
- GAP (Generic Access Profile) discovery (scanning for advertisements or beacons) using MicroPython.
- GATT client support in MicroPython.
    - Connect to another BLE device (such as a sensor) using GAP.
    - Discover services, characteristics, and descriptors on the remote device’s GATT server.
    - Read and write characteristic and descriptor values. For example, read a temperature sensor.
    - Configure notifications or indications on characteristic values.

For more information, see the *Digi MicroPython Programming Guide*.

On XBee 3 Cellular firmware ending in x14 or older

The XBee supports the following BLE features:

- GATT server supporting the XBee API BLE Service and the standard Device Information Service.

The XBee is a GATT server only, which can be used to configure the XBee or transfer data with the User Data Relay Frame.

The XBee does **not** support the following:

- Scanning for advertisements or beacons.
- Connecting to another BLE device, such as a sensor or another XBee.

Enable BLE on an XBee device

This process explains how to enable BLE on your XBee 3 device and verify the connection.

1. Set up your XBee device, and make sure to connect the BLE antenna to the device. See Get started with the XBee Smart Modem.
2. Enable BLE and configure the BLE password using XCTU.
3. Get the Digi XBee Mobile phone application.
4. Connect with BLE and configure your XBee device.

Note The BLE protocol is disabled on the XBee device by default. To ensure that BLE is always enabled, you can create a custom configuration that is used as a new factory default. See Custom configuration: Create a new factory default.

Connect with BLE and configure your XBee device

You can use the Digi XBee Mobile application to verify that BLE is enabled on your XBee device.

1. Get the Digi XBee Mobile phone application.
2. Open the Digi XBee Mobile application. The **Find XBee devices** screen appears and the app automatically begins scanning for devices. All nearby devices with BLE enabled are displayed in a list.
3. Scroll through the list to find your XBee device.

   The first time you open the app on a phone and scan for devices, the device list contains only the name of the device and the BLE signal strength. No identifying information for the device displays. After you have authenticated the device, the device information is cached on the phone. The next time the app on this phone connects to the XBee device, the IMEI for the device displays in the app device list.

4. Tap the XBee device name in the list. A password dialog appears.
5. Enter the password you previously configured for the device in XCTU.
6. Tap **OK**. The **Device Information** screen displays. You can now scroll through the settings for the XBee device and change the device's configuration as needed.

Enable BLE and configure the BLE password using XCTU

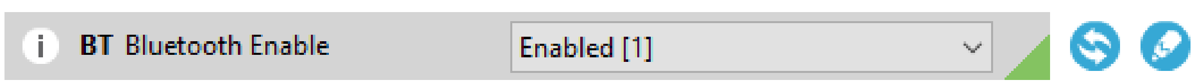
Some of the latest XBee 3 modules support Bluetooth Low Energy (BLE) as an extra interface for configuration. If you want to use this feature, you have to enable BLE. You must also enable security by setting a BLE password on the XBee device in order to connect, configure, or send data over BLE.

The BLE password is configured using XCTU. Make sure you have installed or updated XCTU to version 6.4.2. or later. Earlier versions of XCTU do not include the BLE configuration features. See Download and install XCTU for installation instructions.

Before you begin, you should determine the password you want to use for BLE on the XBee device and store it in a secure place. Digi recommends a secure password of at least 8 characters and a random combination of letters, numbers, and special characters. Digi also recommends using a security management tool such as Bitwarden or Keepass for generating and storing passwords for many devices.

Note When you enter the BLE password in XCTU, the salt and verifier values are calculated as you set your password. For more information on how these values are used in the authentication process, see BLE Unlock API - 0x2C.

1. Launch XCTU .
2. Switch to Configuration working mode .
3. Select a BLE compatible radio module from the device list.
4. In the Bluetooth Options section, select **Enabled[1]** from the **BT Bluetooth Enable** command drop-down.



5. Click the **Write setting** button . The **Bluetooth authentication not set** dialog appears.

   Note If BLE has been previously configured, the **Bluetooth authentication not set** dialog does not appear. If this happens, click **Configure** in the Bluetooth Options section to display the **Configure Bluetooth Authentication** dialog.

6. Click **Configure** in the dialog. The **Configure Bluetooth Authentication** dialog appears.
7. In the **Password** field, type the password for the device. As you type, the **Salt** and **Verifier** fields are automatically calculated and populated in the dialog as shown above. Make a note of the password, as this password is used when you connect to this XBee device via BLE using the Digi XBee Mobile app.
8. Click **OK** to save the configuration.

Get the Digi XBee Mobile phone application

To see the nearby devices that have BLE enabled, you must get the free Digi XBee Mobile application from the iOS App Store or Google Play and downloaded to your phone.

1. On your phone, go to the App store.
2. Search for **Digi XBee Mobile**.
3. Download and install the application.

The Digi XBee Mobile application is compatible with the following operating systems and versions:

- Android 5.0 or higher
- iOS 11 or higher

BLE reference

BLE advertising behavior and services

When the Bluetooth radio is enabled, periodic BLE advertisements are transmitted. The advertisement data includes the product name. When an XBee device connects to the Bluetooth radio, the BLE services are listed:

- Device Information Service
- XBee API BLE Service

Device Information Service

The standard Device Information Service is used. The Manufacturer, Model, and Firmware Revision characters are provided inside the service.

XBee API BLE Service

You can configure the XBee through the BLE interface using API frame requests and responses. The API frame format through Bluetooth is equivalent to setting AP=1 and transmitting the frames over the UART or SPI interface. API frames can be executed over Bluetooth regardless of the AP setting.

The BLE interface allows these frames:

- BLE Unlock API - 0x2C
- BLE Unlock Response - 0xAC
- AT Command - 0x08
- User Data Relay - 0x2D

This API reference assumes that you are familiar with Bluetooth and GATT services. The specifications for Bluetooth are an open standard and can be found at the following links:

- Bluetooth Core Specifications: https://www.bluetooth.com/specifications/bluetooth-core-specification
- Bluetooth GATT: https://www.bluetooth.com/specifications/gatt/generic-attributes-overview

The XBee API GATT Service contains two characteristics: the API Request characteristic and the API Response characteristic. The UUIDs for the service and its characteristics are listed in the table below.

Characteristic	UUID
API Service UUID	53da53b9-0447-425a-b9ea-9837505eb59a
API Request Characteristic UUID	7dddca00-3e05-4651-9254-44074792c590
API Response Characteristic UUID	f9279ee9-2cd0-410c-81cc-adf11e4e5aea

API Request characteristic

UUID: 7dddca00-3e05-4651-9254-44074792c590

Permissions: Writeable

XBee API frames are broken into chunks and transmitted sequentially to the request characteristic using write operations. Valid frames will then be processed and the result will be returned through indications on the response characteristic.

API frames do not need to be written completely in a single write operation to the request characteristic. In fact, Bluetooth limits the size of a written value to 3 bytes smaller than the configured MTU (Maximum Transmission Unit), which defaults to 23, meaning that by default, you can only write 20 bytes at a time.

After connecting, you must perform the unlock process to authenticate the client. If the unlock process has not been completed successfully, all other API frames will be silently ignored and not processed.

API Response characteristic

UUID: f9279ee9-2cd0-410c-81cc-adf11e4e5aea

Permissions: Readable, Indicate

Responses to API requests made to the request characteristic will be returned through the response characteristics. This characteristic cannot be read directly.

Response data will be presented through indications on this characteristic. Indications are acknowledged and re-transmitted at the BLE link layer and application layer and provides a robust transport for this data.

Get started with Digi Remote Manager

Digi Remote Manager® is a cloud-based device and data management platform that you can use to configure and update a device, and view and manage device data.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

The sections below describe how to create a Remote Manager account, upgrading your device, configure your device, and manage data in Remote Manager.

1. Create a Remote Manager account and add devices
2. To ensure that all Remote Manager features are available, you should upgrade your device to the latest firmware. See Update the firmware from the Devices page in Remote Manager or Update the firmware using web services in Remote Manager.
3. Configure your device in Remote Manager

   To be able to configure your device in Remote Manager, the device must be connected to Remote Manager. You can connect to and configure your device in Remote Manager using one of the following methods:

   - **Scheduled connection**: In this method, you create a list of tasks that you want to perform on the device, and then start the operation. This is the recommended method, and is the best choice for low data usage. See Configure Remote Manager features using automations.
   - **Always connected**: This method can be used for initial configuration, or when you are not concerned with low data usage. See Configure XBee settings within Remote Manager.
4. Secure the connection between an XBee and Remote Manager with server authentication.
5. Manage data in Remote Manager
6. Remote Manager reference

Create a Remote Manager account and add devices

To be able to use Remote Manager, you must create a Remote Manager account and add your XBee devices to the device list. You should also verify that the device is enabled to connect to Remote Manager.

1. Create a Remote Manager account.
2. Add an XBee Smart Modem to Remote Manager.
3. Verify the connection between a device and Remote Manager

Create a Remote Manager account

Digi Remote Manager is an on-demand service with no infrastructure requirements. Remote devices and enterprise business applications connect to Remote Manager through standards-based web services. This section describes how to configure and manage an XBee using Remote Manager. For detailed information on using Remote Manager, refer to the *Remote Manager User Guide*, available via the **Documentation** tab in Remote Manager.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

Before you can manage an XBee with Remote Manager, you must create a Remote Manager account. To create a Remote Manager account:

1. Go to https://www.digi.com/products/iot-software-services/digi-remote-manager.
2. Click **90 DAY FREE TRIAL/LOGIN**.
3. Follow the online instructions to complete account registration. You can upgrade your Developer account to a paid account at any time.

When you are ready to deploy multiple XBee Smart Modems in the field, upgrade your account to access additional Remote Manager features.

Add an XBee Smart Modem to Remote Manager

Each XBee Smart Modem must be added to the Remote Manager account inventory list.

Before adding an XBee to your Remote Manager account inventory, you need to determine the International Mobile Equipment Identity (IMEI) number for the device. Use XCTU to view the IMEI number by querying the **IM** parameter.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

To add an XBee to your Remote Manager account inventory, follow these steps:

1. Log into Remote Manager.
2. Click **Devices**.
3. Click **Add**.
4. In the **Device ID, MAC Address or IMEI** field, type or paste the IMEI number of the XBee you want to add.
5. Click **Add Device** to add the device. The XBee is added to your inventory.

Verify the connection between a device and Remote Manager

By default, the XBee is configured to enable communication with Remote Manager. The communication between XBee and Remote Manager is achieved using periodic UDP operations.

You should verify the default settings to ensure that communcation will work as desired.

1. Launch XCTU.
2. Verify that the MO command is set to **6**, which is the default.
3. Configure the frequency of polls for Remote Manager activity using the DF command. The default is 1440 minutes (24 hours).
4. Enable the SM/UDP feature in Remote Manager for each device. See Enable SM/UDP.
5. To ensure that the device is connected to Remote Manager, you must send an SM/UDP request.
   a. Log into Remote Manager.
   b. Click **Devices** in the left pane.
   c. Select the device that you want to work with.
   d. From the right pane, click **Actions** and then **SM/UDP Request Connect**.
   e. If you would like a response, enable **Request Response**.
   f. Click **Request Connect**. When the connection is made, the **Connection Status** icon next to the device on the **Devices** page turns green.

Configure Remote Manager features using automations

Remote Manager provides tools to perform common management and maintenance tasks on your XBee device. Remote Manager automations are a sequence of commands that can be performed on one or more XBee Cellular devices. When an automation is run it becomes an active operation and can be monitored for status and completion.

Note You must upgrade your device to the latest firmware for all features to be available. See Update the firmware.

Some typical examples of useful things that can be done with automations s include:

- Change configuration
- Update your MicroPython application and libraries to add features and capabilities
- Update your security certificates
- Perform a data service device request
- Send an SMS message to your device

Automations can be created and performed through the following methods:

- Remote Manager Automations user interface
- Remote Manager **API Explorer** user interface
- Programming web service calls

Note For any of these methods to work properly, you must have SM/UDP enabled. See Enable SM/UDP.

Overview: Create an automation

When using the most current firmware version, the XBee Cellular devices are designed to poll Remote Manager once per day over the SM/UDP protocol to check for any active operations. In order to perform a set of tasks, the device needs to be told to connect to Remote Manager, perform the sequence of tasks, and then told to disconnect.

The following provides a template of how to create a schedule for an XBee to connect, perform a set of tasks and then disconnect:

1. Make sure that SM/UDP is enabled. See Enable SM/UDP.
2. Log into Remote Manager.
3. Click **Automations**.
4. Click **Create** to launch the wizard.
5. In the **Details** section:
   a. In the **Name** field, enter a descriptive name for the automation, such as "Connect devices".
   b. Click **Save and Continue**.
6. In the **Steps** section:
   a. Click the garbage icon to delete any existing steps.
   b. Click **+** to add a step, and select **SM/UDP Request Connect**.
   c. Add other steps as needed. For examples, refer to the Automation examples section.
   d. Click **+** to add a step, and select **Disconnect**.
   e. Click **Save and Continue**.
7. In the **Targets** section, click **Skip** to skip this section.
8. In the **Triggers** section, click **Skip** to skip this section.
9. Start the automation on a set of devices.
   a. Click **Automations** to show the list of available automations.
   b. Select the automation that you just created.
   c. Click **Action** > **Run Automation**. The **Run Automations** window displays.
   d. Click the **Devices** tab.
   e. Select all of the devices you want to run the automation on.
   f. Click **Confirm** to start the automation.

Automation examples

The examples in the following sections assume you are using the Digi Remote Manager Automations wizard. However, you should be aware that operations can be created and performed programmatically via web service calls or via the API explorer. The XML web service calls provide more options than are available in the GUI dashboard for some tasks.

Example: Read settings and state using Remote Manager

In order to configure devices you will need to know the structure of the XML for your XBee's settings. The easiest way to obtain this is to perform a `query_setting` RCI request against your device.

Note You must upgrade your device to the latest firmware for all features to be available. See Update the firmware.

Note To obtain the state of the device, you can perform the same operations in the example below, but replace `query_setting` with `query_state`.

1. Log into Remote Manager.
2. Click **Automations**.
3. Click **Create** to launch the wizard.
4. In the **Details** section:
   a. In the **Name** field, enter a descriptive name for the automation, such as "Read Settings".
   b. Click **Save and Continue**.
5. In the **Steps** section:
   a. Click the garbage icon to delete any existing steps.
   b. Click **+** to add a step, and select **SM/UDP Request Connect**.
   c. Click **+** again to add another step, and select **RCI**.
      i. In the **RCI Payload** field, enter:

```
<query_setting/>
```

      ii. Enable **Allow Offline**.
   d. Click **+** to add a step, and select **Disconnect**.
   e. Click **Save and Continue**.
6. In the **Targets** section, click **Skip** to skip this section.
7. In the **Triggers** section, click **Skip** to skip this section.
8. Start the automation on a set of devices.
   a. Click **Automations** to show the list of available automations.
   b. Select the automation that you just created.
   c. Click **Action** > **Run Automation**. The **Run Automations** window displays.
   d. Click the **Devices** tab.
   e. Select all of the devices you want to run the automation on.
   f. Click **Confirm** to start the automation.
9. Verify the results of running the automation.
   a. Click **Automations** to show the list of automations.
   b. Click on the name of the automation you just ran to display the status window.
   c. Click the **Runs** tab to see all of the runs for this automation.
   d. Click on the run you are interested in to display a details for each device.
   e. For the device you are interested in, click the **Status** link under the **Summary** column to see more details and the responses.

   If the status was successful, you can to see the response of the RCI query by clicking **Show Details**. This XML structure has the same settings that you will use in the `set_setting` command to configure your XBee as shown in this example: Example: Configure a device from Remote Manager using XML.

Example: Configure a device from Remote Manager using XML

You can configure each XBee device from Remote Manager, using XML. The devices must be in the Remote Manager inventory device list and be active.

Note You must upgrade your device to the latest firmware for all features to be available. See Update the firmware.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

In this configuration example, you are changing the device to poll four times a day instead of just once. In this case, you should change the **DF** parameter to 360 minutes.

1. Log into Remote Manager.
2. Click **Automations**.
3. Click **Create** to launch the wizard.
4. In the **Details** section:
    a. In the **Name** field, enter a descriptive name for the automation, such as "Set Settings".
    b. Click **Save and Continue**.
5. In the **Steps** section:
    a. Click the garbage icon to delete any existing steps.
    b. Click **+** to add a step, and select **SM/UDP Request Connect**.
    c. Click **+** again to add another step, and select **RCI**.
        i. In the **RCI Payload** field, enter:

```
<set_setting>
        <remote_manager>
                <DF>360</DF>
        </remote_manager>
</set_setting>
```

        ii. Enable **Allow Offline**.
    d. Click **+** to add a step, and select **Disconnect**.
    e. Click **Save and Continue**.
6. In the **Targets** section, click **Skip** to skip this section.
7. In the **Triggers** section, click **Skip** to skip this section.
8. Start the automation on a set of devices.
    a. Click **Automations** to show the list of available automations.
    b. Select the automation that you just created.
    c. Click **Action** > **Run Automation**. The **Run Automations** window displays.
    d. Click the **Devices** tab.
    e. Select all of the devices you want to run the automation on.
    f. Click **Confirm** to start the automation.
9. Verify the results of running the automation.

a. Click **Automations** to show the list of automations.
b. Click on the name of the automation you just ran to display the status window.
c. Click the **Runs** tab to see all of the runs for this automation.
d. Click on the run you are interested in to display a details for each device.
e. For the device you are interested in, click the **Status** link under the **Summary** column to see more details and the responses.

Example: Schedule an automation to update the device firmware using Remote Manager

You can use an automation to update the XBee Cellular firmware. Since the device is configured by default to poll Remote Manager once a day, you need to be able to set up a scheduled task to update the device's firmware to take advantage of new features and fixes. To update the firmware to a new version you will need to obtain the .gbl file for the new firmware from our support site. This file is one of the files in the .zip (for example, XBXC-31011.zip) archive that you can download for the product.

Note You must upgrade your device to the latest firmware for all features to be available. See Update the firmware.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

To upgrade using an automation, perform the following steps:

1. Log into Remote Manager.
2. Make sure that you have enabled SM/UDP. See Enable SM/UDP.
3. Click **Automations**.
4. Click **Create** to launch the wizard.
5. In the **Details** section:
   a. In the **Name** field, enter a descriptive name for the automation, such as "Firmware update".
   b. Click **Save and Continue**.
6. In the **Steps** section:
   a. Click the garbage icon to delete any existing steps.
   b. Click **+** to add a step, and select **SM/UDP Request Connect**.
   c. Click **+** again to add another step, and select **Update Firmware**.
      i. From the **Device Type** list box, select the device type.
      ii. From the **Firmware Version** list box, select the version of the firmware to which you want to update the device.
   d. Click **+** to add a step, and select **Disconnect**.
   e. Click **Save and Continue**.
7. In the **Targets** section, click **Skip** to skip this section.
8. In the **Triggers** section, click **Skip** to skip this section.
9. Start the automation on a set of devices.

   a. Click **Automations** to show the list of available automations.
   b. Select the automation that you just created.
   c. Click **Action** > **Run Automation**. The **Run Automations** window displays.
   d. Click the **Devices** tab.
   e. Select all of the devices you want to run the automation on.
   f. Click **Confirm** to start the automation.
10. Verify the results of running the automation.
   a. Click **Automations** to show the list of automations.
   b. Click on the name of the automation you just ran to display the status window.
   c. Click the **Runs** tab to see all of the runs for this automation.
   d. Click on the run you are interested in to display a details for each device.
   e. For the device you are interested in, click the **Status** link under the **Summary** column to see more details and the responses.

Example: Update MicroPython from Remote Manager using an automation

You can create an automation to update the MicroPython application. In this example, you want to add FTP client capability to the MicroPython application. You will need to add the library `uftp.py` and then update the `main.py` application.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

1. Log into Remote Manager.
2. Make sure that SM/UDP is enabled. See Enable SM/UDP.
3. Click **Automations**.
4. Click **Create** to launch the wizard.
5. In the **Details** section:
   a. In the **Name** field, enter a descriptive name for the automation, such as "Update application".
   b. Click **Save and Continue**.
6. In the **Steps** section:
   a. Click the garbage icon to delete any existing steps.
   b. Click **+** to add a step, and select **SM/UDP Request Connect**.
   c. Click **+** again to add another step, and select **RCI**.
      i. In the **RCI Payload** field, enter:

```
<set_setting>
        <remote_manager>
                <MO>7</MO>
        </remote_manager>
</set_setting>
```

      ii. Enable **Allow Offline**.

        iii. From the **On Error** list box, select **Continue**.

        This step disables the MicroPython application so the MicroPython files can be updated, and configures the device to keep the connection open to remote manager.

    d. Click **+** again to add another step, and select **Reboot**.

        i. Enable **Allow Offline**.

        ii. From the **On Error** list box, select **Continue**.

    e. Click **+** again to add another step, and select **Upload Files**.

        i. From the **Choose File** list box, select **main.py** from the FTP sample application.

        ii. In the **Destination File Path** field, enter: **~/MicroPython/main.py**

        iii. Enable **Allow Offline**.

        iv. From the **On Error** list box, select **Continue**.

    f. Click **+** again to add another step, and select **Upload Files**.

        i. From the **Choose File** list box, select the **uftp.py** file from the FTP sample application.

        ii. In the **Destination File Path** field, enter: **~/MicroPython/uftp.py**

        iii. Enable **Allow Offline**.

        iv. From the **On Error** list box, select **Continue**.

    g. Click **+** again to add another step, and select **RCI**.

        i. In the **RCI Payload** field, enter:

```
<set_setting>
        <micrpython>
               <PS>1<//PS>
       </micropython>
       <remote_manager>
                <MO>6</MO>
        </remote_manager>
</set_setting>
```

        ii. Enable **Allow Offline**.

        iii. From the **On Error** list box, select **Continue**.

    h. Click **+** again to add another step, and select **Reboot**.

        i. Enable **Allow Offline**.

        ii. From the **On Error** list box, select **Continue**.

    i. Click **+** to add a step, and select **Disconnect**.

    j. Click **Save and Continue**.

7. In the **Targets** section, click **Skip** to skip this section.
8. In the **Triggers** section, click **Skip** to skip this section.
9. Start the automation on a set of devices.

    a. Click **Automations** to show the list of available automations.

    b. Select the automation that you just created.

    c. Click **Action** > **Run Automation**. The **Run Automations** window displays.

    d. Click the **Devices** tab.

e. Select all of the devices you want to run the automation on.
f. Click **Confirm** to start the automation.

10. Verify the results of running the automation.
    a. Click **Automations** to show the list of automations.
    b. Click on the name of the automation you just ran to display the status window.
    c. Click the **Runs** tab to see all of the runs for this automation.
    d. Click on the run you are interested in to display a details for each device.
    e. For the device you are interested in, click the **Status** link under the **Summary** column to see more details and the responses.

Manage data in Remote Manager

You can view and manage XBee data in Remote Manager.

Review device status information from Remote Manager

You can view address, BLE, cellular, firmware, and I/O sampling status information for a XBee device in Remote Manager. The device must be in the Remote Manager inventory device list and be active.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

1. Set up a persistent connection to connect the device to Remote Manager using one of the following methods:
   - **Remote Manager**: A persistent connection can be set up in Remote Manager. This option should be used when you have many deployed devices and no local access. See Set up a persistent connection to a remote XBee.
   - **XCTU**: This option allows immediate access, and should be used when you have local access, such as when using a development kit or in a lab environment.
2. Log into Remote Manager.
3. Click **Devices**.
4. Select the device that you want to configure.
5. Click **Settings** and expand **Config**.
6. Click on the setting group that has information you want to display. The setting information is related to AT commands. For information about each AT command in the categories, click on the appropriate link below.
   - Addressing
   - Bluetooth
   - Cellular
   - Firmware Version/Information
   - I/O
7. When all changes are complete, disconnect the device from Remote Manager.

Manage secure files in Remote Manager

You can interact with files on the XBee device from Remote Manager, using either the SCI (Server command interface) or in the **File Management** view.

You can securely upload files by appending a hash sign (#) to the end of the file name. After the upload, the hash sign (#) is not retained as part of the file name. For example, you could upload a file named *my-cert.crt* appended with a hash sign (#): *my-cert.crt#*. After the upload is complete, the file is named *my-cert.crt*.

Note Uploading secure files in Remote Manager has the same result as doing an ATFS XPUT locally. See Secure files for more information.

SCI (Server command interface)

You can use the SCI (Server command interface) `file_system` command to securely upload a file. For more information, see the file_system section in the *Digi Remote Manager Programming Guide*.

Device Files view

You can upload and manage files in the Remote Manager **Device Files** view.

1. Log into Remote Manager.
2. Click **Devices**.
3. Select the device for which you want to manage files.
4. Click **Files** to open file management view. From this view you can add or remove files on your device.

Remote Manager reference

Enable SM/UDP

You can use the SM/UDP feature to leverage the very small data footprint of Remote Manager SM protocol over UDP.

Note Battery Operated Mode may be enabled in Digi Remote Manager. Review the Battery Operated Mode section to determine the impact of enabling this mode on SM/UDP.

1. Log into Remote Manager.
2. Click **Devices**.
3. Select the device that you want to configure.
4. Click **Details**.
5. From the **Actions** list box, choose **Configure SM/UDP**.
6. Click **Enable**.

TCP connection

The TCP connection between an XBee and Remote Manager is dependent on the device's firmware version. Options are to query Remote Manager once a day or to maintain a persistent TCP connection.

To determine which connection method is being used, refer to the version listed below.

Module	Upgrade firmware version
XBee 3 LTE-M	11411

- **At or above the listed version**: If your firmware version is at or above the listed version, your device queries Remote Manager only once a day. The device connects to Remote Manager, queries Remote Manager for updates and then receives updates. When the update is complete, the device disconnects from Remote Manager.

  If you upgrade to the new firmware version, it is recommended that you keep the polling frequency low to reduce data usage. In order to upgrade firmware in the future, refer to Example: Schedule an automation to update the device firmware using Remote Manager.

  Note If you wish to restore the persistent connection behavior that was the default in prior firmware versions, see Set up a persistent connection to a remote XBee.

- **Below the listed version**: If your firmware version is below the listed version, a persistent TCP connection is used by default. The device is continually connected to Remote Manager using TCP.

Set up a persistent connection to a remote XBee

The default connectivity to Remote Manager polls once a day using SM/UDP, which means that your XBee will always appear in a disconnected state and will use significantly less data.

If needed, you can set up a persistent connection, where the device is continually connected using TCP. To do this, you will need to set bit 0 of the MO setting. The suggested value for **MO** is **7** to connect securely over TLS, or you can use **1** for no security, which is the legacy value.

You can make the change using one of the following methods:

- **Local access**: If you have local access to the device you can use XCTU to change the **MO** setting back to the former default value.
- **Remote access**: If you only have remote access to your XBee you can change the device to maintain a persistent connection to Remote Manager. To do this you can set up a scheduled operation in Remote Manger for your device, as shown below.

To set up an automation to enable a persistent connection:

1. Log into Remote Manager.
2. Make sure that you have enabled SM/UDP. See Enable SM/UDP.
3. Click **Automations**.
4. Click **Create** to launch the wizard.
5. In the **Details** section:
   a. In the **Name** field, enter a descriptive name for the automation, such as "Enable persistent connection."
   b. Click **Save and continue**.
6. In the **Steps** section:
   a. Click the garbage icon to delete any existing steps.
   b. Click **+** to add a step, and select **SM/UDP Request Connect**.
   c. Click **+** again to add another setp, and select **RCI**.

d. In the **RCI Payload** field, enter:

```
<set_setting>
        <remote_manager>
                <MO>7</MO>
        </remote_manager>
</set_setting>
```

e. Enable **Allow Offline**.

f. Click **Save and continue**.

7. In the **Targets** section, click **Skip** to skip this section.
8. In the **Triggers** section, click **Skip** to skip this section.

Disconnect

The TCP connection remains open and periodic polling occurs until you manually disconnect the TCP connection. After you have disconnected the TCP connection, Remote Manager is no longer updated.

You can disconnect the TCP connection using either of the following methods:

- From the **Devices** page in Remote Manager: See Disconnect a device in the *Digi Remote Manager® User Guide*.
- Using web services in Remote Manager: See Request connect SM/UDP support in the *Digi Remote Manager® Programming Guide*.

Determine the location of the firmware version

You must first determine the location of the firmware version to which you want to update. Digi provides updates by hosting them on an FTP server: **ftp1.digi.com**. If the FTP location is not accessible to your XBee Cellular, such as if you are using a VPN, the files may be retrieved and hosted separately on a server that it can reach.

Firmware is provided in the form of delta images which will migrate the cellular component from a known source to a given target version. You can verify the firmware version level of the cellular component using the MV (Modem Version) AT command. Check documentation and release notes for your XBee Cellular variant to determine the necessary upgrade path for your product.

You will need:

- The FTP hostname or IP address, which for Digi hosted files is: **ftp1.digi.com**
- The port running the FTP server, which is typically 21
- Username. For **ftp1.digi.com**, use: anonymous
- Password. For **ftp1.digi.com**, use your email address.
- Directory path containing update file.
- Update image filename.

Disconnect

The TCP connection remains open and periodic polling occurs until you manually disconnect the TCP connection. After you have disconnected the TCP connection, Remote Manager is no longer updated.

You can disconnect the TCP connection using either of the following methods:

- From the **Devices** page in Remote Manager: See Disconnect a device in the *Digi Remote Manager® User Guide*.
- Using web services in Remote Manager: See Request connect SM/UDP support in the *Digi Remote Manager® Programming Guide*.

Configure XBee settings within Remote Manager

You can configure the device settings to use features with Remote Manager. For more information, see Example: Read settings and state using Remote Manager.

Configure device settings in Remote Manager

You can configure each XBee device from Remote Manager. The devices must be in the Remote Manager inventory device list and be active.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

1. Set up a persistent connection to connect the device to Remote Manager using one of the following methods:
   - **Remote Manager**: A persistent connection can be set up in Remote Manager. This option should be used when you have many deployed devices and no local access. See Set up a persistent connection to a remote XBee.
   - **XCTU**: This option allows immediate access, and should be used when you have local access, such as when using a development kit or in a lab environment. See DO (Device Options) and MO (Remote Manager Options). Both must be enabled.
2. Log into Remote Manager.
3. Click **Devices**.
4. Select the device that you want to configure.
5. Click **Settings** > **Config**.
6. Click on the settings category that you want to configure. The settings in that category appear.
7. Make the desired configuration changes. See AT commands for information about each setting in the categories.
8. As you finish configuring in each setting category, click **Apply** to save the changes. If the changes are valid, Remote Manager writes them to non-volatile memory and applies them.
9. When all changes are complete, disconnect the device from Remote Manager.

Configure Remote Manager keepalive interval

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

Managing the data usage and the keepalive interval is important if you have the MO (Remote Manager Options) command bit 0 set to 1 or if you have enabled the Request connect feature in Remote Manager.

Digi Remote Manager is enabled on the XBee by default and has a 60 second keepalive interval, which can result in excessive cellular data usage, depending on your plan. The K1 and K2 commands can be used to tune the keepalive interval. Your carrier will disconnect an inactive socket automatically if there is no activity, so you need to tune this value based on your carrier's disconnect timeout.

You can further reduce your data usage by periodically duty cycling your Remote Manager connection, either from MicroPython or your host processor. For example, you could enable the Remote Manager connection for 2 hours a day and then disable the connection for 22 hours. Your host processor or MicroPython program would need to keep track of the time to ensure the time interval.

Configure SMS messaging in Remote Manager

You can configure a XBee device to use SMS functionality in Remote Manager. This feature uses a "request connect" operation and asks a device to make a full TCP connection to Remote Manager. For a device with SMS capability this can be significantly lower on latency and data cap consumption, as it does not involve polling.

Each device must be individually configured in Remote Manager to use this feature.

When the device receives an SMS message, it examines the message. If the phone number matches and content contains the correct service ID, it is processed internally rather than being delivered as user data.

By default, the device is configured with **32075** as the Remote Manger phone number and **idgp** as the Remote Manager service ID. If you need an alternate short (domestic) code or a long (international) code, you can re-configure the device using the DP (Remote Manager Phone Number) and RI (Remote Manager Service ID) commands.

Note The SMS provision feature cannot be used. This feature is found by selecting a device and then choosing **More > SMS > Provision**. Attempts to enable this feature are ignored.

---

1. Set up a persistent connection to connect the device to Remote Manager using one of the following methods:
   - **Remote Manager**: A persistent connection can be set up in Remote Manager. This option should be used when you have many deployed devices and no local access. See Set up a persistent connection to a remote XBee.
   - **XCTU**: This option allows immediate access, and should be used when you have local access, such as when using a development kit or in a lab environment. See DO (Device Options) and MO (Remote Manager Options). Both must be enabled.
2. Log in to Remote Manager.
3. Click **Devices**.
4. Select the device that you want to configure.
5. Click **Details**.
6. Click **Action** > **Configure SMS**.
7. In the **Phone Number** field, enter the device's SIM card phone number. You can use the PH (Phone Number) command to discover the device's phone number.
8. Expand the **More Options** section.
   - If you are using SMS in the United States only, make sure the **Server Phone** is **32075** and the **Server Keyword** is **idgp**. These are the default values allowed by the device.
   - If you are using SMS outside of the United States, enter the following:
     - **Server Phone**: 447537431797
     - **Server Keyword**: idgp .

     Because **Server Phone** is not the default used by the device, you must also update the DP (Remote Manager Phone Number) setting and RI (Remote Manager Service

ID) setting on the device so that they match the **Server Phone** and **Server Keyword** settings.

9. Click **Save**.

Device Requests in Remote Manager

You can request to communicate with the XBee Cellular through Remote Manager by using the Data Services Device Request feature. The table below contains the data service target names that you can use to communicate with the XBee Cellular.

For more information on creating a data service device request as an automation step, see Data Service Request step in the *Digi Remote Manager User Guide*.

Device request target name	Description
format	Use the **format** device request to format the XBee module's filesystem. For more information, see Format an XBee module.
micropython	Use the **micropython** device request to communicate with a specified device using MicroPython. For more information, see Use the API Explorer to send Device Requests from the *Digi MicroPython Programming Guide*.

Format an XBee module

You can use the format device request target name to format the XBee module's filesystem. This process removes any previous contents and resets the module to a fresh state.

For best results, you should notify others that you plan to reformat the XBee's filesystem before you initiate a format device request.

When you initiate a format device request, the following occurs:

- The format operation closes all files open by other users of the system, including those that may be in use by a MicroPython application.
- The contents of the filesystem are reset to their initial default state.

Example:

```
<sci_request version="1.0">
   <data_service>
      <targets>
         <device id="Your Device ID here" />
      </targets>
    <requests>
      <device_request target_name="format"/>
    </requests>
   </data_service>
</sci_request>
```

Examples: IOT protocols with transparent mode

The following examples provide some additional scenarios you can use to get familiar with the XBee. If you are interested in using the intelligence built into the XBee, see Get started with MicroPython.

Get started with CoAP

Constrained Application Protocol (CoAP) is based on UDP connection and consumes low power to deliver similar functionality to HTTP. This guide contains information about sending GET, POST, PUT and DELETE operations by using the Coap Protocol with XCTU and Python code working with the XBee Smart Modem and Coapthon library (Python 2.7 only).

The Internet Engineering Task Force describes CoAP as:

> The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation. CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments (source).

CoAP terms

When describing CoAP, we use the following terms:

Term	Meaning
Method	COAP's method action is similar to the HTTP method. This guide discusses the GET, POST, PUT and DELETE methods. With these methods, the XBee Smart Modem can transport data and requests.
URI	URI is a string of characters that identifies a resource served at the server.
Token	A token is an identifier of a message. The client uses the token to verify if the received message is the correct response to its query.
Payload	The message payload is associated with the POST and PUT methods. It specifies the data to be posted or put to the URI resource.
Message ID	The message ID is also an identifier of a message. The client matches the message ID between the response and query.

CoAP quick start example

The following diagram shows the message format for the CoAP protocol; see ISSN: 2070-1721 for details:

<table>
<tr><td colspan="10">0</td><td colspan="10">1</td><td colspan="10">2</td><td colspan="2">3</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="2">Ver</td><td colspan="2">T</td><td colspan="4">TKL</td><td colspan="8">Code</td><td colspan="16">Message ID</td></tr>
<tr><td colspan="32">Token (if any, TKL bytes) ...</td></tr>
<tr><td colspan="32">Options (if any) ...</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="24">Payload (if any) ...</td></tr>
</table>

This is an example GET request:

> 44 01 C4 09 74 65 73 74 B7 65 78 61 6D 70 6C 65

The following table describes the fields in the GET request.

Field	HEX	Bits	Meaning
Ver	44	01	Version 01, which is mandatory here.
T		00	Type 0: confirmable.
TKL		0100	Token length: 4.
Code	01	000 00001	Code: 0.01, which indicates the GET method.
Message ID	C4 09	2 Bytes equal to hex at left	Message ID. The response message will have the same ID. This can help out identification.
Token	74 65 73 74	4 Bytes equal to hex at left	Token. The response message will have the same token. This can help out identification.
Option delta	B7	1011	Delta option: 11 indicates the option data is Uri-Path.
Option length		0111	Delta length: 7 indicates there are 7 bytes of data following as a part of this delta option.
Option value	65 78 61 6D 70 6C 65	7 Bytes equal to hex at left	Example.

Configure the device

1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in Connect the hardware.
2. Open XCTU and click the **Configuration working mode** button.
3. Add the XBee Smart Modem to XCTU; see Add a device to XCTU.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
5. To switch to UDP communication, in the **IP** field, select **0** and click the **Write** button .
6. To set the target IP address that the XBee Smart Modem will talk to, in the **DL** field type **52.43.121.77**and click the **Write** button . A CoAP server is publicly available at address 52.43.121.77.
7. To set the XBee Smart Modem to send data to port 5683 in decimal, in the **DE** field, type **1633** and click the **Write** button.
8. To move into Transparent mode, in the **AP** field, select **0** and click the **Write** button.
9. Wait for the **AI** (Association Indication) value to change to **0** (Connected to the Internet). You can click **Read** to get an update on the **AI** value.

Example: manually perform a CoAP request

Follow the steps in Configure the device prior to this example. This example performs the CoAP GET request:

- Method: GET
- URI: example
- Given message token: test

1. Click the **Consoles working mode** button on the toolbar to add a customized packet.
2. From the AT console, click the **Add new packet button** in the Send packets dialog. The **Add new packet** dialog appears.
3. Click the **HEX** tab and type the name of the data packet: **GET_EXAMPLE**.
4. Copy and past the following text into the **HEX** input tab:
   44 01 C4 09 74 65 73 74 B7 65 78 61 6D 70 6C 65
   This is the CoAP protocol message decomposed by bytes to perform a GET request on an example URI with a token test.
5. Click **Add packet**.
6. Click the **Open** button .
7. Click **Send selected packet**. The message is sent to the public CoAP server configured in Configure the device. A response appears in the Console log. Blue text is the query, red text is the response.

The payload is **Get to uri: example**, which specifies that this is a successful CoAP GET to URI end example, which was specified in the query.

Click the **Close** button to terminate the serial connection.

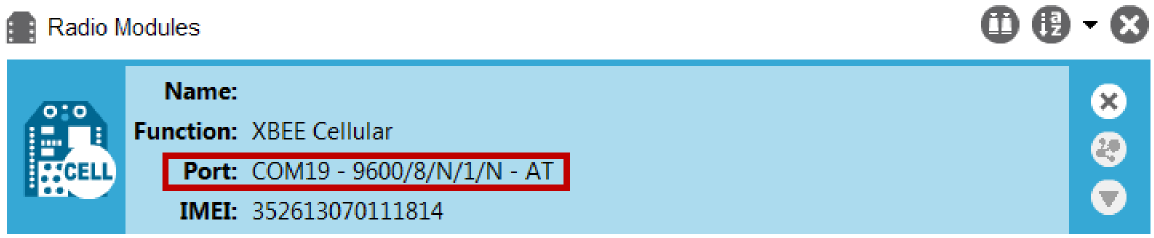
Example: Use Python to generate a CoAP message

This example illustrates how the CoAP protocol can perform GET/POST/PUT/DELETE requests similarly to the HTTP protocol and how to do this using the XBee Smart Modem. In this example, the XBee Smart Modem talks to a CoAP Digi Server. You can use this client code to provide an abstract wrapper to generate a CoAP message that commands the XBee Smart Modem to talk to the remote CoAP server.

Note It is crucial to configure the XBee Smart Modem settings. See Configure the device and follow the steps. You can target the IP address to a different CoAP public server.

1. Install Python 2.7. The Installation guide is located at: python.org/downloads/.
2. Download and install the CoAPthon library in the python environment from pypi.python.org/pypi/CoAPthon.
3. Download these two .txt files: Coap.txt and CoapParser.txt. After you download them, open the files in a text editor and save them as .py files.
4. In the folder that you place the Coap.py and CoapParser.py files, press **Shift** + **right-click** and then click **Open command window**.
5. At the command prompt, type **python Coap.py** and press **Enter** to run the program.
6. Type the USB port number that the XBee Smart Modem is connected to and press **Enter**. Only the port number is required, so if the port is COM19, type 19.

Note If you do not know the port number, open XCTU and look at the XBee Smart Modem in the **Radio Modules** list. This view provides the port number and baud rate, as in the figure below where the baud rate is 9600 b/s.



7. Type the baud rate and press **Enter**. You must match the device's current baud rate.
   XCTU provides the current baud rate in the **BD Baud Rate** field. In this example you would type **9600**.
8. Press **Y** if you want an auto-generated example. Press **Enter** to build your own CoAP request.
9. If you press **Y** it generates a message with:
   - Method: POST
   - URI: example
   - payload: hello world
   - token: test

The send and receive message must match the same token and message id. Otherwise, the client re-attempts the connection by sending out the request.

In the following figure, the payload contains the server response to the query. It shows the results for when you press **Enter** rather than **Y**.

```
C:\Users\jzhang\Desktop\example>python Coap.py
Please enter the serial port number for Xbee: 18
Please enter the baudrate number of Xbee: (9600 or 115200): 9600
Do you want an auto-generated example <Press Y> or build your own <Press ENTER>:

Please enter the HTTP method (GET, POST, PUT, DELETE): PUT
Please enter the uri end path: example
Please enter the payload content. And it cannot be empty: hello world
Please enter the token: digi

##############################################

This is the send out message:
Source: (None, None)
Destination: None
Type: CON
MID: 56045
Code: PUT
Token: digi
Uri-Path: example
Payload:
hello world

This is the received message
Source: (None, None)
Destination: None
Type: ACK
MID: 56045
Code: CHANGED
Token: digi
Payload:
Put hello world to uri: example
```

Get started with MQTT

MQ Telemetry Transport (MQTT) is a messaging protocol that is ideal for the Internet of Things (IoT) due to a light footprint and its use of the publish-subscribe model. In this model, a client connects to a broker, a server machine responsible for receiving all messages, filtering them, and then sending messages to the appropriate clients.

The first two MQTT examples do not involve the XBee Smart Modem. They demonstrate using the MQTT libraries because those libraries are required for Use MQTT over the XBee Cellular Modem with a PC.

The examples in this guide assume:

- Some knowledge of Python.
- An integrated development environment (IDE) such as PyCharm, IDLE or something similar.

The examples require:

- An XBee Smart Modem.
- A compatible development board.
- XCTU. See Install and upgrade XCTU.
- That you install Python on your computer. You can download Python from: https://www.python.org/downloads/.
- That you install the **pyserial** and **paho-mqtt** libraries to the Python environment. If you use Python 2, install these libraries from the command line with **pip install pyserial** and **pip install paho-mqtt**. If you use Python 3, use **pip3 install pyserial** and **pip3 install paho-mqtt**.
- The full MQTT library source code, which includes examples and tests, which is available in the paho-mqtt github repository at https://github.com/eclipse/paho.mqtt.python. To download this repository you must have Git installed.

Example: MQTT connect

This example provides insight into the structure of packets in MQTT as well as the interaction between the client and broker. MQTT uses different packets to accomplish tasks such as connecting, subscribing, and publishing. You can use XCTU to perform a basic example of sending a broker a connect packet and receiving the response from the server, without requiring any coding. This is a good way to see how the client interacts with the broker and what a packet looks like. The following table is an example connect packet:

	Description	Hex value
CONNECT packet fixed header		
byte 1	Control packet type	0x10
byte 2	Remaining length	0x10
CONNECT packet variable header		
Protocol name		
byte 1	Length MSB (0)	0x00

	Description	Hex value
byte 2	Length LSB (4)	0x04
byte 3	(M)	0x4D
byte 4	(Q)	0x51
byte 5	(T)	0x54
byte 6	(T)	0x54
Protocol level		
byte 7	Level (4)	0x04
Connect flags		
byte 8	CONNECT flags byte, see the table below for the bits.	0x02
Keep alive		
byte 9	Keep Alive MSB (0)	0x00
byte 10	Keep Alive LSB (60)	0x3C
Client ID		
byte 11	Length MSB (0)	0x00
byte 12	Length LSB (4)	0x04
byte 13	(D)	0x44
byte 14	(I)	0x49
byte 15	(G)	0x47
byte 16	(I)	0x49

The following table describes the fields in the packet:

Field name	Description
Protocol Name	The connect packet starts with the protocol name, which is MQTT. The length of the protocol name (in bytes) is immediately before the name itself.
Protocol Level	Refers to the version of MQTT in use, in this case a value of 4 indicates MQTT version 3.1.1.
Connect Flags	Indicate certain aspects of the packet. For simplicity, this example only sets the Clean Session flag, which indicates to the client and broker to discard any previous session and start a new one.
Keep Alive	How often the client pings the broker to keep the connection alive; in this example it is set to 60 seconds.

Field name	Description
Client ID	The length of the ID (in bytes) precedes the ID itself. Each client connecting to a broker must have a unique client ID. In the example, the ID is DIGI. When using the Paho MQTT Python libraries, a random alphanumeric ID is generated if you do not specify an ID.

The following table provides the CONNECT flag bits from byte 8, the CONNECT flags byte.

CONNECT Flag Bit(s)	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
User name flag	0							
Password flag		0						
Will retain			0					
Will QoS				0	0			
Will flag						0		
Clean session							1	
Reserved								0

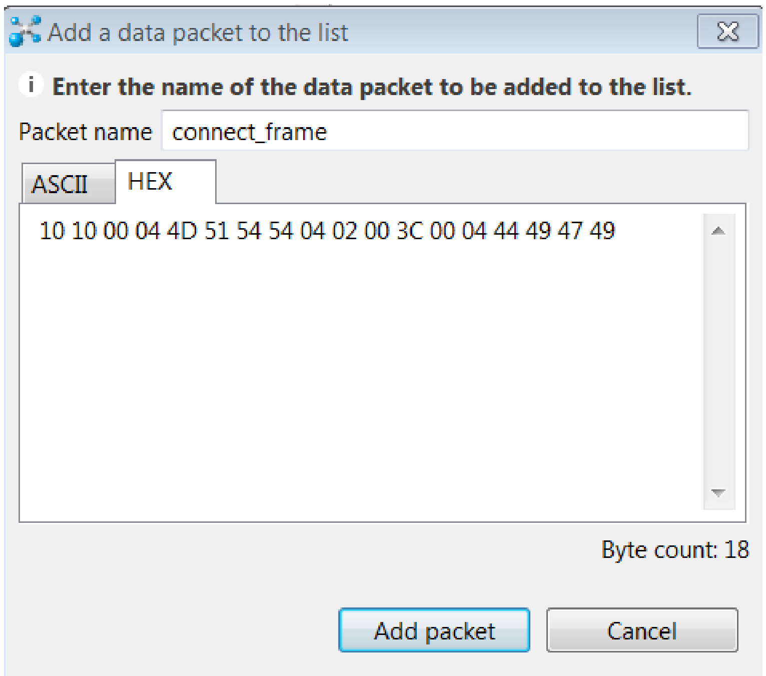
Send a connect packet

Now that you know what a connect packet looks like, you can send a connect packet to a broker and view the response. Open XCTU and click the Configuration working mode button.

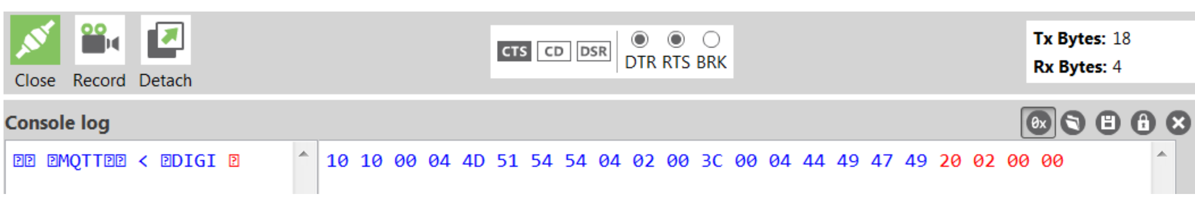
1. Ensure that the device is set up correctly with the SIM card installed and the antennas connected as described in Connect the hardware.
2. Open XCTU and click the **Configuration working mode** button.
3. Add the XBee Smart Modem to XCTU. See Add a device to XCTU.
4. Select a device from the **Radio Modules** list. XCTU displays the current firmware settings for that device.
5. In the **AP** field, set **Transparent Mode** to **[0]** if it is not already and click the **Write** button.
6. In the **DL** field, type the IP address or the fully qualified domain name of the broker you wish to use. This example uses test.mosquitto.org.
7. In the **DE** field, type **75B** and set the port that the broker uses. This example uses **75B**, because the default MQTT port is 1883 (0x75B).
8. Once you have entered the required values, click the **Write** button to write the changes to the XBee Smart Modem.
9. Click the **Consoles working mode** button on the toolbar to open a serial console to the device. For instructions on using the Console, see the AT console topic in the *XCTU User Guide*.
10. Click the **Open** button to open a serial connection to the device.
11. From the AT console, click the **Add new packet button** in the **Send packets** dialog. The **Add new packet** dialog appears.

12. Enter the name of the data packet. Name the packet **connect_frame** or something similar.
13. Click the **HEX** input tab and type the following (these values are the same values from the table in Example: MQTT connect):

    10 10 00 04 4D 51 54 54 04 02 00 3C 00 04 44 49 47 49



14. Click **Add packet**. The new packet appears in the **Send packets** list.
15. Click the packet in the **Send packets** list.
16. Click **Send selected packet**.
17. A CONNACK packet response from the broker appears in the **Console log**. This is a connection acknowledgment; a successful response should look like this:



You can verify the response from the broker as a CONNACK by comparing it to the structure of a CONNACK packet in the MQTT documentation, which is available at http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718081).

Example: send messages (publish) with MQTT

A basic Python example of a node publishing (sending) a message is:

```
mqttc = mqtt.Client("digitest")  # Create instance of client with client ID
"digitest"
mqttc.connect("m2m.eclipse.org", 1883)  # Connect to (broker, port,
keepalive-time)
```

```
mqttc.loop_start()  # Start networking daemon
mqttc.publish("digitest/test1", "Hello, World!")  # Publish message to
"digitest /test1" topic
mqttc.loop_stop()  # Kill networking daemon
```

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.

This example imports the MQTT library, allowing you to use the MQTT protocol via APIs in the library, such as the **connect()**, **subscribe()**, and **publish()** methods.

The second line creates an instance of the client, named **mqttc**. The client ID is the argument you passed in: **digitest** (this is optional).

In line 3, the client connects to a public broker, in this case **m2m.eclipse.org**, on port **1883** (the default MQTT port, or 8883 for MQTT over TLS). There are many publicly available brokers available, you can find a list of them here: https://github.com/mqtt/mqtt.github.io/wiki/brokers.

Line 4 starts the networking daemon with **client.loop_start()** to handle the background network/data tasks.

Finally, the client publishes its message **Hello, World!** to the broker under the topic **digitest/backlog/test1**. Any nodes (devices, phones, computers, even microcontrollers) subscribed to that same topic on the same broker receive the message.

Once no more messages need to be published, the last line stops the network daemon with **client.loop_stop()**.

Example: receive messages (subscribe) with MQTT

This example describes how a client would receive messages from within a specific topic on the broker:

```
import paho.mqtt.client as mqtt


def on_connect(client, userdata, flags, rc):  # The callback for when
    the client connects to the broker print("Connected with result
    code {0}".format(str(rc)))
    # Print result of connection attempt client.subscribe("digitest/test1")
    # Subscribe to the topic "digitest/test1", receive any messages
    published on it


def on_message(client, userdata, msg):  # The callback for when a PUBLISH
    message is received from the server. print("Message received-> "
    + msg.topic + " " + str(msg.payload))  # Print a received msg


client = mqtt.Client("digi_mqtt_test")  # Create instance of client with
client ID "digi_mqtt_test"
client.on_connect = on_connect  # Define callback function for successful
connection
client.on_message = on_message  # Define callback function for receipt of a
message
# client.connect("m2m.eclipse.org", 1883, 60)  # Connect to (broker, port,
keepalive-time)
client.connect('127.0.0.1', 17300)
```

```
client.loop_forever()   # Start networking daemon
```

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.

The first line imports the library functions for MQTT.

The functions **on_connect** and **on_message** are callback functions which are automatically called by the client upon connection to the broker and upon receiving a message, respectively.

The **on_connect** function prints the result of the connection attempt, and performs the subscription. It is wise to do this in the callback function as it guarantees the attempt to subscribe happens only after the client is connected to the broker.

The **on_message** function prints the received message when it comes in, as well as the topic it was published under.

In the body of the code, we:

- Instantiate a client object with the client ID **digi_mqtt_test**.
- Define the callback functions to use upon connection and upon message receipt.
- Connect to an MQTT broker at **m2m.eclipse.org**, on port **1883** (the default MQTT port, or 8883 for MQTT over TLS) with a keepalive of 60 seconds (this is how often the client pings the broker to keep the connection alive).

The last line starts a network daemon that runs in the background and handles data transactions and messages, as well as keeping the socket open, until the script ends.

Use MQTT over the XBee Cellular Modem with a PC

To use this MQTT library over an XBee Smart Modem, you need a basic proxy that transfers a payload received via the MQTT client's socket to the serial or COM port that the XBee Smart Modem is active on, as well as the reverse; transfer of a payload received on the XBee Smart Modem's serial or COM port to the socket of the MQTT client. This is simplest with the XBee Smart Modem in Transparent mode, as it does not require code to parse or create API frames, and not using API frames means there is no need for them to be queued for processing.

1. To put the XBee Cellular Modem in Transparent mode, set **AP** to **0**.
2. Set **DL** to the IP address of the broker you want to use.
3. Set **DE** to the port to use, the default is 1883 (0x75B). This sets the XBee Smart Modem to communicate directly with the broker, and can be performed in XCTU as described in Example: MQTT connect.
4. You can make the proxy with a dual-threaded Python script, a simple version follows:

```
import threading
import serial
import socket


def setup():
    """
    This function sets up the variables needed, including the serial port,
    and it's speed/port settings, listening socket, and localhost adddress.
    """
```

```
    global clisock, cliaddr, svrsock, ser
    # Change this to the COM port your XBee Cellular module is using.  On
    # Linux, this will be /dev/ttyUSB#
    comport = 'COM44'
    # This is the default serial communication speed of the XBee Cellular
    # module
    comspeed = 115200
    buffer_size = 4096  # Default receive size in bytes
    debug_on = 0  # Enables printing of debug messages
    toval = None  # Timeout value for serial port below
    # Serial port object for XBCell modem
    ser = serial.Serial(comport,comspeed,timeout=toval)
    # Listening socket (accepts incoming connection)
    svrsock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    # Allow address reuse on socket (eliminates some restart errors)
    svrsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    clisock = None
    cliaddr = None  # These are first defined before thread creation
    addrtuple = ('127.0.0.1', 17300)  # Address tuple for localhost
    # Binds server socket to localhost (allows client program connection)
    svrsock.bind(addrtuple)
    svrsock.listen(1)  # Allow (1) connection


def ComReaderThread():
    """
    This thread listens on the defined serial port object ('ser') for data
    from the modem, and upon receipt, sends it out to the client over the
    client socket ('clisock').
    """
    global clisock
    while (1):
        resp = ser.read()  ## Read any available data from serial port
        print("Received {} bytes from modem.".format(len(resp)))

        clisock.sendall(resp)  # Send RXd data out on client socket
        print("Sent {} byte payload out socket to client.".format(len
(resp)))


def SockReaderThread():
    """
    This thread listens to the MQTT client's socket and upon receiving a
    payload, it sends this data out on the defined serial port ('ser') to
the
    modem for transmission.
    """

    global clisock
    while (1):
        data = clisock.recv(4096)  # RX data from client socket
        # If the RECV call returns 0 bytes, the socket has closed
        if (len(data) == 0):
            print("ERROR - socket has closed.  Exiting socket reader
thread.")
            return 1  # Exit the thread to avoid a loop of 0-byte receptions
        else:
            print("Received {} bytes from client via socket.".format(len
(data)))
```

```
                print("Sending payload to modem...")
                bytes_wr = ser.write(data)  # Write payload to modem via
UART/serial
                print("Wrote {} bytes to modem".format(bytes_wr))

def main():
    setup()  # Setup the serial port and socket
    global clisock, svrsock
    if (not clisock):  # Accept a connection on 'svrsock' to open 'clisock'
        print("Awaiting ACCEPT on server sock...")
        (clisock,cliaddr) = svrsock.accept()  # Accept an incoming
connection
        print("Connection accepted on socket")
    # Make thread for ComReader
    comthread = threading.Thread(target=ComReaderThread)
    comthread.start()  # Start the thread
    # Make thread for SockReader
    sockthread = threading.Thread(target=SockReaderThread)
    sockthread.start()  # Start the thread

main()
```

Note This script is a general TCP-UART proxy, and can be used for other applications or scripts that use the TCP protocol. Its functionality is not limited to MQTT.

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.

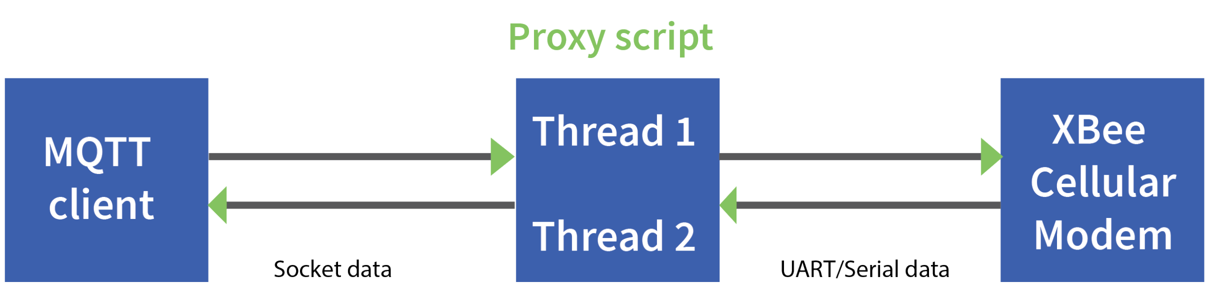
This proxy script waits for an incoming connection on localhost (**127.0.0.1**), on port **17300**. After accepting a connection, and creating a socket for that connection (**clisock**), it creates two threads, one that reads the serial or COM port that the XBee Smart Modem is connected to, and one that reads the socket (**clisock**), that the MQTT client is connected to.

With:

- The proxy script running
- The MQTT client connected to the proxy script via localhost (**127.0.0.1**)
- The XBee Smart Modem connected to the machine via USB and properly powered
- **AP**, **DL**, and **DE** set correctly

the proxy acts as an intermediary between the MQTT client and the XBee Smart Modem, allowing the MQTT client to use the data connection provided by the device.

Think of the proxy script as a translator between the MQTT client and the XBee Smart Modem. The following figure shows the basic operation.



The thread that reads the serial port forwards any data received onward to the client socket, and the thread reading the client socket forwards any data received onward to the serial port. This is represented in the figure above.

The proxy script needs to be running before running an MQTT publish or subscribe script.

1. With the proxy script running, run the subscribe example from Example: receive messages (subscribe) with MQTT, but change the connect line from **client.connect("m2m.eclipse.org", 1883, 60)** to **client.connect("127.0.0.1", port=17300, keepalive=20**). This connects the MQTT client to the proxy script, which in turn connects to a broker via the XBee Smart Modem's internet connection.
2. Run the publish example from Example: send messages (publish) with MQTT in a third Python instance (while the publish script is running you will have three Python scripts running at the same time).

The publish script runs over your computer's normal Internet connection, and does not use the XBee Smart Modem. You are able to see your published message appear in the subscribe script's output once it is received from the broker via the XBee Smart Modem. If you watch the output of the proxy script during this process you can see the receptions and transmissions taking place.

The proxy script must be running before you run the subscribe and publish scripts. If you stop the subscribe script, the socket closes, and the proxy script shows an error. If you try to start the proxy script after starting the subscribe script, you may also see a socket error. To avoid these errors, it is best to start the scripts in the correct order: proxy, then subscribe, then publish.

Update the firmware

You should update your XBee to the latest firmware to take advantage of all the latest fixes and features. Refer to the topics below for information about the available update methods.

Digi strongly recommends that you devise a plan to update the firmware after initial deployment. For more information, see Create a plan for device and cellular component firmware updates.

Create a plan for device and cellular component firmware updates

You should update your XBee to the latest firmware to take advantage of all the latest fixes and features. Changes to the cellular network, security issues, or software bugs may be identified which require firmware updates to resolve. In addition, Digi periodically releases new device firmware which includes new features and improves reliability and performance of existing features. You should evaluate and test the new releases and update your firmware to take advantage of the improvements and new features.

Note Digi will not accept responsibility for customers who have not planned to update their units. Please review the information provided below.

Please review the suggestions below:

- Always test device and any cellular component firmware updates before deploying these updates to units in the field.
- If updates will be performed using a PC, XCTU version 6.5.0 or later is able to perform complete firmware updates on all device cellular modems, including updating the cellular component firmware.
- If updates will be performed using a host processor, see Use a host processor to update the modem firmware for XBee 3 devices over UART prior to *10 or Use a host processor to update the device firmware for XBee 3 devices over UART after *10.
- If updates will be performed over-the-air (OTA):
  - If your XBee application is using API mode, monitor for Modem Status (0x8A) API frames with status codes 0x38 through 0x3A. These modem status frames inform the XBee's host application about ongoing and completed or failed firmware updates.
  - If your XBee application is using Transparent mode, test your application to determine whether it is tolerant to over-the-air firmware updates of the cellular component and XBee firmware. If your application cannot tolerate the network connection being non-functional for up to 30 minutes (for example, if the XBee will be reset in a shorter time than that), do not use over-the-air updates, and be aware that firmware updates to the XBee require

user intervention.

- If the XBee firmware is updated over-the-air using Digi Remote Manager: After the new firmware image has been downloaded and validated, the XBee modem reboots automatically to install the firmware. The XBee then resets into the new firmware once the update is complete, which may take up to 60 seconds.
- If the cellular component firmware is being updated: After the cellular firmware update image has been downloaded, the XBee modem disconnects from the network and the cellular component will be updated. This update will take up to 30 minutes. After the update completes (or fails), the XBee will reconnect to the cellular network automatically.

IMPORTANT

Future cellular component updates may require the use of USB direct mode access.

Ensure your hardware design permits USB Direct functionality, either by designing in a USB port and options for enabling and disabling USB Direct, or by allowing the XBee 3 cellular modem to be removed from its socket and placed on a development board, such as the Digi XBIB-CU-TH.

Update the device and the cellular firmware using XCTU

Use XCTU to update the device firmware, and if needed, XCTU will attempt to update your cellular firmware.

Update the device and cellular firmware using XCTU and USB Direct access

Note Before you begin, make sure you have XCTU installed and the device is added to the utility. See Install and upgrade XCTU.

Update the device and cellular firmware using XCTU and USB Direct access

You can use XCTU to update the device and cellular firmware. XCTU updates the device firmware to the version you select, and then, if needed, XCTU will attempt to update your cellular firmware. Upgrading the cellular component firmware requires USB Direct, which is accessible using an XBIB-CU-TH development board or from your board design.

WARNING! Version 1140F reorganizes the product's flash memory and upgrades the product to version 11410. You cannot downgrade to a version earlier than 11410 after installing 1140F/11410. You also need to use XCTU version 6.4.2 or later.

Prerequisites

- Windows PC
- Digi XCTU version 6.5.6 or newer. You should upgrade XCTU to the latest version.
- The device is added to XCTU. See Add a device to XCTU.
- Digi XBIB-CU-TH development board, or your own hardware which enables USB Direct access
- USB cable for USB Direct access is connected to the PC
- Cellular component USB drivers are installed

To update the device and cellular firmware:

1. Launch XCTU.
2. Click the **Configuration working modes** button.
3. From the **Radio Modules** list, select the device that you want to update.
4. Verify the following configuration. The cellular component firmware update may not work if any of these settings are enabled. Ensure the following:
   - Airplane mode is disabled: ATAM set to 0
   - Bypass mode is disabled: ATAP not 5
   - USB Direct Mode is disabled: ATP1 not 7
5. Click **Update firmware**. The **Update the radio module firmware** dialog appears and displays the available and compatible device firmware for the selected XBee module.
6. Select the product family of the XBee module, the function set, and the latest firmware version for the device.

7. Make sure you check the **Force the module to maintain its current configuration** to ensure you do not lose any changes to your configuration.
8. If desired, you can select the **Force the Cellular modem update** option. When selected, the cellular component is updated even if it is already on the newest firmware version. This step is optional.
9. Click **Update** to update the device firmware.
10. If the cellular component firmware requires an update or if you selected the **Force the Cellular modem update** option, a prompt displays.
11. Click **OK** to continue with the update process. XCTU performs the following:
    - XCTU applies and updates the device firmware.
    - If the cellular firmware is being updated, XCTU reconfigures the XBee for USB Direct access and updates the new cellular firmware on the device.

Update the device firmware

You should update the device firmware on your XBee to the latest version to take advantage of all the latest fixes and features. Security issues or software bugs may be identified which require firmware updates to resolve. In addition, Digi periodically releases new firmware which includes new features and improves reliability and performance of existing features.

- For information about updating the cellular firmware, see Update the cellular firmware.
- For information about using XCTU to update both the device firmware and, if needed, the cellular firmware, see Update the device and the cellular firmware using XCTU.

The table below lists update methods you can use and the instructions for each method.

Method	Instructions
FOTA (DRM)	■ Update the firmware from the Devices page in Remote Manager ■ Update the firmware using web services in Remote Manager ■ Schedule a task to update the device firmware using Remote Manager
API	■ Use a host processor to update the modem firmware for XBee 3 devices over UART prior to *10 ■ Use a host processor to update the modem firmware for XBee 3 devices over UART after *10

Update the firmware from the Devices page in Remote Manager

You can update the device firmware for one or multiple devices from the **Devices** page in Remote Manager.

Before you begin, verify the TCP connection method your device uses to connect to Remote Manager: query once a day or use a persistent TCP connection. See TCP connection.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

WARNING! The firmware version 1140F reorganizes the product's flash memory and upgrades the product to version 11410. You cannot downgrade to a version earlier than 11410 after installing 1140F/11410.

To perform a firmware update:

1. Download the updated firmware file for your device from Digi's support site.
   a. Go to the Digi XBee 3 Cellular LTE-M support page.
   b. Scroll down to the **Firmware Updates** section.

c. Locate and click **Digi XBee 3 Cellular LTE-M/NB-IoT firmware** to download the zip file.
d. Unzip the file. The file contains either a .ebin or a .gbl file.

2. Set up a persistent connection to connect the device to Remote Manager. See Set up a persistent connection to a remote XBee.
3. Log into Remote Manager.
4. Click **Devices** in the left pane.
5. Select the first device you want to update. To select multiple devices (all devices must be of the same type), press the Control key and select additional devices.
6. From the toolbar at the top of the screen, click **Actions**. Scroll down and click **Update Firmware**. The **Update Firmware** dialog appears.
7. Make sure **Update Firmware File** is selected in the list box. This is the default.
8. Click **Choose File** to select the .ebin or .gbl file that you unzipped earlier.
9. Click **Update**. The updated devices automatically reboot when the updates are complete.

   Note The update is immediately rejected and an error is returned if the device is going into sleep mode or is being shut down. See Clean shutdown.

10. When all changes are complete, disconnect the device from Remote Manager.

Update the firmware using web services in Remote Manager

Remote Manager supports both synchronous and asynchronous firmware update using web services. The following examples show how to perform an asynchronous firmware update. See the Remote Manager documentation for more details on firmware updates.

Before you begin, verify the TCP connection method your device uses to connect to Remote Manager: query once a day or use a persistent TCP connection. See TCP connection.

WARNING! The firmware version 1140F reorganizes the product's flash memory and upgrades the product to version 11410. You cannot downgrade to a version earlier than 11410 after installing 1140F/11410.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

1. Download the updated firmware file for your device from Digi's support site.
   a. Go to the Digi XBee 3 Cellular LTE-M\NB-IoT support page.
   b. Scroll down to the **Firmware Updates** section.
   c. Locate and click **Digi XBee 3 Cellular LTE-M/NB-IoT firmware** to download the zip file.
   d. Unzip the file and locate the .gbl file in the unzipped directory.
2. Send an HTTP SCI request to Remote Manager with the contents of the downloaded .gbl file converted to base64 data. Refer to the the following examples:

   Examples for .gbl:

   - Example: Update the XBee .gbl firmware synchronously with Python 3.0
   - Example: Use the device's .gbl firmware image to update the XBee firmware synchronously

Example: Update the XBee .gbl firmware synchronously with Python 3.0

```
import base64
import requests

# Location of firmware image
firmware_path = 'XBXC.gbl'

# Remote Manager device ID of the device being updated
device_id = '00010000-00000000-03526130-70153378'

# Remote Manager username and password
username = "my_Remote_manager_username"
password = "my_remote_manager_password"

url = 'https://remotemanager.digi.com/ws/sci'

# Get firmware image
fw_file = open(firmware_path, 'rb')
fw_data = fw_file.read()
fw_data = base64.encodebytes(fw_data).decode('utf-8')

# Form update_firmware request
data = """
<sci_request version="1.0">
  <update_firmware filename="firmware.gbl">
    <targets>
      <device id="{}"/>
    </targets>
    <data>{}</data>
  </update_firmware>
</sci_request>
""".format(device_id, fw_data)

# Post request
r = requests.post(url, auth=(username, password), data=data)
if (r.status_code != 200) or ("error" in r.content.decode('utf-8')):
    print("firmware update failed")
else:
    print("firmware update success")
```

Example: Use the device's .gbl firmware image to update the XBee firmware synchronously

To update the XBee firmware synchronously with Python 3.0, but using the device firmware image already uploaded to Remote Manager, upload the device's *.gbl firmware to Remote Manager:

1. Download the updated firmware file for your device from Digi's support site. This zip file contains the firmware image.
2. Unzip the file and locate the .gbl file inside the unzipped directory.
3. Log into Remote Manager.
4. Click the arrow next to you your user name, and click **Open Classic Remote Manager**.
5. Click the **Data Services** tab.
6. Click **Data Files**.
7. Click **Upload Files**; browse and select the *.gbl firmware file to upload it.

8. Send an HTTP SCI request to Remote manager with the path of the .gbl file; see the example below.

```
import base64
import requests

# Location of firmware image on Remote Manager
firmware_path = '~/XBXC.gbl'

# Remote Manager device ID of the device being updated
device_id = '00010000-00000000-03526130-70153378'

# Remote Manager username and password
username = "my_remote_manager_username"
password = "my_remote_manager_password"

url = 'https://remotemanager.digi.com/ws/sci'

# Form update_firmware request
data = """
<sci_request version="1.0">
  <update_firmware filename="firmware.gbl">
    <targets>
      <device id="{}"/>
    </targets>
    <file>{}</file>
  </update_firmware>
</sci_request>
""".format(device_id, firmware_path)

# Post request
r = requests.post(url, auth=(username, password), data=data)
if (r.status_code != 200) or ("error" in r.content.decode('utf-8')):
    print("firmware update failed")
else:
    print("firmware update success")
```

Use a host processor to update the modem firmware for XBee 3 devices over UART prior to *10

Use a host processor to update the modem firmware for XBee 3 devices over UART.

This process explains how to update the modem firmware for XBee 3 Cellular devices with a modem firmware version prior to *10 (11410 and 31010).

Update the modem firmware

1. Make sure you have the correct version of the modem firmware for your XBee device.
2. Enter programming (bootloader) mode. Use one of the following methods: AT commands or hardware signaling.
   - **AT commands**
     a. Send the %P command. The %P command must be sent an argument derived from the SL parameter of the module being updated. The argument is the value of SL added to the value 0xDB8A and then masked by performing a bitwise-AND with 0x3FFF.

i. Run ATSL to get the address value, which is in hex.

```
ATSL
123456
```

ii. Add bitwise-AND with 0x3FFF.

```
(0xDB8A + 0x123456) & 0x3FFF= 0x0FE0
```

iii. Send the command AT%PFE0.

```
AT%PFE0
```

b. The **OK** string is returned.

c. After the command is sent, the radio module resets and automatically enters programming mode.

d. Send the FR command to reboot and enter into bootloader.

- **Invoke the bootloader with hardware signaling**

  a. De-Assert RTS (pin 16).

  b. Assert DTR (pin 9).

  c. Put DIN in a low state (break) (pin 3).

  d. Reset the module (pin 5).

  e. Release the break on DIN (pin 3) The module should now be in bootloader at 38400 baud.

3. Once the module is in programming (bootloader) mode, configure the local serial port to 38400/8/N/1.
4. Get the hardware version of the radio module from the bootloader.
   a. Send the V command. The response to that command has the following format:

XXXXYYYYZZAABBBBCCCCCCCCCCCCCCCC	■ **XXXX**: The hardware version. See ATHV, little endian. ■ **YYYY**: The hardware revision. See AT%R, little endian. ■ **ZZ**: The hardware compatibility number. See AT%C. ■ **AA**: Unused and should be 0. ■ **BBBB**: The hardware series. See ATHS, little endian. ■ **CCCCCCCCCCCCCCCC**: The serial number.

5. If possible, change the baud rate of the serial port to optimize the firmware update process. Send the X command to the bootloader.
   - The bootloader answers with the maximum supported baud rate (in ASCII) and, just after that, the bootloader changes its baud rate to that value. Change your baud rate to

match the max supported rate.

- If the bootloader does not answer to this command, remain at the current rate.

6. Send the I command (initialization command). This command erases the current firmware from the device.
7. Transfer the firmware to the device using the transfer protocol shown below.

Transfer the firmware to the device

1. You must split the file into 512 byte blocks.
2. Transfer each block using the following structure, with block index and CRC16 sent in little endian byte:

```
P [2 bytes for block index] [block data with page size length] [2 bytes for CRC16]
```

Note CRC16 is calculated only with the bytes of the page to be sent, and is initialized with 0x0000. The polynomial used for the CRC16 is 0x8005.

3. After each block is transfered, wait for a response. Options are:
   - 0x55 - ACK: This is the expected answer.
   - 0x12: Checksum/CRC16 error.
   - 0x13: Flash write/verify error.

Note If an error occurs, you may try to transfer each block up to three times.

4. Verify and write the firmware to flash.
   a. Send the C command (verify) to verify and write the firmware to the flash.
   b. Verify that the answer to this command is 0x55 (ACK). Any other result is an error.
5. Wait a couple of seconds for the firmware to be installed and start running.

Use a host processor to update the device firmware for XBee 3 devices over UART after *10

This process explains how to update the device firmware for XBee 3 Cellular devices over UART with a firmware version after *10.

Update the modem firmware

1. Make sure you have the correct version of the device firmware for your XBee device.
2. Enter programming (bootloader) mode.
   a. Send the %P command. The **%P** command must be sent an argument derived from the SL parameter of the device being updated. The argument is the value of SL added to the value 0xDB8A and then masked by performing a bitwise-AND with 0x3FFF. For example:
      i. Run ATSL to get the address value, which is in hex.

```
ATSL
123456
```

    ii. Add bitwise-AND with 0x3FFF.

```
(0xDB8A + 0x123456) & 0x3FFF= 0x0FE0
```

    iii. Send the command AT%PFE0.

```
AT%PFE0
```

  b. You will receive a response.
     - If successful, **OK** is returned.
     - If an error occurs, **ERROR** is returned.
  c. After the command is sent, the radio module resets and automatically enters programming mode.
3. Once the device is in programming (bootloader) mode, configure the local serial port to 115200/8/N/1.

Send a firmware image

After invoking the bootloader, a menu is sent out the UART at 115200 baud.

Note If no menu is received after the switch to 115200, send the CR (Carriage Return) command to attempt to receive the prompt again.

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.
4. 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". Note that the previous firmware is maintained, making this error recoverable. 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.

Update the cellular firmware

You should update the cellular firmware on your device to take advantage of all the latest fixes and features.

Note You should also create a plan to update the cellular component firmware on a regular basis, after initial deployment. Security issues or software bugs may be identified which require firmware updates to resolve.

- For information about updating the device firmware, see Update the device firmware.
- For information about using XCTU to update both the device firmware and, if needed, the cellular firmware, see Update the device and the cellular firmware using XCTU.

Method	Instructions
FOTA (DRM)	Update an XBee module cellular component using FOTA
API	Update an XBee module cellular component using API mode (over the wire)
USB	Update the cellular module from a PC using the EasyFlash Firmware Update

Update an XBee module cellular component using FOTA

You can update an XBee 3 Cellular LTE-M/NB-IoT module from A2.00 or A2.01 to L0.00.00.05.08,A.02.04 (maintenance release) using FOTA and the script described in this section.

Prerequisites

- You will need a list of the IMEI values for all XBee 3 Cellular LTE-M/NB-IoT modules to be updated.
- Each module must be running firmware 11413 or later. For update instructions, see Update to the latest firmware from XCTU.
- Each module must have an active, registered SIM card installed.
- Each module must be configured to be connected to Remote Manager and to reconnect when connection is lost. For instructions, see Set up a persistent connection to a remote XBee.
- Each module must have visibility to an FTP server hosting the files.
- The machine that will be performing the update must have Python 3.6 or higher installed and the script must be run using Python 3.

  Note If installing on Windows, ensure that the checkbox for "Add Python to PATH" is checked. If Python is not added to the PATH, you will need to manually specify the directory in which Python is installed in order to proceed.

- The script dependencies must be satisfied by running `pip install --user -r requirements.txt` in the directory containing the script. Creating and using a Python virtual environment with the `virtualenv` command is recommended as a best practice. If using `virtualenv`, install the script dependencies using `pip install -r` `requirements.txt` once the environment is activated. Refer to the instructions in the rest of this section for when to run this command.

Script usage

The arguments and default values for the script are shown below. By default, Digi hosts the update files on a public FTP server and the script attempts to apply the appropriate update files based on intelligence built into the script. The optional arguments should only rarely need to be changed and generally only if the Digi FTP server is not accessible on a private APN, or by direction of Digi Technical Support.

```
usage: fota.py [-h] [--type {auto,es2,ip}] [--ftp_server FTP_SERVER]
               [--ftp_port FTP_PORT] [--ftp_user FTP_USER]
               [--ftp_password FTP_PASSWORD] [--ftp_basedir FTP_BASEDIR]
               <IMEI> [<IMEI> ...]
Perform Over The Air Firmware update (FOTA) on XBee Cellular LTE-M/NB-IoT

positional arguments:
   <IMEI>                 IMEI of device on which to perform update.
optional arguments:
   -h, --help             show this help message and exit
   --type {auto,es2,ip}   Type of modules (default: auto)
   --ftp_server FTP_SERVER
                          FTP server (default: ftp1.digi.com)
   --ftp_port FTP_PORT    FTP port (default: 21)
   --ftp_user FTP_USER    FTP username (default: anonymous)
   --ftp_password FTP_PASSWORD
                          FTP password (default: fota@digi.com)
   --ftp_basedir FTP_BASEDIR
                          FTP base directory (default: /support/ublox)

Script version: 2.0
```

Run the update script

The IMEI values for the modules you want to update are entered as arguments when you run the script. See Script usage for information about the arguments.

1. Download the script.
   a. Go to the Digi XBee 3 Cellular LTE-M support page.
   b. Scroll down to the **Firmware Updates** section.
   c. Locate and click **u-blox Module Over the Air Update Script** to download the zip file: over-the-air.zip.
   d. Unzip the downloaded file.
2. Go to a command line and navigate to the directory containing the script.
3. Run `pip install --user -r requirements.txt` in the directory containing the script.

   Note If using `virtualenv`, run `pip install -r requirements.txt` instead.

4. Run the script. Type: `python fota.py <arguments>`

   where <*arguments*> includes the IMEIs and any optional parameters needed for the environment. For more information, see Script usage.

5. Press **Enter**. You will be prompted for your Remote Manager user name and password.
6. After you have entered the user name and password, the script connects to Remote Manager. For each device listed as an argument in the command line, the script performs the following items:
   - Verifies connectivity to Remote Manager and the module configuration.
   - Applies a sequence of updates to the module that are needed to perform the update.
   - Verifies the final cellular component version.
7. When complete, a CSV is generated and named using this format: `results-<timestamp>.csv`, where `<timestamp>` is the time of the run. The file lists each IMEI, whether the update for that IMEI was successful, and any error message if the update failed.
8. A successful update appears as:

```
SUCCESS updating 352753090812345090861053
```

9. If the update was not successful, try the update again with this flag: `--type=ip`
10. If the update was still not successful, contact Digi Technical Support at **tech.support@digi.com**.

Update an XBee module cellular component using API mode (over the wire)

You can update an XBee 3 Cellular LTE-M/NB-IoT module from the current module cellular component version to a more recent version over the wire, using API mode and the script described in this section.

Prerequisites

- Each module must be inserted into a development board such as the Digi XBIB-CU-TH, powered on, and connected to the PC.
- Each device module must be running firmware 11413 or later. For update instructions, see Update the device and the cellular firmware using XCTU or Update the device firmware.
- Each module must have an active, registered SIM card installed.
- Each module must be configured in the default operating mode:
  - ATAM set to 0 to disable airplane mode
  - ATSM set to 0 to disable sleep
  - ATP0 set to 0 and ATP1 set to 0 to disable direct USB.

  If any of these settings needed to be changed, issue ATWR to save changes.
- Each module must be configured to use unescaped API mode: ATAP set to 1.
- Each module must be configured to use the same baud rate on the UART interface (same ATBD value).
- The machine that will be performing the update must have Python 3.6 or higher installed and the script must be run using Python 3.

Note If installing on Windows, ensure that the checkbox for "Add Python to PATH" is checked. If Python is not added to the PATH, you will need to manually specify the directory in which Python is installed in order to proceed.

- The machine that will be performing the update must have visibility to an HTTP server hosting the files.
- The script dependencies must be satisfied by running `pip install --user -r requirements.txt` in the directory containing the script. Creating and using a Python virtual environment with the `virtualenv` command is recommended as a best practice. If using `virtualenv`, install the script dependencies using `pip install -r requirements.txt` once the environment is activated. Refer to the instructions in the rest of this section for when to run this command.

Script usage

The arguments and default values for the script are shown below. By default, Digi hosts the update files on a public FTP server and the script attempts to apply the appropriate update files based on intelligence built into the script. The optional arguments should only rarely need to be changed and generally only if the Digi FTP server is not accessible on a private APN, or by direction of Digi Technical Support.

```
usage: update.py [-h] [--baud <BAUD>] [--type {auto,es2,ip}]
                 [--remote BASE_URL | --local BASE_DIRECTORY] [-f FILE] [-v]
                 <PORT> [<PORT> ...] [<BAUD>]

Update the u-blox SARA-R410M cellular module used by the Digi XBee 3 Cellular
LTE-M/NB-IoT device to a firmware version using a serial link to one or XBee 3
Cellular devices. Script version: 1.1.

optional arguments:
  -h, --help            show this help message and exit

Serial port:
  Arguments which control how the script interfaces with the XBee

  <PORT>                COM port (or /dev/ttyUSBx device) for the XBee 3
                        Cellular device. Can be specified multiple times to
                        update multiple devices.
  --baud <BAUD>         Current baud rate of the XBee 3 Cellular device(s)
                        (default: 9600)

Firmware update:
  Arguments which control the firmware update process

  --type {auto,es2,ip}  Type of modules (default: auto)
  --remote BASE_URL     HTTP server base path where update files are located
                        (default: http://ftp1.digi.com/support/ublox)
  --local BASE_DIRECTORY
                        Local directory where update files are located
                        (overrides --remote) (default: None)
  -f FILE, --file FILE  Apply only this one file (default: None)
Other arguments:
  -v, --verbose         Also emit log messages about communications with the
                        XBee (default: False)
For additional help, see the XBee 3 Cellular LTE-M/NB-IoT user guide or contact
Digi Technical Support at <tech.support@digi.com>.
```

Example

Examples of usage:

```
C:\Users\admin\over-the-
wire>C:\Users\admin\AppData\Local\Programs\Python\Python37-32\python.exe
update.py COM17
```

Run the update script

The serial port (COM port or `/dev/ttyX` device) values for the modules you want to update are entered as arguments when you run the script. See Script usage for information about the arguments.

1. Identify the source version of the module cellular component using ATMV to identify the source version
2. Download the scripts needed to transition from the source version to the latest version.
   a. Go to the Digi XBee 3 Cellular LTE-M support page.
   b. Scroll down to the **Firmware Updates** section.
   c. Look at the links that begin with **u-blox Over the Wire Update Script**.
   d. Click the appropriate links to download the desired script or scripts. You might need more than one script to reach the most recent version.

   Note You can download more than one zip file into the same directory.

   e. Unzip the downloaded file (or files).

   Note If you unzip more than one downloaded file, make sure to unzip the zip files into different directories.

3. Go to a command line.
4. Navigate to the directory containing the downloaded script.

   Note If you have downloaded more than one script, you should begin with the oldest script first.

5. Run `pip install --user -r requirements.txt` in the directory containing the script.

   Note If using `virtualenv`, run `pip install -r requirements.txt` instead.

6. Verify that you are in the same directory as the script.
7. Run the script. Type: `python update.py <arguments>`

   where <*arguments*> includes the serial port (COM# port or `/dev/tty#` device, where # is the port number) and any optional parameters needed for the environment. For more information, see Script usage.

   Windows example

   ```
   C:\Users\admin\over-the-
   wire>C:\Users\admin\AppData\Local\Programs\Python\Python37-
   32\python.exe update.py COM17
   ```

   Linux examples

```
$ python update.py /dev/tty5
$ python update.py COM10 --baud=115200
```

8. Press **Enter** to begin running the script. For a detailed description of how the update script works, see How the script works.
9. When complete, a `digi-update.log` text file is created, which contains a copy of the log messages emitted to the terminal while the script was running.
10. A successful update appears as:

```
2019-06-27 16:06:28,479 update.py: INFO: Reading cellular component version
string (ATMV)...
2019-06-27 16:06:28,501 update.py: INFO: Update successful. New MV value:
L0.00.00.05.08,A.02.04
2019-06-27 16:06:28,502 update.py: INFO: Cellular component update successful.
2019-06-27 16:06:28,503 update.py: INFO: Restoring previous baud rate and RTS/CTS
configuration.
2019-06-27 16:06:28,743 update.py: INFO: Completed update of XBee 3 Cellular
device with IMEI 352753090861053
2019-06-27 16:06:28,744 update.py: INFO: Updates complete. This script should
automatically exit in a moment. If it does not exit, use Ctrl-C to stop the
script.
```

11. If you have downloaded more than one script, follow the process again, starting with Step 4.

Update the cellular module from a PC using the EasyFlash Firmware Update

A critical issue patch and a maintenance release to update the cellular modem firmware of the XBee 3 Cellular LTE-M module using EasyFlash and the XBIB-CU-TH development board are available.

These updates apply to the XBee 3 Cellular device containing the following cellular modem:

u-blox SARA-R410M-02B

For instructions, see Digi XBee 3 Cellular LTE-M Cellular Module Patch and Maintenance Release via EasyFlash Firmware Update.

Technical specifications

Interface and hardware specifications

The following table provides the interface and hardware specifications for the device.

Specification	Value
Dimensions	24.38 mm x 32.94 mm (0.960 x 1.297 in)
Weight	5 g (0.18 oz)
Operating temperature	-40 to +85 °C
Antenna connector	Cellular: U.FL Bluetooth: U.FL
Digital I/O	13 I/O lines, I^2C
ADC	4 10-bit analog inputs
Analog input voltage range	0 - 2.5 V
Cellular chipset	u-blox SARA-R410M-028
Form factor	Digi XBee 20-pin through-hole
SIM size	4FF Nano

Cellular RF characteristics

The following table provides the RF characteristics for the device.

Specification	Value
Transmit power	Up to 23 dBm, Power Class 3
Receive sensitivity	-105 dBm

Bluetooth RF characteristics

The following table provides the Bluetooth RF characteristics for the device.

Specification	Value
Transmit power	Up to 8 dBm
Receive sensitivity, 1 Mb/s data rate	-92 dBm
Receive sensitivity, 2 Mb/s data rate	-88 dBm
Operating frequency band	ISM 2.4 - 2.4835 GHz

Cellular networking specifications

The following table provides the networking and carrier specifications for the device.

Specification	Value
Carrier and technology	AT&T and Verizon LTE-M T-Mobile NB-IoT in US Vodafone and Deutsche Telekom NB-IoT in Europe Compatible with other LTE-M carriers, see supported bands
Supported bands	LTE FDD bands: ■ Band 12 (700 Mhz) ■ Band 28 (700 MHz) ■ Band 13 (700 MHz) ■ Band 20 (800 MHz) ■ Band 26 (850 MHz) ■ Band 18 (850 MHz) ■ Band 5 (850 MHz) ■ Band 19 (850 MHz) ■ Band 8 (900 MHz) ■ Band 4 (1700 MHz) ■ Band 3 (1800 Mhz) ■ Band 2 (1900 MHz) ■ Band 25 (1900 MHz)[1] ■ Band 1 (2100 MHz) LTE TDD bands: ■ Band 39 (1900 MHz)
Security	Digi Trustfence™

[1]Band 25 is only supported by products containing SARA-R410M-02B-01 and newer. Band 25 is for LTE CAT M1 only.

Specification	Value
Downlink/uplink speeds	LTE M1 ■ up to 300 kb/s DL ■ up to 375 kb/s UL LTE NB1 ■ up to 27.2 kb/s DL ■ up to 62.5 kb/s UL
Duplex mode	Half-duplex
Addressing options	SMS and IP-based protocols may not be available. Check with your carrier's specifications for LTE-M\NB-IoT.

Power requirements

The following table provides the power requirements for the device.

Specification	Value
Supply voltage	3.3 to 4.3 V

Power consumption

Specification	State	Using serial interface VCC = 3.3 V (at room temperature)	Using USB direct mode VCC = 3.3 V (at room temperature)
Peak transmit current	Bluetooth disabled	550 mA	550 mA
	Bluetooth enabled	610 mA	610 mA
Average connected mode current	TX/RX @ 23 dBm	205 mA	220 mA
Active mode current	Idle/connected, listening	20 mA	35 mA
Power save mode current		20 µA	Not supported
Deep sleep current		10 µA	10 µA (USB direct mode must be disabled during sleep)

Electrical specifications

The following table provides the electrical specifications for the XBee Smart Modem.

Symbol	Parameter	Condition	Min	Typical	Max	Units
VCCMAX	Maximum limits of VCC line		0		4.3	V
VDD_IO	Internal supply voltage for I/O		(VCC - 0.15 V) or 3.3 V, whichever is lower	VCC or 3.3 V, whichever is lower	3.3	V
VI	Other XBee pins		-0.3		VDD_IO + 0.3	V
	Voltage on XBee pin 6 (5 V tolerant)		-0.3	5.25 or VDD_IO+2, whichever is lower[1]		
VIL	Input low voltage				0.3*VDD_IO	V
VIH	Input high voltage		0.7*VDD_IO			V
VOL	Voltage output low	Sinking 3 mA, VCC = 3.3 V			0.2*VDD_IO	V
VOH	Voltage output high	Sourcing 3 mA, VCC = 3.3 V	0.8*VDD_IO			V
I_IN	Input leakage current	High Z state I/O connected to Ground or VDD_IO		0.1	30	nA
RPU	Internal pull-up resistor	Enabled		40		kΩ
RPD	Internal pull-down resistor	Enabled		40		kΩ

[1]Pin 6 is 5 V tolerant even when the XBee Smart Modem is not powered. We recommend only driving this pin with 3.3 V for compatibility with other XBee products.

Regulatory approvals

The following table provides the regulatory and carrier approvals for the device.

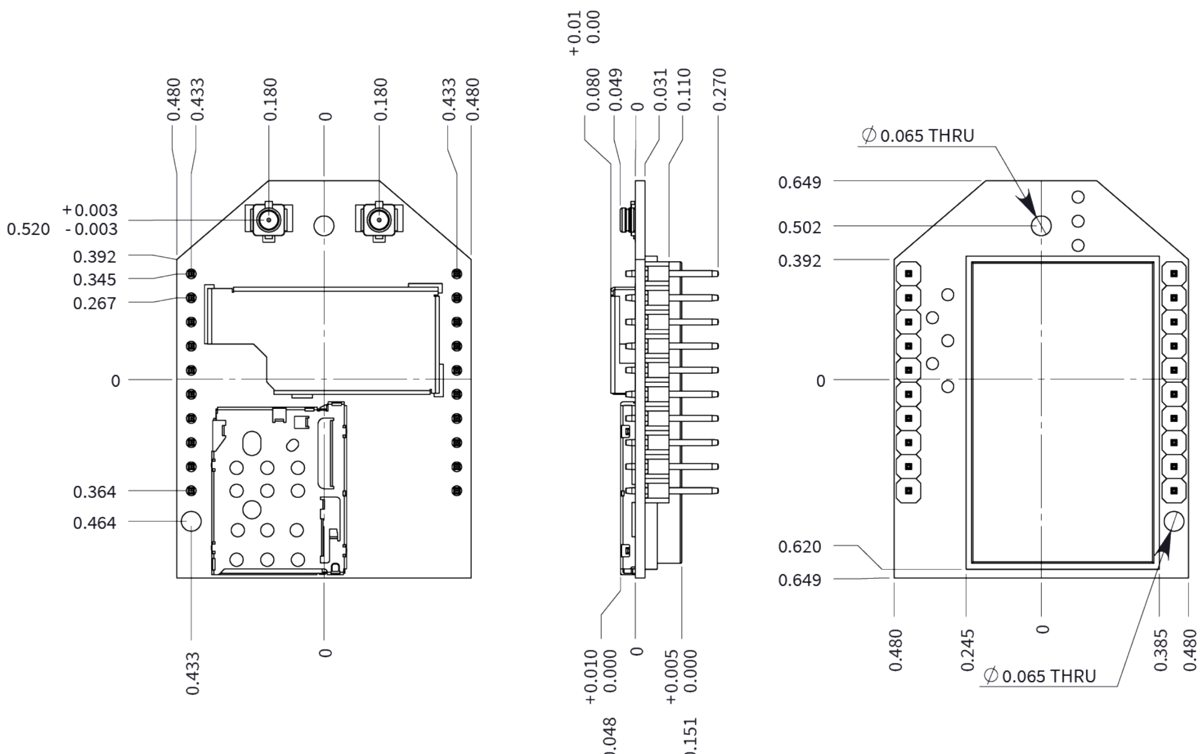
Specification	Value
Model	XB3M1
United States	FCC ID: MCQ-XB3M1 FCC ID: XPY2AGQN4NNN
Innovation, Science and Economic Development Canada (ISED)	IC: 1846A-XB3M1 IC: 8595A-2AGQN4NNN
RoHS	Lead-free and RoHS compliant
AT&T end-device certified	Complete
Verizon end-device certified	Complete
PTCRB	Complete
Bluetooth	Declaration ID: D042514 QDID: 121268

Hardware

Mechanical drawings

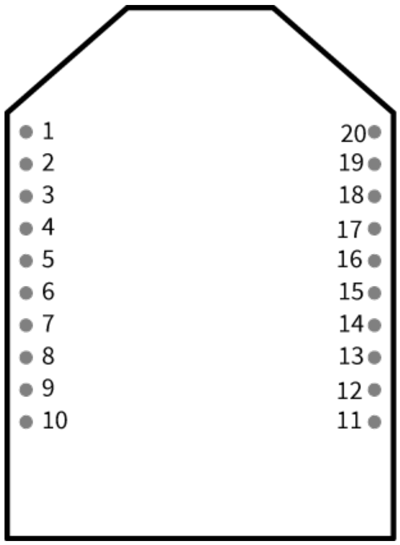
The following figures show the mechanical drawings for the XBee Smart Modem. All dimensions are in inches.

For XBee header information, see XBee header connector requirements.



Pin signals

The pin locations are:



The following table shows the pin assignments for the through-hole device. In the table, low-asserted signals have a horizontal line above signal name.

Pin	Name	Direction	Default	Description
1	V_{CC}			Power supply
2	DOUT	Output	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$	Input	Input	UART Data In
4	DIO12 / SPI_MISO / RX2	Either	Disabled	Digital I/O 12 / SPI Slave Output line / Receive (RX2) Input See the secondary UART note below for more information.
5	$\overline{\text{RESET}}$	Input		
6	PWM0 / RSSI / DIO10/USB_VBUS	Either	Output	PWM Output 0 / RX Signal Strength Indicator / Digital I/O 10
7	PWM1 / DIO11/USB D+ / I2C_SDA	Either	Disabled	PWM Output 1 / Digital I/O 11 or USB Direct D+ line / I2C SDA See the I2C note below for more information.
8	USB D-			USB Direct D- line
9	$\overline{\text{DTR}}$ / SLEEP_RQ/ DIO8	Either	Disabled	Pin Sleep Control Line or Digital I/O 8
10	GND			Ground

Pin	Name	Direction	Default	Description
11	DIO4 / SPI_MOSI / TX2	Either	Disabled	Digital I/O 4 or SPI Slave Input Line / Transmit (TX2) Output See the secondary UART note below for more information.
12	CTS / DIO7	Either	Output	Output Clear-to-Send Flow Control or Digital I/O 7
13	ON /SLEEP/DIO9	Output	Output	Module Status Indicator or Digital I/O 9
14	VREF	-		Feature not supported on this device. Used on other XBee devices for analog voltage reference.
15	Associate / DIO5	Either	Output	Associated Indicator, Digital I/O 5
16	RTS / DIO6	Either	Disabled	Input Request-to-Send Flow Control, Digital I/O 6
17	AD3 / DIO3 / SPI_SS / CTS2	Either	Disabled	Analog Input 3 / Digital I/O 3 / SPI low enabled select line / Clear to Send (CTS2) Input See the secondary UART note below for more information.
18	AD2 / DIO2 / SPI_CLK / RTS2	Either	Disabled	Analog Input 2 or Digital I/O 2, SPI Clock line / Ready to Receive (RTS2) Output See the secondary UART note below for more information.
19	AD1 / DIO1 / SPI_ATTN / I2C_SCL	Either	Disabled	Analog Input 1 or Digital I/O 1, SPI Attention line output / I2C SCL See the I^2C note below for more information
20	AD0 / DIO0	Either	Input	Analog Input 0, Digital I/O 0

Note **Secondary UART**: TX2, RX2, RTS2, and CTS2 (pins 4, 11, 18, and 17) may optionally be configured as a secondary UART serial port using MicroPython. See Class UART in the *Digi MicroPython Programming Guide* and for details.

Note **Class I^2C**: For more information, see Class I^2C in the *Digi MicroPython Programming Guide*.

Pin connection recommendations

To ensure compatibility with future updates, make USB D+ and D- (pin 7 and pin 8) available in your design.

The recommended minimum pin connections are VCC, GND, DIN, DOUT, RTS, DTR and RESET. Firmware updates require access to these pins.

XBee header connector requirements

The XBee header connectors require the following attributes:

- female
- 2 mm pitch
- 10 positions
- single row

RSSI PWM

The RSSI/PWM output is enabled continuously unlike other XBee products where the output is enabled for a short period of time after each received transmission. If running on the XBIB development board, DIO10 is connected to the RSSI LEDs, which may be interpreted as follows:

PWM duty cycle	Number of LEDs turned on	Received signal strength (dBm)
79.39% or more	3	-83 dBm or higher
62.42% to 79.39%	2	-93 to -83 dBm
45.45% to 62.42%	1	-103 to -93 dBm
Less than 45.45%	0	Less than -103 dBm, or no cellular network connection

SIM card

The XBee Smart Modem uses a 4FF nano-SIM card. The SIM interface supports only 1.8 V SIM types.

CAUTION! Never remove the SIM card while the power is on!

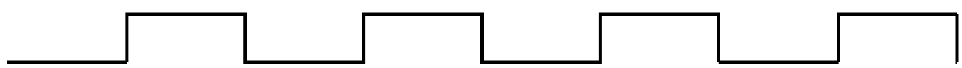
Associate LED functionality

The following table describes the Associate LED functionality. For the location of the Associate LED on the XBIB-U development board, see number 6 on the XBIB-U-DEV reference.

LED status	Blink timing	Meaning
On, solid		Not joined to a mobile network.

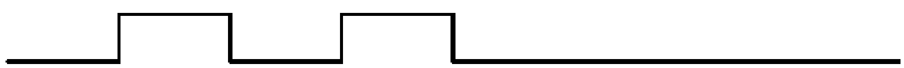
LED status	Blink timing	Meaning
Double blink	½ second	The last TCP/UDP/SMS attempt failed. If the LED has this pattern, you may need to check DI (Remote Manager Indicator) or CI (Protocol/Connection Indication) for the cause of the error. **Note** This pattern applies only to the Transparent mode. Other transmission modes do not affect the Associate LED blink pattern.
Standard single blink	1 second	Normal operation.

The normal association LED signal alternates evenly between high and low as shown below:



Where the low signal means LED off and the high signal means LED on.

When **CI** is not **0** or **0xFF**, the Associate LED has a different blink pattern that looks like this:

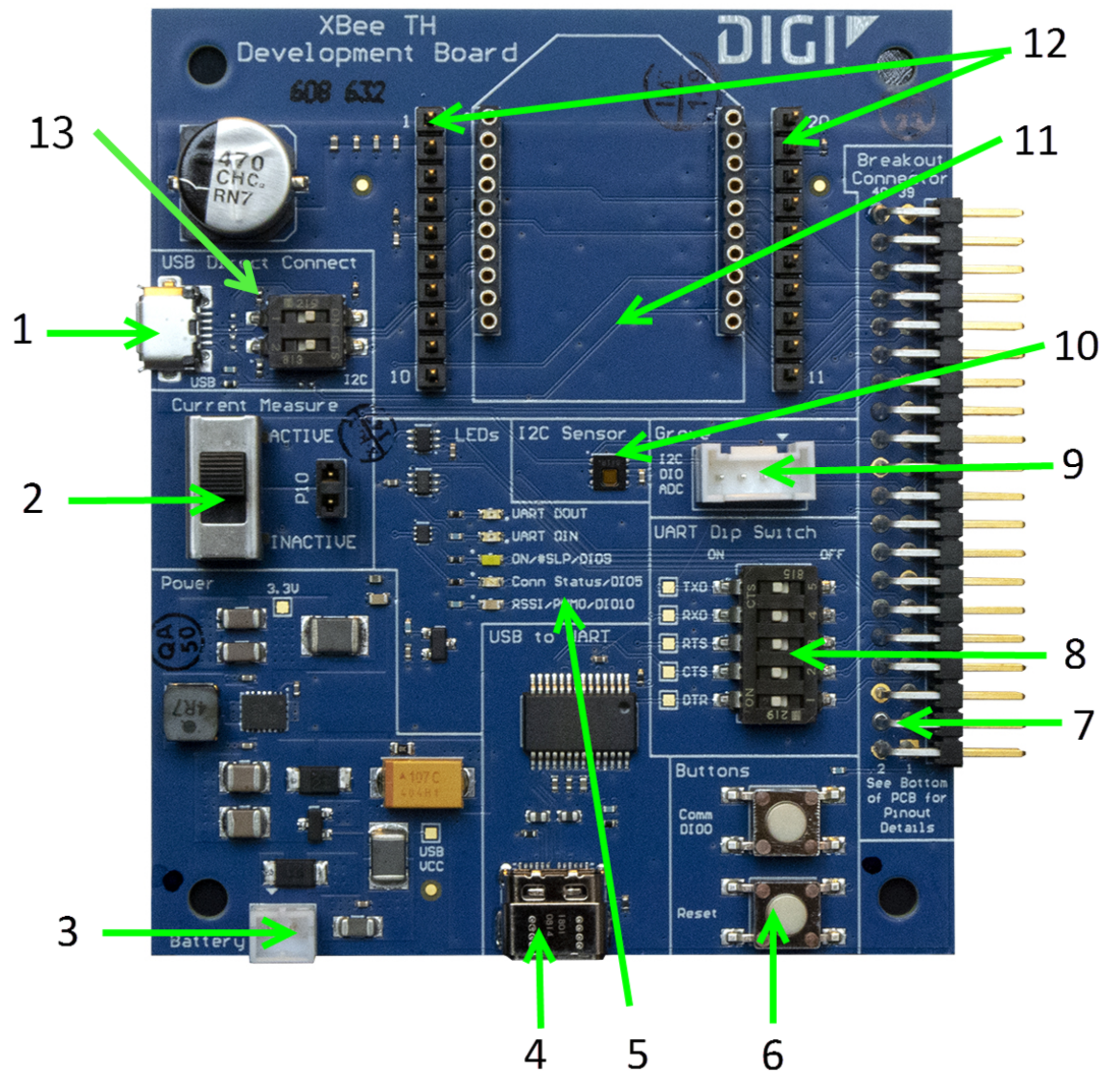


Development boards

XBIB-CU-TH reference

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

Note This module is sold separately or in our XBee3 Cellular Kits.


XBee TH
Development Board
DIGI
12
13
11
1
10
2
9
8
7
3
4
5
6
USB Direct Connect
Current Measure
ACTIVE
INACTIVE
P10
LEDs
I2C Sensor
UART DOUT
UART DIN
ON/SLP/DIO9
Conn Status/DIO5
RSSI/PWM0/DIO10
UART Dip Switch
ON
OFF
TXD
RXD
RTS
CTS
DTR
Power
3.3V
USB to UART
Buttons
Comm
DIO0
Reset
Battery
USB
VCC
Breakout
Connector
See Bottom
of PCB for
Pinout
Details

Number	Item	Description
1	USB Direct Connect (USB MICRO B) and DIP Switch	The USB Direct connector allows for direct connection to the cellular module on the XBee. This connection is the fastest method of upgrading the cellular modem firmware and allows for development of applications that directly interface to the cellular modem. The USB Direct connector is always connected to the pins 7 and 8 of the XBee module. For reliable USB communication to the cellular modem, the DIP switches must be in the OFF (left) position, which disconnects pins 7 and 8 of the XBee module from the breakout header and from the I^2C bus. To use I^2C, the DIP switches must be in the ON (right) position and the USB micro B cable should be disconnected. **Note** This USB connector will not power device. Use USB-C to power the device. **WARNING!** If the DIP switches are left ON while making a USB Direct connection, the cellular modem may enumerate on a computer, but communication to the modem will be unreliable. **WARNING!** USB Direct port should not be connected when used with XBees that do not support USB communications.
2	Current Measure	The switch allows the XBee VCC pin to disconnected from the 3.3V supplied by the XBIB. When in the INACTIVE (downward) position, the XBIB powers the XBee normally. When in the ACTIVE position, power must be delivered via jumper P10. This allows current measurements to be conducted by attaching a current meter across the jumper P10. **Note** The USB-to-serial communications connection may affect this current measurement.

Number	Item	Description
3	Battery Connector	If desired, a battery or other power source can be attached to provide power to the development board. The voltage can range from 2 V to 5.5 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. **WARNING!** Battery current discharge rating must be enough to support 5 W or more. **WARNING!** There is no circuit to prevent over discharge of battery. Battery must contain its own protection circuitry. **WARNING!** Move UART switches to the OFF position when using battery or external power or for when the XBee and the USB-C connector is not powering the UART. The USB to UART converter is powered only via the USB-C connection. **Note** While the battery voltage can vary from 2V to 5V, the XBIB-CU-TH will regulate that voltage to 3.3V for the XBee. Lower input voltages will require higher input currents to supply the necessary power to the XBee and any attached devices.
4	USB-C Connector	Provides power for the XBee and development board as well as serial communications to the XBee. **Note** To run XBee Cellular modules requires connecting this to a USB 3.0 capable port (usually a blue port) due to the power requirements. Connecting to a USB 2.0 port will result in unreliable operation.
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

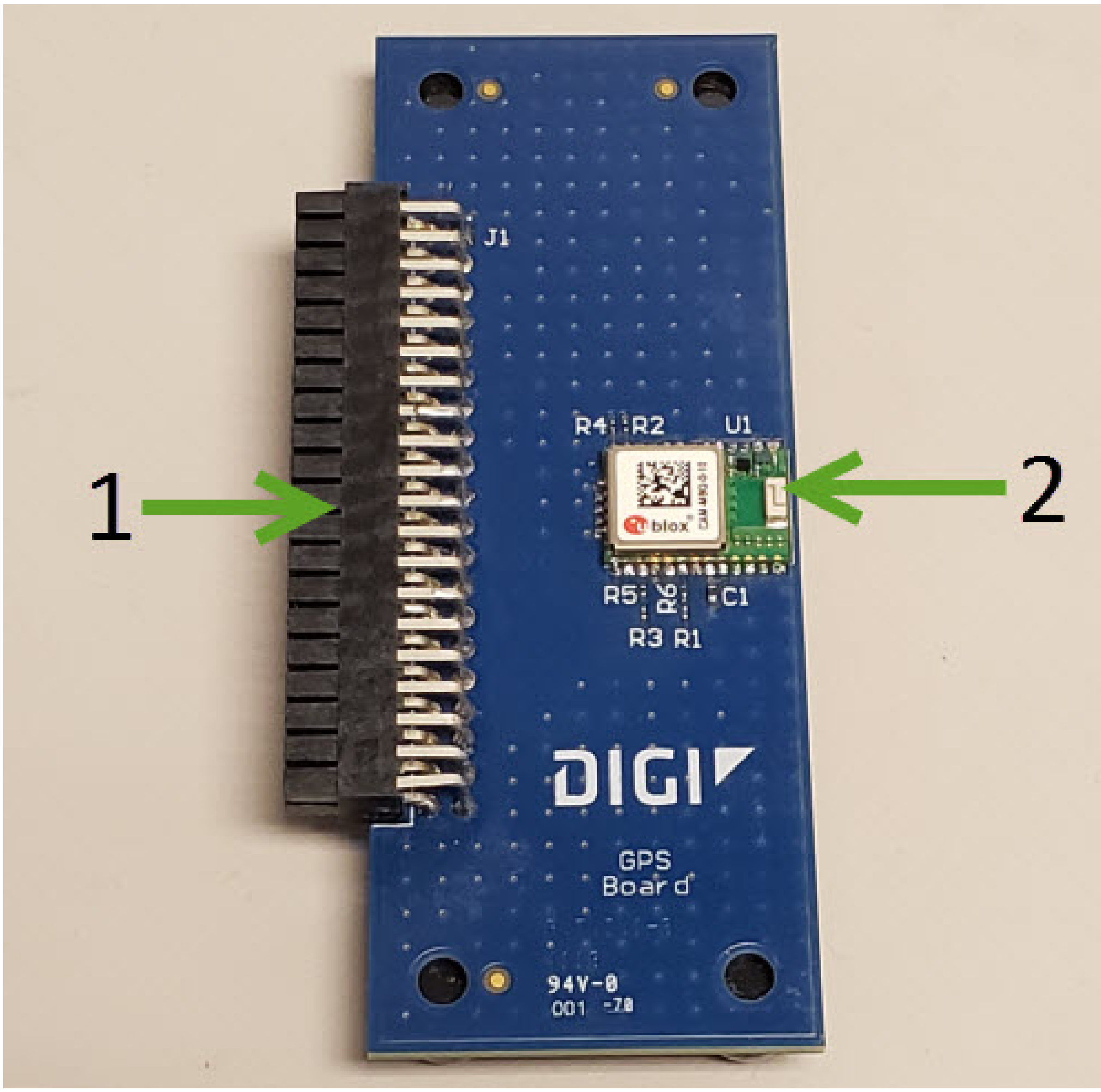
Number	Item	Description
6	User Buttons	Comm DIO0 Button connects the Commissioning/DIO0 pin to GND when pressed. **Note** The XBee Cellular does not implement any commissioning function like other XBees. Connection to the cellular network is automatic when a SIM card is inserted and the modem is powered on. $\overline{\text{RESET}}$ button resets the XBee module when pressed.
7	Breakout Connector	This 40 pin connects to various XBee pins as shown on the silkscreen on the bottom of the board. See XBIB-C Development Boards for details.
8	UART Dip Switch	Push DIP switches to the right (OFF position) to disconnect the XBee from the USB-to-serial converter. The USB-to-serial converter should be disconnected from the XBee to use the serial lines on the breakout connector, when taking current measurements, or when powering the XBIB from the Battery Connector.
9	Grove Connector	This connector attaches I^{2}C-enabled devices to the development board. The XBee3 devices all include I^{2}C. Move both USB direct connect switches to the right (closed position) and disconnect the USB micro port for correct operation of the I^{2}C to connector. ■ Pin 1: I^{2}C_CLK/XBee DIO1 ■ Pin 2: I^{2}C_SDA/XBee DIO11 ■ Pin 3: VCC ■ Pin 4: GND
10	Temp/Humidity Sensor	This part is a Texas Instruments HDC1080 temperature and humidity sensor connected through I^{2}C on XBee pins DIO1 and DIO11. For correct operation of the I^{2}C sensor, both USB direct connect switches must be to the right (closed position) and be sure to disconnect the USB micro port.
11	XBee Socket	This is the socket for the XBee (TH form factor).
12	XBee Test Point Pins	Allows easy access to pins 1 to 20 of the XBee.
13	Switches	The switch position varies, depending on the feature you are using. ■ USB direct mode: Both switches must be in the left position. For more information, see Connect the hardware for USB Direct mode. ■ I^{2}C sensor: Both switches must be in the right position. See item 10, Temp/Humidity Sensor.

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 GPS module is sold separately and requires an XBIB-CU-TH development board.

Note You run a demo using MicroPython to parse UART to GPS communications. See Run the MicroPython GPS demo.

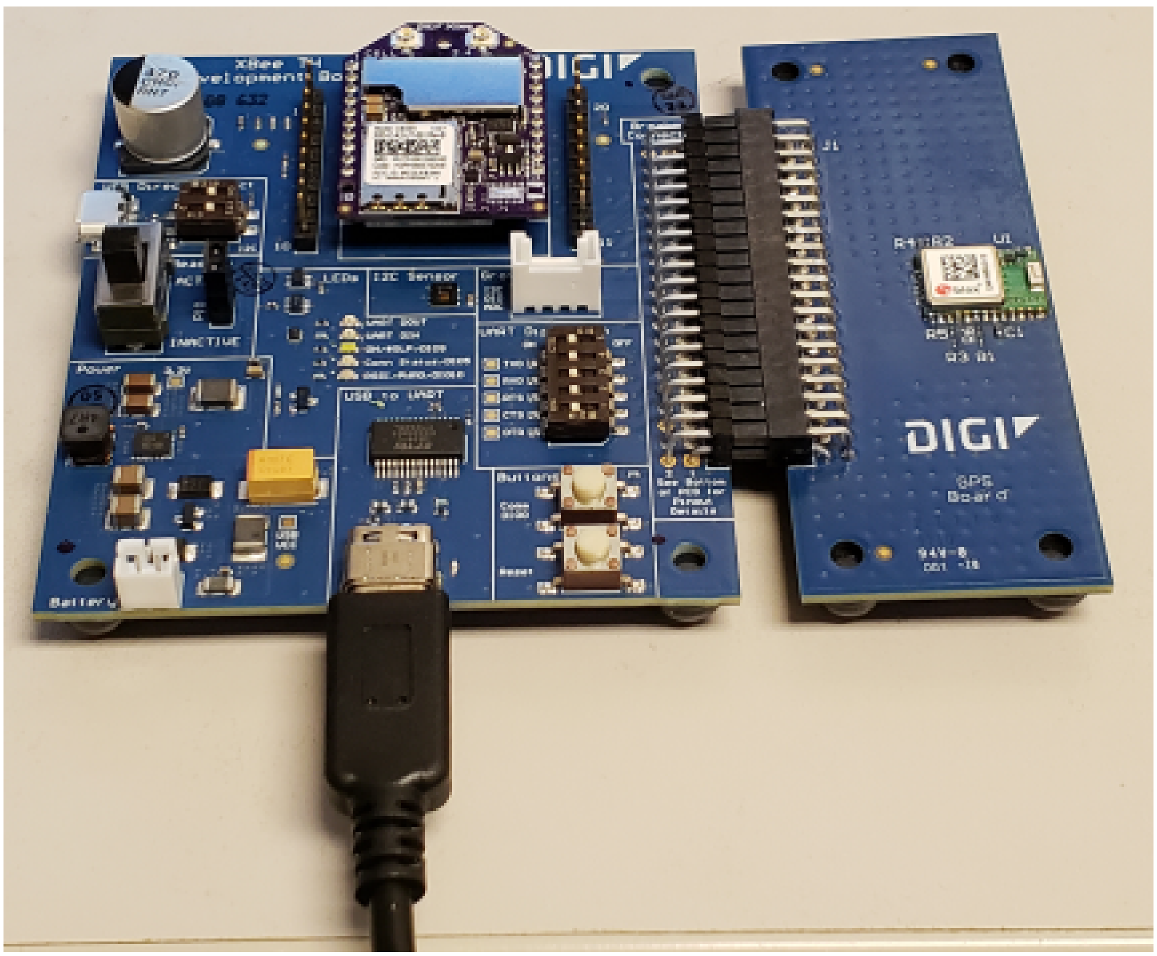


Number	Item	Description
1	40-pin header	This header is used to connect the XBIB-C-GPS board to a compatible XBIB development board. Insert the XBIB-C-GPS module slowly with alternating pressure on the upper and lower parts of the connector. Do not bend pins during insertion or removal process.
2	GPS unit	Contains GPS module CAM-M8Q-0-10. Proper orientation is with the board laying flat, with the GPS module having a clear view of the sky.

Interface with the XBIB-C-GPS module

The XBee Smart Modem can interface with the XBIB-C-GPS board through the 40-pin header. This header is designed to fit into XBIB-C development board. This allows the XBee Smart Modem in the XBIB-C board to communicate with the XBIB-C-GPS board—provided the XBee device 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 I^{2}C 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 Smart Modem.

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

UART (RX and TX) are pins connected from the XBIB-C-GPS to the host board by the 40-pin header. By default, the UART on the XBIB-C-GPS board is active and sends GPS readings once every second. The baud rate of the UART is 9600 baud.

For more information about using Micro Python to communicate to the XBIB-C-GPS module, see Class UART.

Run the MicroPython GPS demo

The Digi MicroPython github repository contains a GPS demo program that parses the GPS NMEA data from the UART and prints them.

Note If you are unfamiliar with MicroPython on XBee, see Get started with MicroPython. For more detailed information, refer to the *Digi MicroPython Programming Guide*.

Step 1: Clone or download the XBee MicroPython repository

1. Navigate to: **https://github.com/digidotcom/xbee-micropython/**
2. 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 2: Edit the MicroPython file

1. Navigate to the location that you created in Step 1.
2. Navigate to: **samples/gps_uart**
3. Open the MicroPython file: *main.py*

Step 3: Run the program

1. Copy the file onto your device's root filesystem directory.
2. Open XCTU and use the MicroPython Terminal to run the demo.
3. Type <CTRL>-R from the MicroPython prompt to run the code.

Antenna recommendations

For additional antenna regulatory requirements, refer to:

- Antenna regulatory information: FCC and ISED

Antenna placement

Antenna location is important for optimal performance. The following suggestions help you achieve optimal antenna performance. See Regulatory Information for details on cellular and Bluetooth antennas that you may use with the XBee Smart Modem.

Keep the antenna(s) as far away from metal objects and other electronics (including the XBee Smart Modem) as possible. Metal objects near the antenna cause parasitic coupling and detuning, preventing the antenna from radiating efficiently. Metal objects between the transmitter and receiver can also block the radiation path or reduce the transmission distance. Some objects that are often overlooked are:

- Metal poles
- Metal studs or beams in structures
- Concrete (reinforced with metal rods)
- Metal enclosures
- Vehicles
- Elevators
- Ventilation ducts
- Batteries
- Tall electrolytic capacitors

Often, small antennas are desirable, but may come at the cost of reduced range and efficiency.

Bluetooth and cellular antennas

If you implement the Bluetooth interface, ensure that the Bluetooth and cellular antennas are at least 3 inches apart (6 inches recommended) to prevent cellular sensitivity from being degraded.

Design recommendations

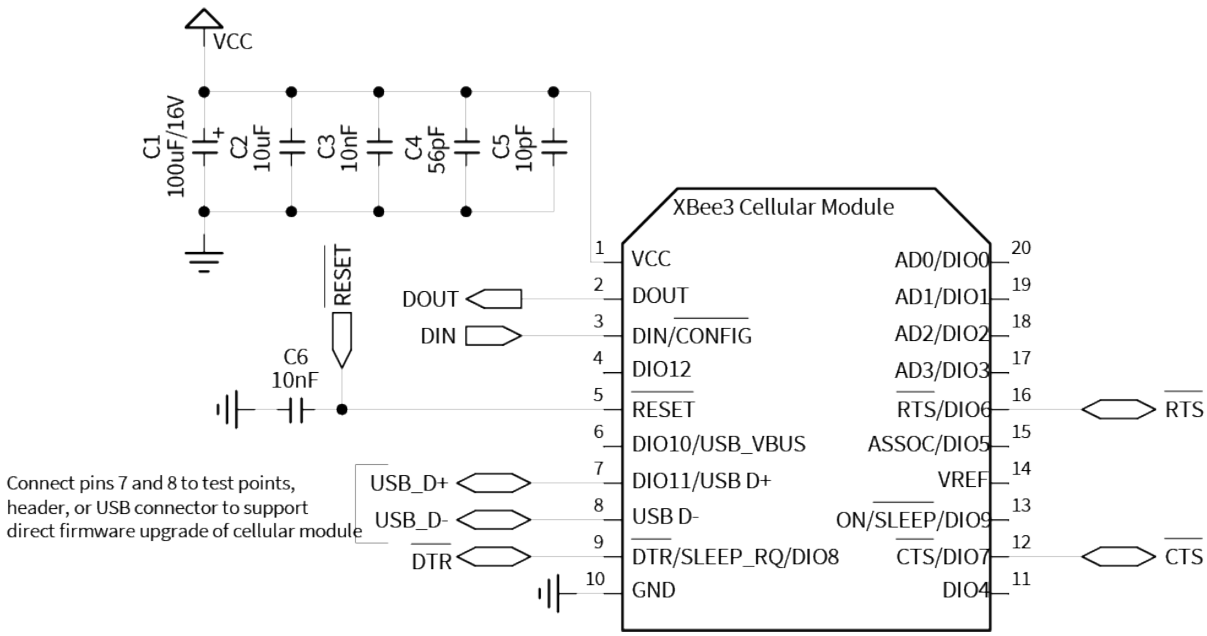
Power supply considerations

When considering a power supply, use the following design practices.

1. Power supply ripple should be less than 75 mV peak to peak.
2. The power supply should be capable of providing a minimum of 750 mA at 3.3 V (2.5 W).
3. Place sufficient bulk capacitance on the XBee VCC pin to maintain voltage above the minimum specification during transmissions. Power consumption lists the peak current during transmitting.
4. Place smaller high frequency ceramic capacitors very close to the XBee Smart Modem VCC pin to decrease high frequency noise.
5. Use a wide power supply trace or power plane to ensure it can handle the peak current requirements with minimal voltage drop. The supply should be inside the supply voltage operating range at startup and should not be allowed to droop lower than 3.2 V during operation.

Minimum connection diagram

In high EMI noise environments, we recommend adding a 10 nF ceramic capacitor very close to pin 5.



Heat considerations and testing

The XBee Smart Modem may generate significant heat during sustained operation. In addition to heavy data transfer, other factors that can contribute to heating include ambient temperature, air flow around the device, and proximity to the nearest cellular tower (the XBee Smart Modem must transmit at a higher power level when communicating over long distances). Overheating can cause device malfunction and potential damage.

The XBee Smart Modem must not be operated in ambient temperatures exceeding 85 °C. Additionally, if you expect to operate the product above 70 °C, we recommend that you perform an analysis of your application to characterize the self-heating of the XBee device:

1. Set up the device in the typical operating scenario you plan to use it in.
2. Monitor the device temperature using TP (Temperature) until it reaches a steady state.
3. Convert the returned value from hex format to decimal.

If the reading is greater than 5 °C above the ambient temperature, we recommend either de-rating the maximum ambient temperature or implementing heat mitigating measures; for example, reduce transmission frequency and duration, enter sleep mode more frequently, or improve airflow. Addressing heat issues will help to ensure long term device reliability.

Clean shutdown

WARNING! Improper shutdown of the modem may result in the underlying cellular module becoming irrecoverably unresponsive.

Digi strongly recommends performing a clean shutdown procedure on your XBee cellular devices before removing power from the devices. Performing a shutdown allows the module to unregister from the cellular network and safely store operating parameters. Failure to shutdown properly has the potential to result in delays resuming network operation and in some rare instances may result in an unrecoverable module failure.

You can use any of the following methods to perform a clean shutdown.

SD (Shutdown) command

You should use the SD command to safely shut down a device before removing power. This is the recommended method.

Issue the **SD** command. When the shut down process is complete, the device returns **OK**. After the device responds **OK**, you can safely remove power from the device.

The device will return **ERROR** if any of the following actions are in progress:

- Over-the-air update of the cellular component
- Local update of the cellular component
- Over-the-air update of the XBee firmware.

In addition, if the radio can't be fully shut down within two minutes, the device returns **ERROR**.

You can verify the state of the device using the AI command. After you issue the **SD** command and a response has been returned (either **OK** or **ERROR**), issue the **AI** command. If the shutdown was successful, **2D** is returned.

Cellular component firmware updates

Even if you do not plan to use the USB Direct interface (Pin 7 and 8), we strongly recommend you provide a way to access the USB pins (Pin 7 and 8) to support direct firmware updates of the Cellular modem. USB Direct provides the fastest means to update the cellular modem firmware. You should keep Pins 7 and 8 routing as a 90 ohm diff pair for USB communications.

CAUTION! If you do not provide access to these USB pins, you may be unable to perform cellular component firmware updates.

One way to provide access to the USB interface is to connect the USB pins to a header or USB connector on the host design. At a minimum you should connect pins 7 and 8 to test points so they are easy to wire to a connector if necessary.

If you are using the USB pins for other purposes you must provide a way to disconnect those interfaces during USB operation, such as using zero ohm resistors.

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 you need, for example, to maintain certain settings for manufacturing or want to ensure a feature is always enabled. When you perform a factory reset on the device using the RE command, the custom configuration is set on the device rather than 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 the **RE** command to reset the device to the factory defaults, the Bluetooth configuration is 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 device's memory runs out of space. If there is no space left to save a configuration, XBee returns an error.

You can use the **!C** command to clear or overwrite a custom configuration at any time.

Set a custom configuration

1. Open XCTU on the device.
2. Enter Command mode.
3. Perform the following process for each configuration that you want to set as a factory default.
   a. Issue an **AT%F** command. This command enables you to enter a custom configuration.
   b. Issue the custom configuration command. For example: **ATBT 1**. This command sets the default for Bluetooth to enabled.

Clear all custom configurations on a device

After you have set configurations using the AT%F command, you can return all configurations to the original factory defaults.

1. Open XCTU on the device.
2. Enter Command mode.
3. Issue **AT!C**.

SIM cards

- For reliability, use a SIM card with gold-plated contacts. Gold-plated contacts provide protection against oxidation, which can occur over time and with exposure to humidity in the air.
- Vibration in the application environment is the most common cause of SIM card failure, which results in loss of communications with the mobile network.
- The specific failure mode is fretting between the contacts of the SIM card and the card holder. For highest reliability, Digi strongly recommends that you apply a thin layer of dielectric grease to the SIM contacts prior to installing the SIM card. You need only to apply enough dielectric grease that the mating area of the contacts is protected from exposure to air and humidity.

Cellular connection process

Connecting

In normal operations, the XBee Smart Modem automatically attempts both a cellular network connection and a data network connection on power-up. The sequence of these connections is as follows:

Cellular network

1. The device powers on.
2. The modem reads the SIM card.
3. It looks for cellular towers.
4. It chooses a candidate tower based on SIM card setting and signal strength.
5. It negotiates a connection.
6. It completes cellular registration.

Data network connection

1. The network enables the evolved packet system (EPS) bearer with an access point name (APN). See AN (Access Point Name) if you have APN issues.
2. The device negotiates a data connection with the access point.
3. The device receives its IP configuration and address.
4. The AI (Association Indication) command now returns a **0** and the sockets become available.

Data communication with remote servers (TCP/UDP)

Once the data network connection is established, communication with remote servers can be initiated in several ways.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

- Transparent mode data sent to the serial port (see TD (Text Delimiter) and RO (Packetization Timeout) for timing).
- API mode: Transmit (TX) Request: IPv4 - 0x20 received over the serial connection.
- Extended Sockets API frames
- MicroPython
- Digi Remote Manager connectivity begins.

Data communication begins when:

1. A socket opens to the remote server.
2. Data is sent.

Data connectivity ends when:

1. The server closes the connection.
2. The **TM** timeout expires (see TM (IP Client Connection Timeout)).

3. The cellular network may also close the connection after a timeout set by the network operator.

Disconnecting

When the XBee Smart Modem is put into Airplane mode, deep sleep is requested, or ATSD (shutdown) command is executed:

1. Sockets are closed, cleanly if possible.
2. The cellular connection is shut down.
3. The cellular component is powered off.

Note We recommend performing a safe shutdown before resetting or rebooting the device to allow the cellular module to detach from the network.

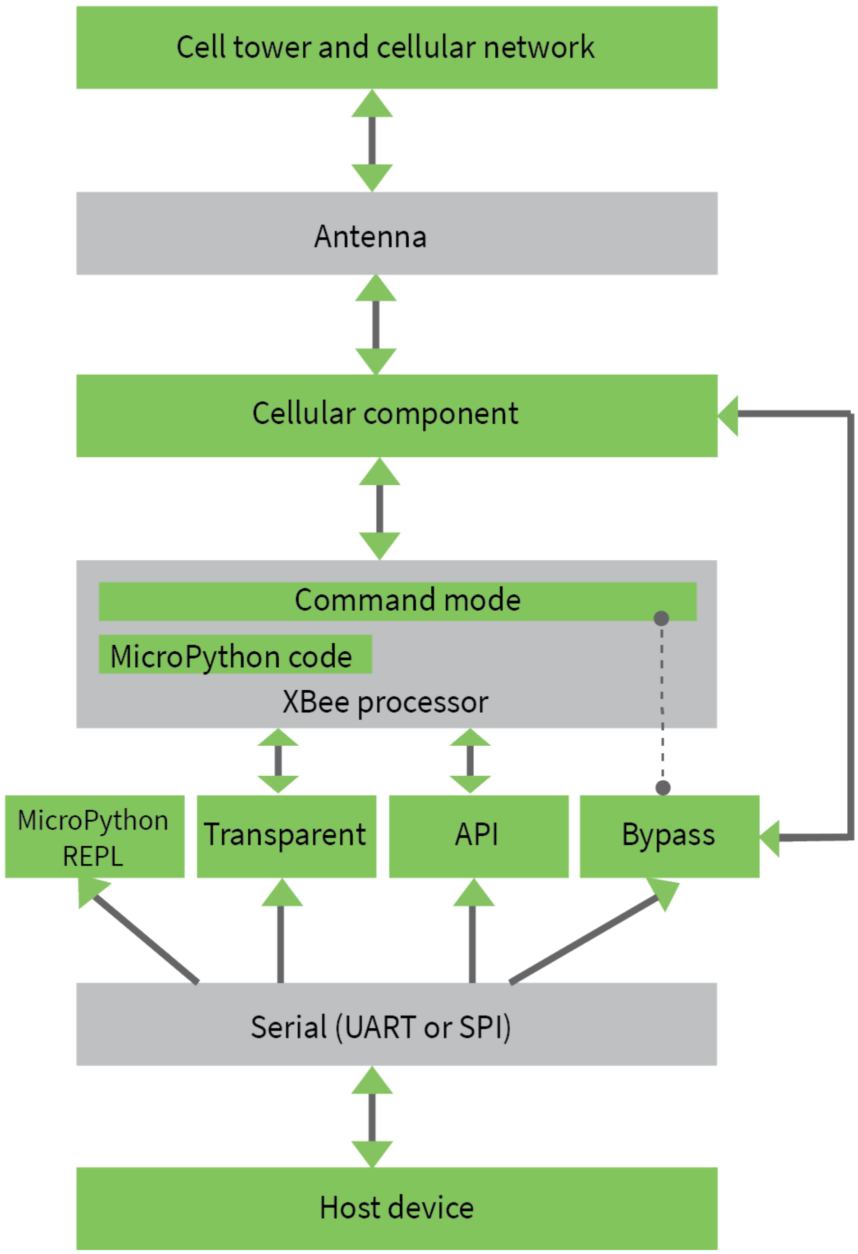
Modes

Select an operating mode

The XBee Smart Modem interfaces to a host device such as a microcontroller or computer through a logic-level asynchronous serial port. It uses a UART for serial communication with those devices.

The XBee Smart Modem supports three operating modes: Transparent operating mode, API operating mode, and Bypass operating mode. The default mode is Transparent operating mode. Use the AP (API Enable) command to select a different operating mode.

The following flowchart illustrates how the modes relate to each other.



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 serial 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. You can set the configuration parameters using Command mode.

The IP (IP Protocol) command setting controls how Transparent operating mode works for the XBee Smart Modem.

Note Transparent operation is not available when using SPI.

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

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 Smart Modem 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 Smart Modem 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.

Enter Command mode

To get a device to switch into Command mode, you must issue the following sequence: **+++** within one second. There must be at least one second preceding and following the **+++** sequence. Both the command character (**CC**) and the silence before and after the sequence (**GT**) are configurable. 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 Smart Modem 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 Sequence Character), CT (Command Mode Timeout) and GT (Guard Times).

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 (Baud Rate) = **3** (9600 b/s).

There are two alternative ways to enter Command mode:

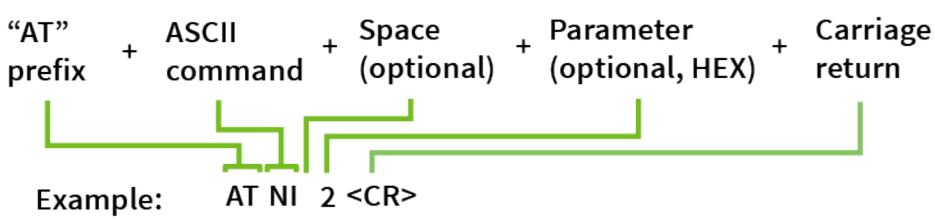
- A serial break for six seconds enters Command mode. You can issue the "break" command from a serial console, it is often a button or menu item.
- Asserting DIN (serial break) upon power up or reset enters Command mode. XCTU guides you through a reset and automatically issues the break when needed.

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.



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>**.

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 Smart Modem responds with **OK<cr>** if successful and **ERROR<cr>** if not.

For devices with a file system:

ATAP1<cr>

OK<cr>

When reading parameters, the device returns the current parameter value instead of an **OK** message.

ATAP<cr>

1<cr>

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. Exit Command mode.

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 a RE (Restore Defaults) to wipe all settings to their factory defaults including those saved using **WR**.

Note You still have to use **WR** to save the changes enacted with **RE**.

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.

MicroPython mode

MicroPython mode (**AP** = **4**) allows you to communicate with the XBee Smart Modem using the MicroPython programming language. You can use the MicroPython Terminal tool in XCTU to communicate with the MicroPython stack of the XBee Smart Modem through the serial interface.

MicroPython mode connects the primary serial port to the stdin/stdout interface on MicroPython, which is either the REPL or code launched at startup.

When code runs in MicroPython with **AP** set to a value other than **4**, stdout is discarded and there is no input to read on stdin.

USB direct mode

Note In order to use USB direct mode in Digi XBee development kits, you must use the XBIB-C-TH development board.

Note You should use this mode if you want to connect using PPP through the cellular modem while using a host operating system, such as embedded Linux.

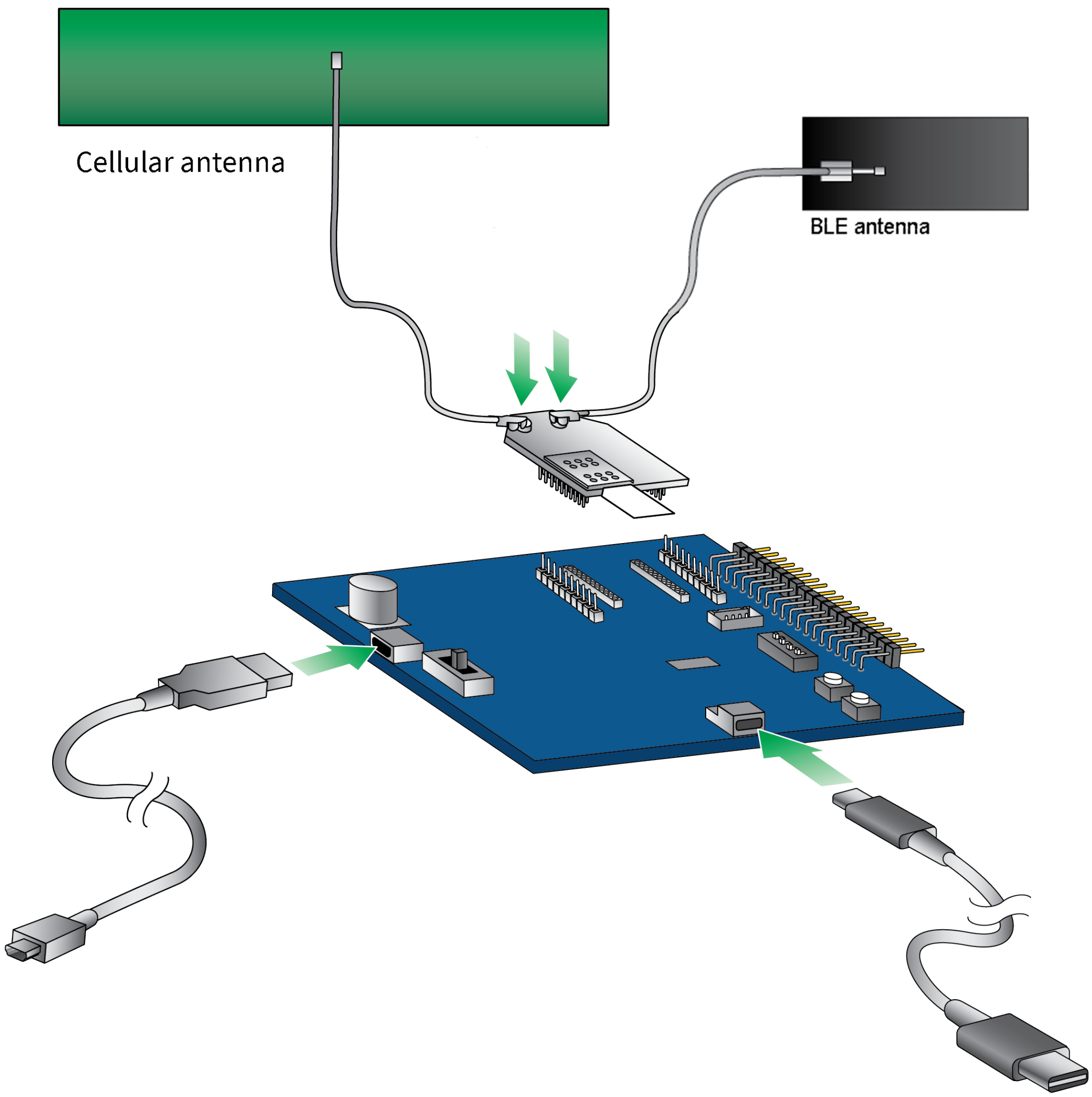
This mode allows you to access the XBee Smart Modem's USB interface directly through XBee pins 7 and 8. VBUS functionality is optionally provided on XBee pin 6 if you wish to enable and disable USB mode based on an external source. While in USB mode the cellular modem is not able to communicate serially with the XBee MCU. All communication with the cellular modem must be performed by the user via the USB port.

Connect the hardware for USB Direct mode

Before you begin, you must connect the hardware. Refer to the image below.

1. Connect the USB-C cable from a PC to the USB port on the development board. The computer searches for a driver, which can take a few minutes to install.
2. Connect the micro USB cable from a PC to the micro USB port on the development board.
3. Move both switches to the left position. For more information, see XBIB-CU-TH reference.

Note The USB port on the PC should be a minimum of USB 3.0 to supply adequate power, and for the device to work as expected.



Enable USB direct mode

When you enable USB direct mode, you must also determine the USB VBUS signal state.

1. Enable USB direct mode.

   Set ATP1 to **7** to configure pins 7 and 8 for USB direct mode.

   When set to **7**, DIO11/PWM1 (pin 7) brings out the USB D+ signal of the cellular component. The USB D- signal is available on pin 8. With these pins connected to a USB host, a direct connection is made to the cellular component which is not mediated by the XBee processor.

   When in USB direct mode, ATAI returns **0x2B**.

   Note If USB Direct is not enabled (**P1** is not set to **7**), then setting ATDO to bit 2 and ATP0 to **6** have no effect on the USB VBUS state.

2. Determine the USB VBUS signal state, using one of the following options.
   - Set ATP0 to **6**. The USB VBUS signal sent to the modem is based on the state of the **P0** pin.

     Apply a logic high signal to DIO10/PWM0 (pin 6) to enable USB, or a logic low signal to disable USB.
   - If ATP0 is not set to **6**, the USB VBUS signal state that the XBee sends to the modem follows the state of ATDO bit **2**. Options are:
     - If **ATDO** bit **2** is set, VBUS is set to high.
     - If **ATDO** bit **2** is cleared, VBUS is set to low.
3. Reset the device to complete the process. When USB direct mode is enabled, AI (Association Indication) returns 0x2B.

Note Although pin 6 is 5 V tolerant on this device, it operates with the same 3.3 V logic as the other XBee device pins. For compatibility with other XBee devices we recommend driving the line with no more than 3.3 V. Moreover, driving the pin at 5 V will cause input leakage current to increase to 3.3 μA typical.

Configure and use PPP with an XBee 3 modem

Your XBee 3 Cellular device can communicate directly with the modem and can drop into PPP mode.

Prerequisites

- A working SIM card to get onto the network.
- Knowledge of the APN for the given network and SIM.
- A Linux distribution with pppd/chat.

Step 1: Configure the device for PPP

USB direct is used to gain access to the underlying modem, which enables the use of PPP.

1. Set up USB direct mode.
2. Issue the WR command to save the settings.

Once USB direct is configured, an additional USB device should be attached to the Linux machine. In order to have a consistent device name on the Linux machine, you should set up a udev rule for the device, as described in the next step.

Step 2: Set up the USB device for use with PPP

A udev rule is needed to give the USB connection a constant name using a symlink.

1. Make sure that the modem is plugged in.
2. Place the following **ppp-setup.rules** file here: **/etc/udev/rules.d**

```
# Sara-R410 rule
SUBSYSTEM=="tty", ATTRS{bInterfaceNumber}=="02", ENV{ID_VENDOR_ID}=="05c6",
ENV{ID_MODEL_ID}=="90b2", SYMLINK+="ppp_direct_usb"
```

3. You must run the two commands shown below to restart the udev daemon to apply the new rule.

```
sudo udevadm control --reload-rules
sudo udevadm trigger
```

4. Verify that the new device has been created: **/dev/ppp_direct_usb**. If was not, make sure the modem is plugged in and then repeat this process.

Step 3: Configure PPPD

PPPD by default looks in the **/etc/ppp/** directory for an options file and a chat script. The option file configures and specifies the chat script for PPPD. The chat script configures and dials the modem for the PPP connection.

1. Below is an example of an options file. This file must be in the **/etc/ppp/** directory.

```
## Show debug info
debug
## Modem serial port
/dev/ppp_direct_usb
## Baud-rate
921600
## Hardware flow control using rts/cts
crtscts
## For debugging purposes
nodetach
## Bring up the connection if it gets shutdown
persist
## Disable remote authentication
noauth
## Control character map
asyncmap 0
## Setup interface as default route
defaultroute
replacedefaultroute
## disable getting the local IP address from the host-name
noipdefault
## Accept new IP addresses from IPCP negotiations (default)
ipcp-accept-local
ipcp-accept-remote
## Lock the serial device
lock
## Let the remote designate the name-servers
usepeerdns
## Enable IPv6 and use provided address
+ipv6 ipv6cp-use-ipaddr
## Connect script (chat script)
connect "/usr/sbin/chat -V -t 60 -f net-chat"
```

2. Place the chat script in the **/etc/ppp/** directory. An example is shown below. The net-chat script is an automated script that both configures and dials the modem for the PPP connection. This script turns on hardware flow-control, sets the APN, sets the DSR line to ON, and dials the peer.

Note In the net-chat script below, you must replace <APN> with the correct APN for your network and SIM.

```
ABORT 'ERROR'
ABORT 'BUSY'
ABORT 'NO CARRIER'
'' AT
OK AT+IFC=2,2
OK ATE0
OK AT+CGDCONT=1,"IP","<APN>"
OK AT&S0
OK ATD*99***1#
CONNECT
```

Step 4: Run PPPD

PPPD is the program that brings up the PPP interface.

1. You should bring down any other network interfaces that may complicate routing.
2. Run PPPD to bring up the PPP interface.

```
sudo pppd
```

3. Various LCP, PAP and IPCP messages should be output. If the interface was brought up correctly **ifconfig** should list a PPP interface as **pppx** (where x is a number).
4. Ping a web server from the PPP interface.

```
ping www.digi.com
```

Step 5: Low power use case

You may want to reduce power consumption by turning off the XBee modem. Follow this process to properly bring down the PPP connection and shut down the modem.

1. Terminate PPPD by sending a terminate signal: Ctrl+C
2. Issue the shutdown command to the modem over the USB connection.

```
AT+CPWROFF
```

3. Wait for an **OK** response.
4. When received, remove power from the XBee.
5. Restart the PPP connection.
   a. Power on the XBee.
   b. Issue the ppp command.

   ```
   sudo pppd
   ```

Note Do not power cycle the modem too often as it can lead to network registration rejection. Cycling should not be performed more than a few times an hour. Check with your network carrier for the exact limits.

Troubleshooting

Error after running sudo pppd

```
+CME ERRORScript /usr/sbin/chat -V -t 60 -f net-chat finished (pid 5523), status
= 0x4
Connect script failed
```

This indicates that the <APN> field was most likely not set correctly in the net-chat script.

Error after running sudo pppd

```
pppd: In file /etc/ppp/options: unrecognized option '/dev/ppp_direct_usb'
```

This indicates pppd could not open up the USB port to the modem. Make sure that the modem is plugged in and shows up under the **/dev/** directory as **ppp_direct_usb**.

Error after running "ping www.digi.com"

```
ping: unknown host www.digi.com
```

The name server was not setup correctly for the PPP interface. Make sure there is a valid name server in **/etc/resolv.conf**.

Bypass operating mode (DEPRECATED)

WARNING! Bypass mode is now deprecated and is not recommended for new designs. XBee 3 Cellular products support direct USB to access the cellular modem directly. See USB direct mode for details on how to configure your XBee to use direct USB.

CAUTION! Bypass operating mode is an alternative to Transparent and API modes for advanced users with special configuration needs. Changes made in this mode might change or disable the device and we do not recommended it for most users.

CAUTION! The lack of hardware flow control support in the cellular component may impact the reliability of Bypass mode. See Hardware flow control in Bypass mode for details.

In Bypass mode, the device acts as a serial line replacement to the cellular component. In this mode, the XBee Smart Modem exposes all control of the cellular component's AT port through the UART. If you use this mode, you must setup the cellular modem directly to establish connectivity. The modem does not automatically connect to the network.

Note The cellular component can become unresponsive in Bypass mode. See Unresponsive cellular component in Bypass mode for help in this situation.

When Bypass mode is active, most of the XBee Smart Modem's AT commands do not work. For example, **IM** (IMEI) may never return a value, and **DB** does not update. In this configuration, the firmware does not test communication with the cellular component (which it does by sending AT commands). This is useful in case you have reconfigured the cellular component in a way that makes it incompatible with the firmware. Bypass operating mode exists for users who wish to communicate directly with the cellular component settings and do not intend to use XBee Smart Modem software features such as API mode.

Command mode is available while in Bypass mode; see Enter Command mode for instructions.

Enter Bypass operating mode

To configure a device for Bypass operating mode:

1. Set the AP (API Enable) parameter value to **5**.
2. Send WR (Write) to write the changes.
3. Send FR (Force Reset) to reboot the device.
4. After rebooting, enter Command mode and verify that Bypass operating mode is active by querying AI (Association Indication) and confirming that it returns a value of **0x2F**.

It may take a moment for Bypass operating mode to become active.

Leave Bypass operating mode

To configure a device to leave Bypass operating mode:

1. Set AP (API Enable) to something other than 5.
2. Send WR (Write) to write the changes.
3. Send FR (Force Reset) to reboot the device.
4. After rebooting, enter Command mode and verify that Bypass operating mode is not active by querying AI (Association Indication) and confirming that it returns a value other than **0x2F**.

Restore cellular settings to default in Bypass operating mode

Send **AT&F1** to reset the cellular component to its factory profile.

Sleep modes

About 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.

Normal mode

Set **SM** to 0 to enter Normal mode.

Normal mode is the default sleep mode. If a device is in this mode, it does not sleep and is always awake.

Devices in Normal mode are typically mains powered.

Pin sleep mode

Set **SM** to 1 to enter pin sleep mode.

Pin sleep allows the device to sleep and wake according to the state of the SLEEP_RQ pin (SLEEP_RQ).

When you assert SLEEP_RQ (high), the device finishes any transmit or receive operations, closes any active connection, and enters a low-power state.

When you de-assert SLEEP_RQ (low), the device wakes from pin sleep.

Cyclic sleep mode

Set **SM** to 4 to enter Cyclic sleep mode.

Cyclic sleep allows the device to sleep for a specific time and wake for a short time to poll.

If you use the **D7** command to enable hardware flow control, the $\overline{\text{CTS}}$ pin asserts (low) when the device wakes and can receive serial data, and de-asserts (high) when the device sleeps.

Cyclic sleep with pin wake up mode

Set **SM** to 5 to enter Cyclic sleep with pin wake up mode.

This mode is a slight variation on Cyclic sleep mode (**SM** = 4) that allows you to wake a device prematurely by de-asserting the SLEEP_RQ pin (SLEEP_RQ).

In this mode, you can wake the device after the sleep period expires, or if a high-to-low transition occurs on the SLEEP_RQ pin.

SPI mode and sleep pin functionality

SLEEP_RQ/ DIO8 is configured as a peripheral by default and is used for pin sleep to wake the XBee Smart Modem and put it to sleep. This applies regardless of if the serial interface is UART or SPI.

However, if SLEEP_RQ 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_RQ. Asserting SPI_SSEL by driving it low wakes the XBee Smart Modem, or keeps it awake. De-asserting SPI_SSEL by driving it high puts the device to sleep.

If neither pin is configured as a peripheral, then the device stays awake, being unable to sleep when SM (Sleep Mode) is **1**.

DIO8/SLEEP_RQ configured as peripheral (D8 = 1)?	DIO3/SPI_SSEL configured as peripheral (D3 = 1)?	Pin sleep controlled by...
Yes	Yes	DIO8/SLEEP_RQ
Yes	No	DIO8/SLEEP_RQ
No	Yes	DIO3/SPI_SSEL
No	No	Neither (pin sleep does not work)

Advantage of using SPI_SSEL to control sleep:

- One less physical pin connection is required to implement pin sleep. This makes DIO8/SLEEP_RQ available for another purpose.

Disadvantages of using SPI_SSEL to control sleep:

- The XBee Smart Modem is put to sleep whenever the SPI master negates SPI_SSEL, even if that was not the intent.
- The XBee Smart Modem begins entering sleep as soon as the control pin is asserted (brought high). Immediately de-asserting the control pin (bringing it low) only has the effect of preventing the microcontroller from entering low-power mode before waking up the device—all other sleep preparations (such as closing sockets) continue as in typical sleep operation. This can take several seconds, and this added time in the case of an unintended sleep request may not be acceptable.

Sleep timer

The sleep timer starts when the device wakes and resets on re-configuration. When the sleep timer expires the device returns to sleep.

MicroPython sleep behavior

When the XBee Smart Modem enters Deep Sleep mode, any MicroPython code currently executing is suspended until the device comes out of sleep. When the XBee Smart Modem comes out of sleep mode, MicroPython execution continues where it left off.

Upon entering deep sleep mode, the XBee Smart Modem closes any active UDP connections and turns off the cellular component. As a result, any sockets that were opened in MicroPython prior to sleep report as no longer being connected. This behavior appears the same as a typical socket disconnection event will:

- **socket.send** raises **OSError: ENOTCONN**
- **socket.sendto** raises **OSError: ENOTCONN**
- **socket.recv** returns the empty string, the traditional end-of-file return value
- **socket.recvfrom** returns an empty message, for example:
  (b'', (<address from connect()>, <port from connect()>))
  The underlying UDP socket resources have been released at this point.

Power saving features and design recommendations

Airplane mode

While not technically a sleep mode, Airplane mode is another way of saving power. When set, the cellular component of the XBee Smart Modem is fully turned off and no access to the cellular network is performed or possible. Use AM (Airplane Mode) to configure this mode.

Power Saving Mode (PSM)

Enable PSM

To enable PSM, set DO (Device Options) bit 3.

Note For NB-IoT, TCP and SMS support is dependent on the network. Contact your network provider for details.

Note The cellular module comes out of the PSM low-power state whenever any network activity occurs, including Remote Manager activity. See Verify the connection between a device and Remote Manager.

When PSM is enabled, the cellular component spends most of its time in a low power state. In the low power state the XBee Smart Modem still has an IP address and is registered to the network, which allows for quick resumption of activity, but is not reachable so cannot receive IP or SMS traffic until it wakes up. This low power state is used even when taking advantage of XBee sleep features (such as Pin sleep mode or Cyclic Sleep), rather than powering the cellular component off entirely to ensure readiness when exiting sleep.

The cellular component wakes to participate in maintaining the network state periodically based on timers negotiated with the cell tower. It is also triggered to wake up when the user performs any activity requiring network connectivity such as mobile-originated traffic like sending an SMS or UDP/TCP traffic. When it wakes up, it spends a short time awake so that it is reachable through the network at that time and then returns to the low power state.

Overview of PSM functionality on XBee 3 Cellular

Before you enable PSM, you should be aware of these behaviors.

Actions taken when the cellular component enters a PSM dormant state

When the cellular component enters into the PSM "dormant" state:

- Any existing TCP or TLS connections are immediately closed.
- Existing sockets are implicitly closed.
- Any queued or not-yet-in-progress transmissions such as data sent on sockets, or requested SMS transmissions, are canceled and an error status is returned.

Wake up a dormant cellular component

The following sections explain the different activities that will wake a dormant cellular component.

Mobile-originated activity

If the cellular component is "dormant" (AI = 0x2C), any mobile-originated activity which requires network connectivity will trigger the cellular component to wake up, including creating a TCP or TLS connection, sending a UDP datagram, or sending an SMS. Typical applications do not need to monitor **ATAI** to detect the PSM "dormant" state (AI = 0x2C), because this state will not cause these activities to error out.

Note In firmware versions ending in *16 and earlier, sending an SMS did not wake the cellular component.

Mobile-terminated activity

While the cellular component is in the PSM "dormant" state, the device is not reachable over the network, and mobile-terminated traffic such as SMS or IP connections will not trigger it to wake up.

Additional activities

Additional activities which will trigger the cellular component to wake up from PSM include:

- Creating a TCP or TLS connection. This includes the connection implicitly created when using transparent mode or TX IPv4 API frames, as well as explicit connections like a Socket Connect API frame, or calling `connect(...)` on a MicroPython socket or using a MicroPython library which does so.
- Creating a TCP "listener" socket.
- Sending a UDP datagram.
- Binding a UDP socket to a specific port.
- Performing a DNS lookup request using ATLA, or `socket.getaddrinfo(...)` in MicroPython.
- Using MicroPython to upload datapoints to Digi Remote Manager.

XBee 3 cellular device features that periodically wake up the cellular component

Features of the XBee 3 Cellular device which will automatically wake up the cellular component on a periodic basis include:

- Digi Remote Manager status checks. See the DF command to control the interval of these checks.
- Upload of health metrics to Digi Remote Manager. See the HM and HF commands.

XBee 3 cellular device features that continually wake up the cellular component

Features of the XBee 3 Cellular device that continually wake up the cellular component do not take advantage of power savings and should not be used in combination with PSM.

Features that continually wake up the cellular component include:

- Persistent TCP connection to Digi Remote Manager (if bit 0 of MO is set).
- Using a server/listening socket in transparent mode or API mode (if C0 is not 0 and AP is 0, 1 or 2).

XBee sleep features (Pin Sleep or Cyclic Sleep) and the PSM feature

You can use XBee sleep features (such as Pin Sleep or Cyclic Sleep) when the PSM feature is enabled. The sections below explain how they work together.

- XBee sleep features (such as Pin Sleep or Cyclic Sleep) do not immediately put the cellular component into its PSM state. When PSM is enabled, during XBee sleep the cellular component continues to be powered so that it can manage the PSM active and tracking area timers and go into the PSM state on its own schedule. The XBee cannot and does not directly put the cellular component into a PSM dormant state.

- When waking from XBee sleep (such as Pin Sleep or Cyclic Sleep), if the cellular component is in the PSM state (AI = 0x2C), it remains in the PSM state until there is activity which triggers it to wake up. In other words, XBee sleep is typically orthogonal to PSM.

Note In firmware versions ending in *16 and earlier, waking from XBee sleep would trigger the cellular component to wake up. This was changed in firmware version *17 to ensure better power-saving behavior.

PSM behavior

Commands exist to influence the PSM behavior when PSM is enabled (**DO** bit 3 is set).

Note For NB-IoT, TCP and SMS support is dependent on the network. Contact your network provider for details.

They are:

- PA (Requested Active Timer)
- PU (Requested Tracking Area Update Timer)

PA and **PU** are the values the XBee Smart Modem requests from the network. The network is free to assign values other than those which have been requested.

See the LTE-M Deployment Guide from GSMA for a description of what the PSM timers are and what functions they perform on the network.

Low voltage shutdown

The XBee Smart Modem can monitor the XBee VCC line in order to detect a failing power supply. Monitoring the VCC line can prevent possible memory corruption on both the cellular modem and the file system due to insufficient power. This feature is recommended for users who run the XBee off of a battery.

You must first enable this feature and then set a base threshold for the voltage on the XBee Vcc line. When the voltage falls below the base threshold, the XBee goes into a shutdown state. When in a shutdown state:

- The cellular modem will be shut down completely, halting any network activity.
- The file system will be shut down completely, disallowing any file system operations.

Once in this state, the XBee will resume normal functionality only after a reset. A reset is triggered if the voltage rises above an upper threshold set by a combination of values.

Note The XBee VCC voltage gets read periodically, once every two minutes. Consequently, it may take up to two minutes to change to or from a shutdown state.

Enable and configure the low voltage shutdown feature

1. Enable the feature by setting the DO command bit 4.
2. Set the base threshold for the voltage on the XBee VCC line using the %L command. When the voltage for the XBee VCC line goes below the base threshold, the XBee goes into a shut down state.
3. Set the reset offset for the XBee VCC line using the %M command. The XBee resets and resumes normal operation when the voltage reaches the base threshold set in the %L command, plus the value of the reset offset set in the %M command.

Example

The graph shown below demonstrates this feature. In this example, AT%L (Base Threshold) is set to 0xC1C (3100 mV) and AT%M (Reset Offset) is set to 0x64 (100 mV).

- After the XBee VCC voltage drops below the base threshold of 3100 mV (set by AT%L), the XBee goes into the shutdown state.
- When in the shutdown state, the XBee VCC voltage must rise 100 mV (set by AT%M) above the shutdown voltage (AT%L) to reset and then resume normal operation.

Deep Sleep mode

In Deep Sleep mode the cellular component is shut off and the XBee processor is put to sleep.

Note When the XBee Smart Modem enters deep sleep mode, any MicroPython code currently executing is suspended until the device comes out of sleep.

Serial communication

Serial interface

The XBee Smart Modem interfaces to a host device through a serial port. The device's serial port can communicate:

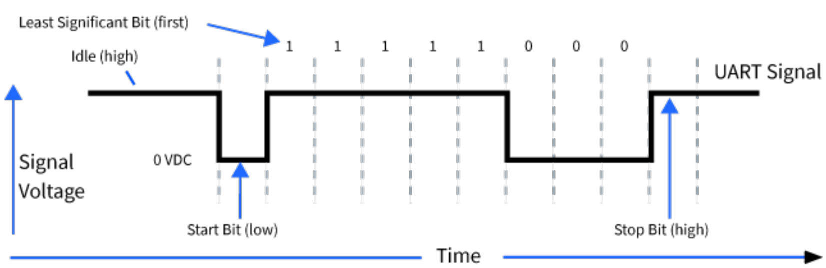
- Through a logic and voltage compatible universal asynchronous receiver/transmitter (UART).
- Through a level translator to any serial device, for example, through an RS-232 or USB interface board.
- Through a serial peripheral interface (SPI) port.

Serial data

A device sends data to the XBee Smart Modem's UART through pin 3 DIN 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 Smart Modem) 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.

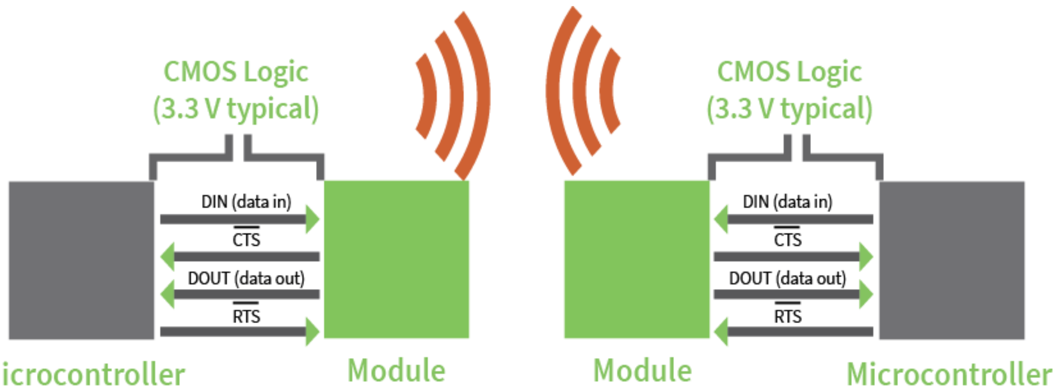


You can configure the UART baud rate, parity, and stop bits settings on the device with the **BD**, **NB**, and **SB** commands respectively. For more information, see Serial interfacing commands.

In the rare case that a device has been configured with the UART disabled, you can recover the device to UART operation by holding DIN low at reset time. DIN forces a default configuration on the UART at 9600 baud and it brings the device up in Command mode on the UART port. You can then send the appropriate commands to the device to configure it for UART operation. If those parameters are written, the device comes up with the UART enabled on the next reset.

UART data flow

Devices that have a UART interface connect directly to the pins of the XBee Smart Modem 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.



Serial buffers

The XBee Smart Modem maintains internal 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 or SPI port.

Flow control (output)

We strongly encourage you to use flow control with the XBee Smart Modem to prevent buffer overruns.

Flow control (output) is enabled by default; you can disable it with D7 (DIO7/CTS). When the serial receive buffer fills with the number of bytes specified by FT (Flow Control Threshold), the device de-asserts CTS (sets it high) to signal the host device to stop sending serial data. The device re-asserts CTS when less than FT-32 bytes are in the UART receive buffer.

Note Serial flow control is not possible when using the SPI port.

Flow control (input)

If you set D6 (DIO6/RTS) to enable flow control (input), 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.

Enable UART or SPI ports

To enable the UART port, configure DIN and DOUT (**P3** and **P4** parameters) as peripherals. To enable the SPI port, enable SPI_MISO (**P2**), SPI_MOSI (**D4**), SPI_SSEL (**D3**), and SPI_CLK (**D2**) as peripherals. If you enable both ports then output goes to the UART until the first input on SPI.

When both the UART and SPI ports are enabled on power-up, all serial data goes out the UART. As soon as input occurs on either port, that port is selected as the active port and no input or output is allowed on the other port until the next device reset.

If you change the configuration so that only one port is configured, then that port is the only one enabled or used. If the parameters are written with only one port enabled, then the port that is not enabled is not used even temporarily after the next reset.

If both ports are disabled on reset, the device uses the UART in spite of the wrong configuration so that at least one serial port is operational.

I^2C

For I^2C see the Class I2C: two-wire serial protocol section in the *MicroPython Programming Guide* for details.

SPI operation

SPI communications

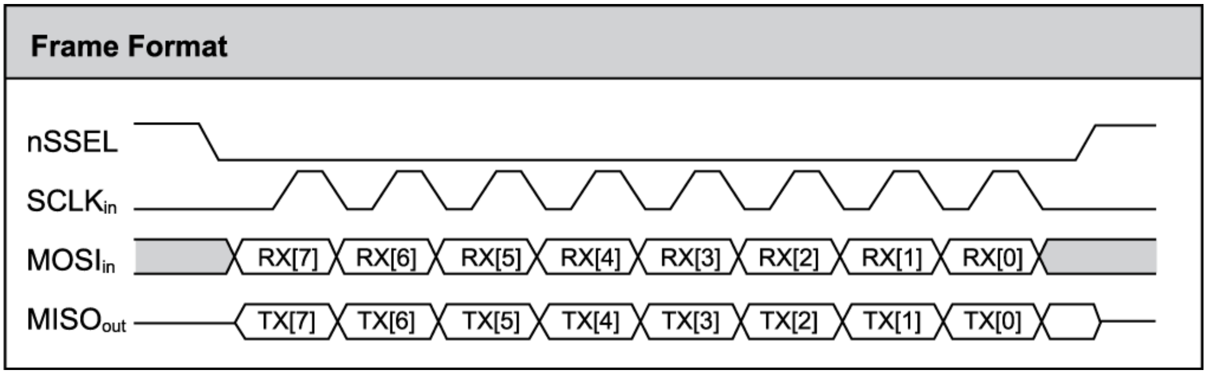
The XBee Smart Modem supports SPI communications in slave mode. Slave mode receives the clock signal and data from the master and returns data to the master. The following table shows the signals that the SPI port uses on the device.

Signal	Function
SPI_MOSI (Master Out, Slave In)	Inputs serial data from the master
SPI_MISO (Master In, Slave Out)	Outputs serial data to the master
SPI_SCLK (Serial Clock)	Clocks data transfers on MOSI and MISO
SPI_$\overline{\text{SSEL}}$ (Slave Select)	Enables serial communication with the slave
SPI_$\overline{\text{ATTN}}$ (Attention)	Alerts the master that slave has data queued to send. The XBee Smart Modem 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 4.8 MHz 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 option on the device development evaluation boards.

Full duplex operation

The specification for SPI includes the four signals SPI_MISO, SPI_MOSI, SPI_CLK, and SPI_SSEL. Using these four signals, the SPI master cannot know when the slave needs to send and the SPI slave cannot transmit unless enabled by the master. For this reason, the SPI_ATTN signal is available in the design. This allows the SPI slave to alert the SPI master that it has data to send. In turn, the SPI master is expected to assert SPI_SSEL and start SPI_CLK, unless these signals are already asserted and active respectively. This, in turn, allows the XBee Smart Modem SPI slave to send data to the master.

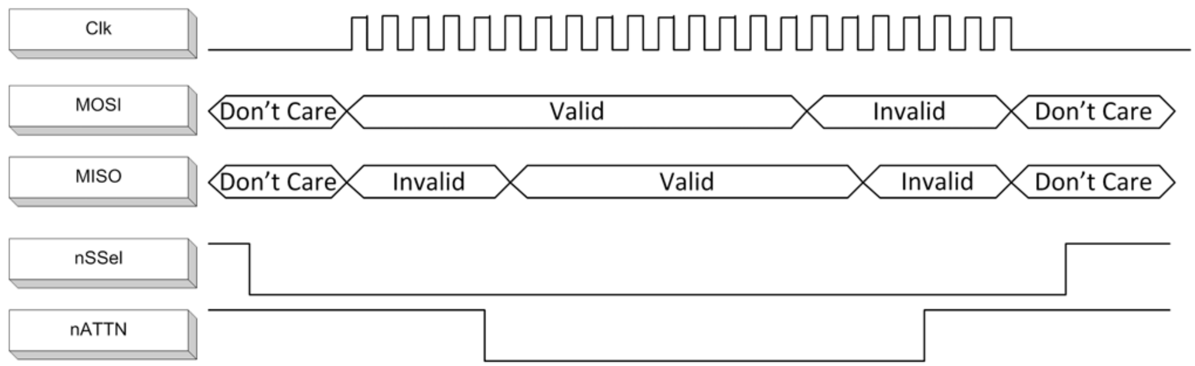
SPI data is latched by the master and slave using the SPI_CLK signal. When data is being transferred the MISO and MOSI signals change between each clock. If data is not available then these signals will not change and will be either 0 or 1. This results in receiving either a repetitive 0 or 0xFF. The means of determining whether or not received data is valid is by packetizing the data with API packets, without escaping. Valid data to and from the XBee Smart Modem is delimited by 0x7E, a length, the payload, and finally a checksum byte. Everything else in both directions should be ignored. The bytes received between frames will be either 0xff or 0x00. This allows the SPI master to scan for a 0x7E delimiter between frames.

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 is sending data to the slave and the slave has valid data to send in the middle of receiving data from the master, it allows a true full duplex operation where data is valid in both directions for a period of time. During this time, the master and slave must simultaneously transmit valid data at the clock speed so that no invalid bytes appear within an API frame, causing the whole frame to be discarded.

An example follows to more fully illustrate the SPI interface during the time valid data is being sent in both directions. First, the master asserts SPI_SSEL and starts SPI_CLK to send a frame to the slave.

Initially, the slave does not have valid data to send the master. However, while it is still receiving data from the master, it has its own data to send. Therefore, it asserts SPI_ATTN low. Seeing that SPI_SSEL is already asserted and that SPI_CLK is active, it immediately begins sending valid data, even while it is receiving valid data from the master. In this example, the master finishes its valid data before the slave does. The master will have two indications of valid data: The SPI_ATTN line is asserted and the API frame length is not yet expired. For both of these reasons, the master should keep SPI_SSEL asserted and should keep SPI_CLK toggling in order to receive the end of the frame from the slave, even though these signals were originally turned on by the master to send data. During the time that the SPI master is sending invalid data to the SPI slave, it is important no 0x7E is included in that invalid data because that would trigger the SPI slave to start receiving another valid frame.

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 (pin 17) 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.

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, hold DOUT/DIO13 pin 2 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. This method forces the configuration to provide full SPI support for the following parameters:

- **D1** (This parameter will only be changed if it is at a default of zero when the method is invoked.)
- **D2**
- **D3**

- **D4**
- **P2**

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. After a reset, parameters that were forced and then written to flash become the mode of operation.

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 enabled at the time of reset, then output goes to the UART until the host sends the first input. If that first input comes on the SPI port, then all subsequent output goes to the SPI port and the UART is disabled. If the first input comes on the UART, then all subsequent output goes to the UART and the SPI 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. The only way to switch the selected serial port is to reset the device. On surface-mount devices, forcing DOUT low at the time of reset has no effect. To use SPI mode on the SMT devices, assert the SPI_SSEL (pin 17) low after reset and before any UART data is input.

When the master asserts the slave select (SPI_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.

Data format

SPI only operates in API mode 1. The XBee Smart Modem does not support Transparent mode or API mode 2 (which escapes control characters). This means that the AP configuration only applies to the UART, and the device ignores it while using SPI. The reason for this operation choice is that SPI is full duplex. If data flows in one direction, it flows in the other. Since it is not always possible to have valid data flowing in both directions at the same time, the receiver must have a way to parse out the valid data and to ignore the invalid data.

The XBee Smart Modem sends **0xFF** when there is no data to send to the host.

File system

For detailed information about using MicroPython on the XBee Smart Modem refer to the *Digi MicroPython Programming Guide*.

Overview of the file system

XBee Smart Modem firmware versions ending in **0B** (for example, 1130B, 100B, 3100B) and later include support for storing files on an internal 1 MB SPI flash.

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 ATFS FORMAT confirm and ensure that the format is complete.

Directory structure

The SPI flash appears in the file system as **/flash**, the only entry at the root level of the file system. It has a **lib** directory intended for MicroPython modules and a **cert** directory for files used for TLS sockets.

Paths

The XBee Smart Modem stores all of its files in the top-level directory **/flash**. On startup, the **ATFS** commands and MicroPython each use that 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 the command **ATFS ls .** 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.

Secure files

The file system includes support for secure files with the following properties:

- Created via the **ATFS XPUT** command or in MicroPython using a mode of ***** with the **open()** method.
- Unable to download via the **ATFS GET** command or MicroPython's **open()** method.
- SHA256 hash of file contents available from **ATFS HASH** command (to compare with a local copy of a file).
- Encrypted on the SPI flash.
- MicroPython can execute code in secure files.
- Sockets can use secure files when creating TLS connections.

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.

Encrypt files

You can encrypt files on the file system. This provides two things:

1. Protection of the client private key for TLS authentication while it is stored on the XBee Smart Modem.
2. Protection for user's MicroPython applications.

Use ATFS XPUT filename to place encrypted files on the file system. The XPUT operation is otherwise identical to the PUT operation. Files placed in this way are indicated with a **pound sign** (**#**) following the filename. The XBee Smart Modem does not allow an encrypted file to be read by normal use so it:

1. Cannot be retrieved with the GET operation.
2. Cannot be opened and read in MicroPython applications.
3. Cannot be created by a MicroPython application.

When ATFS HASH filename is run with the filename of an encrypted file, it reports the SHA256 hash of the file contents. In this way you can validate that the correct file has been placed on the XBee Smart Modem.

Socket behavior

See Socket leaks for instances where a socket leaks when closing a connection while there is pending RX data.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Supported sockets

The XBee Smart Modem supports the following number of sockets:

- 6 maximum: 4 TLS sockets maximum.[1]

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Best practices when using sockets

Sockets and Remote Manager

If you use Remote Manager to remotely communicate with and configure your XBee Cellular device, you must leave at least two sockets available in the system: one UDP socket (for periodic low-data-usage check-ins), and one TCP/TLS socket (to be used when a full connection is needed).

If your application allocates so many sockets that Remote Manager functionality in the firmware cannot get the sockets that it requires, Remote Manager functionality will be prevented from working until sockets become available.

For example, each call to `socket.socket()` in MicroPython will allocate a socket, and this socket will remain allocated to MicroPython until the socket's close method is called, or the MicroPython REPL is restarted using Ctrl-D.

See Supported sockets for more information on the total number of sockets supported by the device.

Sockets and API mode

When using API mode to transmit TCP/TLS data to a remote destination (using the Transmit (TX) Request: IPv4 - 0x20 or Tx Request with TLS Profile - 0x23 frames), sending a large amount of data as a single API frame is preferable to multiple smaller API frames. Using a single large API frame allows the XBee to transmit the data using fewer operations than transmitting multiple pieces of data in sequence, which improves overall throughput.

Additionally, one API frame consumes less dynamic memory in the system than multiple smaller API frames, which means there will be more memory available to process incoming IP data as well as subsequent API frames sent into the XBee Cellular device.

Socket timeouts

The XBee Smart Modem implicitly opens the socket any time there is data to be sent, and closes it according to the timeout settings. The TM (IP Client Connection Timeout) command controls the timeout settings.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

[1] 1 TCP socket is used for Remote Manager, so if you have Remote Manager enabled, subtract 1 socket from the values above.

Socket limits in API mode

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

In API mode there are a fixed number of sockets available; see Supported sockets. When a Transmit (TX) Request: IPv4 - 0x20 frame is sent to the XBee Smart Modem for a new destination, it creates a new socket. The exception to this is when using the UDP protocol with the C0 source port, which allows unlimited destinations on the socket created by C0 (Source Port). If no more sockets are available, the device sends back a Transmit (TX) Status - 0x89 frame with a Resource Error. The Resource Error resolves when an existing socket is closed. An existing socket may be closed when the socket times out (see TM (IP Client Connection Timeout) and TS (IP Server Connection Timeout)) or when the socket is closed via a TX request with the CLOSE flag set.

In API mode each socket has a maximum number of pending Transmit (TX) Requests allowed. When a Transmit (TX) Request: IPv4 - 0x20 frame is sent to the XBee Smart Modem for an existing destination, it sends that request using the socket for that destination. If the number of pending Transmit (TX) Requests would be exceeded for the socket, the device sends back a Transmit (TX) Status - 0x89 frame with a Resource Error indicating that the device is not able to send the request and should retry again later. The Resource Error resolves when a Transmit (TX) Request that is pending on the socket is transmitted; this is indicated by the Transmit (TX) Status frame for the request.

UDP datagram size limits

The maximum supported size for UDP datagrams either transmitted from or received by the XBee is as follows:

	Max supported size
Transmitted from	512
Received by	Firmware 11417 and earlier: 512 Firmware 11418 and newer: 1024

Enable incoming TCP connections

TCP establishes virtual connections between the XBee Smart Modem and other devices. You can enable the XBee Smart Modem to listen for incoming TCP connections. Listen means waiting for a connection request from any remote TCP and port.

The XBee Smart Modem only supports incoming TCP and UDP connections as configured in IP (IP Protocol), TLS is not supported.

Enable incoming connections in XCTU

1. Set AP (API Enable) to **Transparent Mode [0]** or **API Mode**. You can use either API mode with escapes or without escapes.
2. Set **IP** to **TCP [1]** or **UDP [0]**.
3. Set C0 (Source Port) to the value of the TCP port that the device listens on.
4. Click the **Write** button .

Enable incoming connections in MicroPython

When you enable incoming connections in MicroPython (set AP (API Enable) to **MicroPython REPL [4]**), note that the port and protocol are specified in the MicroPython code. No extra steps are needed.

API mode behavior for outgoing TCP and TLS connections

To initiate an outgoing TCP or TLS connection to a remote host, send a Transmit (TX) Request: IPv4 - 0x20 frame to the XBee Smart Modem's serial port specifying the destination address and destination port for the remote host; the data is optional and the source port is **0**.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

If the connection is disconnected at any time, send a Transmit TX Request frame to trigger a new connection attempt.

To send data over this connection use the Transmit (TX) Request: IPv4 - 0x20.

The device sends a Transmit (TX) Status - 0x89 frame in reply to the Transmit TX Request indicating the status of the request. A status of **0** indicates the connection and/or data was successful, a value of 0x32 indicates a temporary Resource Error (see Socket limits in API mode), and other values indicates a failure.

Any data received on the connection is sent out the XBee Smart Modem's serial port as a Receive RX frame.

A connection is closed when:

- The remote end closes the connection.
- No data is sent or received for longer than the socket timeout set by TM (IP Client Connection Timeout).
- A Transmit TX Request is sent with the CLOSE flag set.

API mode behavior for outgoing UDP data

To send a UDP datagram to a remote host, send a Transmit (TX) Request: IPv4 - 0x20 frame to the XBee Smart Modem's serial port specifying the destination address and destination port of the remote host. If you use a source port of **0**, the device creates a new socket for the purpose of sending to the remote host. The XBee Smart Modem supports a finite number of sockets, so if you need to send to many destinations:

1. The socket must be closed after use.

   or

2. You must use the socket specified by the C0 (Source Port) setting.

To use the socket specified by the **C0** setting, in the Transmit TX request frame use a source port that matches the value configured for the **C0** setting.

The device sends a Transmit (TX) Status - 0x89 frame in reply to the Transmit TX Request to indicate the status of the request. A status of **0** indicates the connection and/or data was successful, a value of 0x32 indicates a temporary Resource Error (see Socket limits in API mode), and other values indicates a failure.

Any data received on the UDP socket is sent out the XBee Smart Modem's serial port as a Receive (RX) Packet: IPv4 - 0xB0 frame.

A UDP socket is closed when:

- No data has been sent or received for longer than the socket timeout set by TM (IP Client Connection Timeout).
- A transmit TX Request is sent with the CLOSE flag set.

API mode behavior for incoming TCP connections

For incoming connections and data in API mode, the XBee Smart Modem uses the C0 (Source Port) and IP (IP Protocol) settings to specify the listening port and protocol used. The XBee Smart Modem does not currently support the TLS protocol for incoming connections.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

When the **IP** setting is TCP the XBee Smart Modem allows multiple incoming TCP connections on the port specified by the **C0** setting. Any data received on the connection is sent out the XBee Smart Modem's serial port as a Receive (RX) Packet: IPv4 - 0xB0 frame.

To send data from the device over the connection, use the Transmit (TX) Request: IPv4 - 0x20 frame with the corresponding address fields received from the Receive RX frame. In other words:

- Take the source address, source port, and destination port fields from the Receive (RX) frame and use those respectively as:
- The destination address, destination port, and source port fields for the Transmit (TX) Request frame.

A connection is closed when:

- The remote end closes the connection.
- No data has been sent or received for longer than the socket timeout set by TS (IP Server Connection Timeout).
- A Transmit (TX) Request frame is sent with the CLOSE flag set.

API mode behavior for incoming UDP data

When the IP (IP Protocol) setting is UDP, any data sent from a remote host to the XBee Smart Modem's network port specified by the C0 (Source Port) setting is sent out the XBee Smart Modem's serial port as a Receive (RX) Packet: IPv4 - 0xB0 frame.

To send data from the XBee Smart Modem to the remote destination, use the Transmit (TX) Request: IPv4 - 0x20 frame with the corresponding address fields received from the Receive RX frame. In other words take the source address, source port, and destination port fields from the Receive (RX) frame and use those respectively as the destination address, destination port, and source port fields for the Transmit (TX) Request frame.

Transparent mode behavior for outgoing TCP and TLS connections

For Transparent mode, the IP (IP Protocol) setting specifies the protocol and the DL (Destination Address) and DE (Destination port) settings specify the destination address used for outgoing data (UDP) and outgoing connections (TCP and TLS).

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

To initiate an outgoing TCP or TLS connection to a remote host, send data to the XBee Smart Modem's serial port. If CI (Protocol/Connection Indication) reports a value of **0**, then the connection was successfully established, otherwise the value of **CI** indicates why the connection attempt failed. Any data received over the connection is sent out the XBee Smart Modem's serial port.

A connection is closed when:

- The remote end closes the connection.
- No data has been sent or received for longer than the socket timeout set by TM (IP Client Connection Timeout).
- You make and apply a change to the **IP**, **DL**, or **DE**.

Transparent mode behavior for outgoing UDP data

To send outgoing UDP data to a remote host, send data to the XBee Smart Modem's serial port. If CI (Protocol/Connection Indication) reports a value of **0**, the data was successfully sent; otherwise, the value of **CI** indicates why the data failed to be sent.

The RO (Packetization Timeout) setting provides some control in how the serial data gets packetized before being sent to the remote host. The first send opens up a UDP socket used to send and receive data. Any data received by this socket is sent out the XBee Smart Modem's serial port.

Note Set **RO** to **FF** for realtime typing by humans. Also, see TD (Text Delimiter).

Transparent mode behavior for incoming TCP connections

The C0 (Source Port) and IP (IP Protocol) settings specify the listening port and protocol used for incoming connections (TCP) and incoming data (UDP) in Transparent mode. TLS is not currently supported for incoming connections.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

When the **IP** setting is TCP and there is no existing connection to or from the XBee Smart Modem, the device accepts one incoming connection. Any data received on the connection is sent out the XBee Smart Modem's serial port. Any data sent to the XBee Smart Modem's serial port is sent over the connection. If the connection is disconnected, it discards pending data.

Transparent mode behavior for incoming UDP connections

When the IP (IP Protocol) setting is UDP any data sent from a remote host to the XBee Smart Modem's network port specified by C0 (Source Port) is sent out the XBee Smart Modem's serial port. Any data sent to the XBee Smart Modem's serial port is sent to the network destination specified by the DL (Destination Address) and DE (Destination port) settings. If the **DL** and **DE** settings are unspecified or invalid, the XBee Smart Modem discards data sent to the serial port.

Extended Socket frames

The XBee Cellular product line includes a set of Extended Socket frames. You can use these frames in applications where the existing frames (Transmit Request (0x20), TLS Transmit (0x23) and Receive (0xB0)) limit the possibilities for an application.

You can use Extended Socket frames to do the following:

- Multiple simultaneous connections can be made to the same port on the same host. For example, you can overlap simultaneous HTTP requests.
- Immediate unsolicited notification of changes in socket status. This allows an application to react to a server-side socket closure rather than relying on an implicit connection to be re-established for continuing communication.
- A generalized mechanism for per-socket option selection. Currently used for TLS profile selection. Previously this required a unique frame, as options are added, this allows combinations of choices.
- Allow DNS look up during the connection process rather than a separate step.

In addition, for diagnostic purposes, you can use the Socket Info (SI) AT command to retrieve information regarding all open sockets currently active in the system. This can be queried during development or used by an application to confirm or refresh information during execution.

Note Sockets opened with the Extended Socket frames cannot be used with the legacy frames (Transmit Request (0x20), TLS Transmit (0x23) and Receive (0xB0)), nor vice versa.

For a list of the socket frames, see Available Extended Socket frames.

Examples

In the examples below the Frame IDs in all frames are set to 1 for simplicity. Socket IDs in all frames after the Socket Create are hard-coded to 0 as well. If you wish to use the example repeatedly the XBee should be rebooted between attempts.

We recommend the use of the XCTU frame generator for experimentation with frames during development. Paste the provided frame content directly into the **Add API frame to list** window in XCTU to follow along manually.

Extended Socket example: Single HTTP Connection

Extended Socket example: UDP

Extended Socket example: TCP Listener

Available Extended Socket frames

Note For information about all frames, see API frames.

Extended Socket example: Single HTTP Connection

This example demonstrates a complete request with an HTTP server. It fetches a random fact about a number from a web services API offered by the website http://numbersapi.com.

Note Digi is not affiliated with numbersapi.com and the example is for education only.

Send a Socket Create frame

Note To adapt this example for an HTTPS server, change **Protocol** below to 0x04 (TLS) and optionally use the Socket Option frame to specify a TLS profile.

Field	Value
Frame type	0x40 (Socket Create)
Frame ID	0x01
Protocol	0x01 (TCP)

Socket Create frame data:

```
7E 00 03 40 01 01 BD
```

Receive a Socket Create response

The XBee responds to the Socket Create request with a response. The response contains the socket ID assigned. In this example, the socket ID is 0.

Field	Value
Frame type	0xC0 (Socket Create Response)
Frame ID	0x01
Socket ID	0x00
Status	0x00 (Success)

Socket Create Response received from XBee:

```
7E 00 04 C0 01 00 00 3E
```

Send Socket Connect

This examples uses the "string" destination address type to have the XBee perform DNS look-up during the connection process.

Note To adapt this example for TLS, use destination port 0x01 0xbb (decimal 443). Be aware that many HTTPS servers use SNI (Server Name Identification) which is not currently supported.

Field	Value
Frame type	0x42 (Socket Create Response)
Frame ID	0x01
Socket ID	0x00
Destination Port	0x00 0x50 (80 decimal, HTTP)
Destination Address Type	0x01 (String)
Destination Address	numbersapi.com

Socket Connect frame data:

```
7E 00 14 42 01 00 00 50 01 6E 75 6D 62 65 72 73 61 70 69 2E 63 6F 6D C8
```

Receive a Socket Connect Response

The request to connect is immediately acknowledged with a response. However, it is not permitted to proceed transmitting data until the next stage, after a Socket Status frame has been received indicating success.

Field	Value
Frame type	0xC2 (Socket Connect Response)
Frame ID	0x01
Socket ID	0x00
Status	0x00 (Success)

Socket Connect Response received from XBee:

```
7E 00 04 C2 01 00 00 3C
```

Receive a Socket Status

The socket has been fully established when a Socket Status frame is received with the connected status after the socket has connected.

Field	Value
Frame type	0xCF (Socket Status)
Socket ID	0x00
Status	0x00 (Connected)

Socket Status received from XBee with connected status:

```
7E 00 03 CF 00 00 30
```

Send HTTP Request using Socket Send frame

The request uses the "Connection: close" header to have the server close the connection on request completion. This allows the example to demonstrate the Socket Status reporting of a close by the peer.

Field	Value
Frame type	0x44 (Socket Status)
Frame ID	0x01
Socket ID	0x00
Transmit Options	0x00
Data	GET /random/trivia HTTP/1.1 Host: numbersapi.com Connection: close

Socket Send frame data:

```
7E 00 4C 44 01 00 00 47 45 54 20 2F 72 61 6E 64 6F 6D 2F 74 72 69 76 69 61 20 48
54 54 50 2F 31 2E 31 0D 0A 48 6F 73 74 3A 20 6E 75 6D 62 65 72 73 61 70 69 2E 63
6F 6D 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D 0A 0D 0A B6
```

Receive TX Status

Extended sockets use the existing TX Status frame (0x89) to report acceptance of the data for transmit.

Field	Value
Frame type	0x89 (TX Status)
Frame ID	0x01
Status	0x00 (Success)

TX Status received from XBee data:

```
7E 00 03 89 01 00 75
```

Receive one or more Receive Data frames

The server will respond with an interesting fact about a number. The following information is a sample response. Multiple frames may be needed to contain the full response content depending on size and network conditions.

Field	Value
Frame type	0xCD (Socket Receive)
Frame ID	0x00
Socket ID	0x00
Status	0x00
Payload	HTTP/1.1 200 OK Server: nginx/1.4.6 (Ubuntu) Date: Thu, 18 Jul 2019 16:13:47 GMT Content-Type: text/plain; charset="UTF-8"; charset=utf-8 Content-Length: 53 Connection: close X-Powered-By: Express Access-Control-Allow-Origin: * Access-Control-Allow-Headers: X-Requested-With X-Numbers-API-Number: 270 X-Numbers-API-Type: trivia Pragma: no-cache Cache-Control: no-cache Expires: 0 270 is the average number of days in human pregnancy.

Receive Data received from XBee containing web service response:

```
7E 01 C5 CD 00 00 00 48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D 0A 53 65 72
76 65 72 3A 20 6E 67 69 6E 78 2F 31 2E 34 2E 36 20 28 55 62 75 6E 74 75 29 0D 0A
44 61 74 65 3A 20 54 68 75 2C 20 31 38 20 4A 75 6C 20 32 30 31 39 20 31 36 3A 31
33 3A 34 37 20 47 4D 54 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 74 65 78
74 2F 70 6C 61 69 6E 3B 20 63 68 61 72 73 65 74 3D 22 55 54 46 2D 38 22 3B 20 63
68 61 72 73 65 74 3D 75 74 66 2D 38 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74
68 3A 20 35 33 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D 0A 58
2D 50 6F 77 65 72 65 64 2D 42 79 3A 20 45 78 70 72 65 73 73 0D 0A 41 63 63 65 73
73 2D 43 6F 6E 74 72 6F 6C 2D 41 6C 6C 6F 77 2D 4F 72 69 67 69 6E 3A 20 2A 0D 0A
41 63 63 65 73 73 2D 43 6F 6E 74 72 6F 6C 2D 41 6C 6C 6F 77 2D 48 65 61 64 65 72
73 3A 20 58 2D 52 65 71 75 65 73 74 65 64 2D 57 69 74 68 0D 0A 58 2D 4E 75 6D 62
65 72 73 2D 41 50 49 2D 4E 75 6D 62 65 72 3A 20 32 37 30 0D 0A 58 2D 4E 75 6D 62
65 72 73 2D 41 50 49 2D 54 79 70 65 3A 20 74 72 69 76 69 61 0D 0A 50 72 61 67 6D
61 3A 20 6E 6F 2D 63 61 63 68 65 0D 0A 43 61 63 68 65 2D 43 6F 6E 74 72 6F 6C 3A
20 6E 6F 2D 63 61 63 68 65 0D 0A 45 78 70 69 72 65 73 3A 20 30 0D 0A 0D 0A 32 37
30 20 69 73 20 74 68 65 20 61 76 65 72 61 67 65 20 6E 75 6D 62 65 72 20 6F 66 20
64 61 79 73 20 69 6E 20 68 75 6D 61 6E 20 70 72 65 67 6E 61 6E 63 79 2E 8B
```

Receive Socket Status indicating closed connection

Finally, due to the "Connection" header in the request, the server should remotely close the connection.

Field	Value
Frame type	0xCF (TX Status)
Socket ID	0x00
Status	0x07 (Connection lost)

Example Socket Status received from XBee indicating connection lost:

```
7E 00 03 CF 00 07 29
```

When Socket Status indicating a connection close is received, the socket ID will have been de-allocated by the XBee and no further operations are possible or necessary using that ID.

Extended Socket example: UDP

UDP is connection-less, so this example demonstrates that a Socket Connect frame is not required to begin communication and that multiple peers can be used with a single socket.

Send a Socket Create frame

Field	Value
Frame type	0x40 (Socket Create)
Frame ID	0x01
Protocol	0x00 (UDP)

UDP Socket Create frame data:

```
7E 00 03 40 01 00 BE
```

Receive a Socket Create response

Field	Value
Frame type	0xC0 (Socket Create Response)
Frame ID	0x01
Socket ID	0x00
Status	0x00 (Success)

Socket Create Response received from XBee:

```
7E 00 04 C0 01 00 00 3E
```

Bind local source addres

The bind/listen operation is necessary prior to transmit in order to assign a known source address to all data sent from this socket.

Field	Value
Frame type	0x46 (Socket Bind/Listen)
Frame ID	0x01
Socket ID	0x00
Source Port	0x12 0x34

Socket Bind/Listen frame data:

```
7E 00 05 46 01 00 12 34 72
```

Receive Bind/Listen Response

The XBee generates a response indicating the status of the request to bind the requested port.

Field	Value
Frame type	0xC6 (Socket Bind/Listen Response)
Frame ID	0x01
Socket ID	0x00
Status	0x00 (Success)

Socket Bind/Listen Response received from XBee:

```
7E 00 04 C6 01 00 00 38
```

Send to Digi echo server

Digi hosts a server at 52.43.121.77 port 10001 which echos all UDP traffic sent to it.

Field	Value
Frame type	0x45 (Socket SendTo)
Frame ID	0x01
Socket ID	0x00
Destination Address	0x34 0x2B 0x79 0x4D (52.43.121.77)
Destination Port	0x27 0x11 (decimal 10001)
Transmit Options	0x00
Payload	echo this

Socket SendTo frame data:

```
7E 00 13 45 01 00 34 2B 79 4D 27 11 00 65 63 68 6F 20 74 68 69 73 E5
```

Receive TX Status

Extended sockets use the existing TX Status frame (0x89) to report acceptance of the data for transmit.

Field	Value
Frame type	0x89 (TX Status)
Frame ID	0x01
Status	0x00 (Success)

TX Status received from XBee:

```
7E 00 03 89 01 00 75
```

Receive echoed data

When the response from the server is sent back, the XBee provides it using a Socket Receive From frame.

Field	Value
Frame type	0xCE (Socket Receive From)

Field	Value
Frame ID	0x00
Socket ID	0x00
Source address	0x34 0x2B 0x79 0x4D (52.43.121.77)
Source Port	0x27 0x11 (decimal 10001)
Status	0x00 (Success)
Payload	echo this

Socket ReceiveFrom received from XBee, containing echoed data:

```
7E 00 13 CE 00 00 34 2B 79 4D 27 11 00 65 63 68 6F 20 74 68 69 73 5D
```

Send to Digi time server

Digi hosts a server at 54.43.121.77 port 10002 which will reply with the time when it receives a packet.

Field	Value
Frame type	0x45 (Socket SendTo)
Frame ID	0x01
Socket ID	0x00
Destination Address	0x34 0x2B 0x79 0x4D (52.43.121.77)
Destination Port	0x27 0x12 (decimal 10002)
Transmit Options	0x00
Payload	0x20 (ASCII space, any value should do)

Socket SendTo time server frame data:

```
7E 00 0B 45 01 00 34 2B 79 4D 27 12 00 20 3B
```

Receive TX Status

This is exactly the same as the previous transmission to the echo server on success.

Receive daytime value

When the response from the server is sent back, the XBee will provide it using a Socket Receive From frame.

Field	Value
Frame type	0xCE (Socket Receive From)
Frame ID	0x00
Socket ID	0x00
Source address	0x34 0x2B 0x79 0x4D (52.43.121.77)
Source Port	0x27 0x12 (decimal 10002)
Status	0x00 (Success)
Payload	<current UTC time>

Socket Receive From frame received from XBee containing time data:

```
7E 00 1E CE 00 00 34 2B 79 4D 27 12 00 32 30 31 39 2D 30 37 2D 31 38 20 31 38 3A
35 32 3A 34 33 0A 08
```

Close the socket

When the socket is no longer needed it should be closed to return resources to the system.

Field	Value
Frame type	0x43 (Socket Close)
Frame ID	0x01
Status	0x00

Socket Close frame data:

```
7E 00 03 43 01 00 BB
```

Receive close response

Finally, the XBee indicates the socket has been closed with a Socket Close Response frame.

Field	Value
Frame type	0xC3 (Socket CloseResponse)
Frame ID	0x01
Socket ID	0x00
Status	0x00 (Success)

Socket Close Response received from XBee:

```
7E 00 04 C3 01 00 00 3B
```

Extended Socket example: TCP Listener

The following example demonstrates setting up a TCP listener on the XBee Cellular and interacting with incoming connections. It will open up a listener socket on a given port and then receive data from a client.

Note The module must either have a public IP or a be on a private network in order to be accessible as a server (listener).

Send a Socket Create frame

Note The XBee Cellular does not support incoming TLS sockets.

Field	Value
Frame type	0x40 (Socket Create)
Frame ID	0x01
Protocol	0x01 (TCP)

Socket Create frame data:

```
7E 00 03 40 01 01 BD
```

Receive a Socket Create response

The response contains the socket ID assigned. This example assumes zero.

Field	Value
Frame type	0xC0 (Socket Create Response)
Frame ID	0x01
Socket ID	0x00
Status	0x00 (Success)

Socket Create Response received from XBee:

```
7E 00 04 C0 01 00 00 3E
```

Designate the socket as a listener

The Socket Bind/Listen Frame takes the socket ID from the socket create response and a source port that the socket will then listen on. In this example port 10001 is used.

Field	Value
Frame type	0x46 (Socket Listen)
Frame ID	0x01
Socket ID	0x00
Source Port	0x2711 (10001)

Socket Bind/Listen frame data:

```
7E 00 05 46 01 00 27 11 80
```

Receive a Socket Bind/Listen Response

The Socket Bind/Listen Response contains a Status. A Status of zero is a success and any other value is an error.

Field	Value
Frame type	0xC6 (Socket Listen)
Frame ID	0x01
Socket ID	0x00
Status	0x00 (Success)

Socket Bind/Listen frame received from XBee:

```
7E 00 04 C6 01 00 00 38
```

Making a connection to the listener socket

The IP of the XBee can be acquired through the MY at command.

```
ATMY
172.20.1.235
```

Using an external tool like netcat, a connection can be made to the given address.

```
nc -p 10001 172.20.1.235 10001
Hello XBee!
```

After the connection has been made, the XBee outputs a Socket New IPv4 Client frame indicating the presence of a new client connection. It contains the listener's socket ID and the new Client Socket ID along with the connection's remote address information.

Field	Value
Frame type	0xCC (Socket New IPv4 Client)

Field	Value
Socket ID	0x00
Client Socket ID	0x01
Remote Address	0x0A 0x0A 4A 9D
Remote Port	0x27 0x11

Socket New IPv4 Client frame:

```
7E 00 09 CC 00 01 0A 0A 4A 9D 27 11 FF
```

Note XBee Cellular Cat-1 variants require data to be sent before the connection is presented. Other variants present the connection as soon as it is made.

Receiving Data from the new socket

After the connection is established, data received from the new socket is contained in a Socket Receive frame just like any other TCP socket.

Field	Value
Frame type	0xCD (Socket Status)
Frame ID	0x01
Socket ID	0x01
Status	0x00
Payload	Hello XBee!

Receive Data indicating data from remote TCP peer:

```
7E 00 10 CD 00 01 00 48 65 6C 6C 6F 20 58 42 65 65 21 0A 8E
```

Receive a Socket Status indicating closed connection

You may close the client socket remotely which elicits a Socket Status with a Status of 0x07.

Field	Value
Frame type	0xCF (Socket Status)
Socket ID	0x01
Status	0x07 (Connection lost)

Socket Status received from XBee indicating connection lost:

```
7E 00 03 CF 01 07 28
```

When a Socket Status indicating a connection close is received, the socket ID will have been de-allocated by the XBee and no further operations are possible or necessary using that ID.

Transport Layer Security (TLS)

For detailed information about using MicroPython on the XBee Smart Modem refer to the *Digi MicroPython Programming Guide*.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Specifying TLS keys and certificates

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

These AT commands, when used together, let you interact with TLS features: ATFS (File System), TL (TLS Protocol Version), IP (IP Protocol), $0 (TLS Profile 0), $1 (TLS Profile 1), and $2 (TLS Profile 2). The format of the **$** commands is:

AT$<num>[<ca_cert>];[<client_cert>];[<client_key>]

Where:

- **num**: Profile index. Index zero is used for Transparent mode connections and TLS connections using Transmit (TX) Request: IPv4 - 0x20.
- **ca_cert**: (optional) Filename of a file in the **certs/** directory. Indicates the certificate identifying a trusted root certificate authority (CA) to use in validating servers. If **ca_cert** is empty the server certificate will not be authenticated. This must be a single root CA certificate. The modules do not allow a non-self signed certificate to work, so intermediate CAs are not enough.

Note This module will only work with the originating end of chain Root CA, so you will need to use that one. For example, with Amazon web services ATS endpoints Digi recommends that you use the Starfield Services Root Certificate from https://ssl-ccp.secureserver.net/repository/sf-class2-root.crt. The intermediate "root CAs" from Amazon will not work. You will need the actual end of chain certificate.

- **client_cert**: (optional) Filename of a file in the **certs/** directory. Indicates the certificate presented to servers when requested for client authentication. If **client_cert** is empty no certificate is presented to the server should it request one. This may result in mutual authentication failure.
- **client_key**: (optional) Filename of a file in the **certs/** directory. Indicates the private key matching the public key contained in **client_cert**. This should be a secure file uploaded with ATFS XPUT filename. This should always be provided if **client_cert** is provided and match the certificate or client authentication will fail.

The default value is "**;;**". This default value preserves the legacy behavior by allowing the creation of encrypted connections that are confidential but not authenticated.

To specify a key stored outside of **certs/**, you can either use a relative path, for example **../server.pem** or an absolute path starting with **/flash**, for example **/flash/server.pem**. Both examples refer to the same file.

It is not an error at configuration time to name a file that does not yet exist. An error is generated if an attempt to create a TLS connection is made with improper settings.

- Files specified should all be in PEM format, not DER.
- Upload private keys securely with ATFS XPUT filename.
- Certificates can be uploaded with ATFS PUT filename as they are not sensitive. It is not possible to use ATFS GET filename to **GET** them if they have been securely uploaded.

To authenticate a server not participating in a public key infrastructure (PKI) using CAs, the server must present a self-signed certificate. That certificate can be used in the **ca_cert** field to authenticate that single server.

There are effectively three levels of authentication provided depending on the parameters provided

1. No authentication: None of the parameters are provided, this is the default value. With this configuration identity is not validated and a man in the middle (MITM) attack is possible.
2. Server authentication: Only **ca_cert** is provided. Only the servers identity is checked
3. Mutual authentication: All items are provided and both sides are assured of the identity of their peer

It is not possible to only have client authentication.

Transparent mode and TLS

Transparent mode connections made when IP (IP Protocol) = **4** (TLS) are made using the configuration specified by $0 (TLS Profile 0).

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

API mode and TLS

On the Transmit (TX) Request: IPv4 - 0x20 frame, when you specify protocol **4** (TLS), the profile configuration specified by $0 (TLS Profile 0) is used to form the TLS connection. Tx Request with TLS Profile - 0x23 lets you choose the IP setting for the serial data.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Key formats

The RSA PKCS#1 format is the only common format across XBee Cellular device variants. You can identify a PKCS#1 key file by the presence of **BEGIN RSA PRIVATE KEY** in the file header.

Digi's implementation does not support encrypted keys, we use file system encryption to protect the keys at rest in the system.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Certificate formats

For SARA-R410 cellular components used in the XBee 3 Cellular LTE-M Global Smart Modem, if the server certificate has a Common Name (CN) that is greater than 31 characters the SSL connection fails.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Certificate limitations

The XBee Smart Modem only supports certificate files that contain a single certificate in them.

The implications of this are:

- For client certificate files (for example when client authentication is required):
  - Self-signed certificates will work.
  - Certificates signed by the root CA will work, because the root CA can be omitted per RFC 5246. The root certificate authority may be omitted from the chain, under the assumption that the remote end must already possess it in order to validate it in any case.
  - Certificate chains that include a intermediate CA are problematic. To work around this the client's certificate chain has to be supplied to the server outside of the connection.
- For server certificate files (when server authentication is required) this is not a problem unless the client is expected to connect to multiple servers that are using different self signed certificates or are using certificate chains that are signed by different root CA certificates. To work around this you have to change the certificates before making the connection, or in the case of API mode specify a different authentication profile.

Cipher suites

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

The only documented shared suites between the XBee 3 Cellular LTE Cat 1 Smart Modem and the XBee 3 Cellular LTE-M Global Smart Modem are:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA

For the u-blox SARA-R410 and SARA-U201 cellular components:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

Server Name Indication (SNI)

We do not currently support SNI. Therefore servers which use SNI to present certificates based on client provided host data may be unable to establish the expected connections.

Secure the connection between an XBee and Remote Manager with server authentication

The XBee devices can secure the TLS connection to Digi Remote Manager. The default configuration provides confidentiality of the communication but is not able to authenticate the server without a certificate being provided.

You should follow the procedure below to add the necessary certificate if server authentication is needed.

Step 1: Get the certificate

1. Navigate to the **Firmware Updates** section of the Digi XBee 3 Cellular LTE-M/NB-IoT support page.
2. Click **Remote Manager TLS Public Certificate** to download the certificate .zip file.
3. Unzip the .zip file.
4. Calculate the SHA-256 hash to verify that the file is correct. The correct file will have an SHA-256 hash of:

   `33d91e18668b0d8a9ec59c5f9f312c53ca2884adaa62337839e5495c26d2d64c`

Step 2: Configure device

You should confirm that the default settings are correct. You can use either Remote Manager or XCTU to verify these settings and place the certificate file in the correct location.

1. Verify the following settings:

Setting	Value
DO	Bit 0 (mask 0x1) must be set. This enables the use of Digi Remote Manager within the firmware.
MO	Bit 1 (mask 0x2) must be set. When this value is set the Remote Manager TCP connection will be secured with TLS.
$D	By default will contain the value */flash/cert/digi-remote-mgr.pem*. This is the file system location where the firmware will look for the certificate to use.

2. Use XCTU or Remote Manager to place the downloaded and unzipped certificate file in the location specified in the **$D** command.

Step 3: Verify that authentication is being performed

The next TCP connection to Remote Manager should only succeed if the server can be authenticated using the provided certificate. You can confirm that the server has been authenticated.

1. Cause an active connection to Remote Manager. For example, you could set bit 0 for the **MO** command. Make sure that you do not clear bit 1.
2. After a short wait you should be able to see the device as connected in Remote Manager.
   a. Log in to Remote Manager.
   b. Click **Device Management**.
   c. Locate the device in the device list and verify that the connection icon in the left column is blue and the hover tool tip says "Connected".
3. When the device is connected to Remote Manager, the **DI** command can take on any of the three values shown below, based on the security level of the connection. Verify the that the **DI** command is set to **6** to verify that the server was correctly authenticated.

- **0**: Connected without TLS
- **5**: Connected with TLS but without authentication
- **6**: Connected with TLS and with authentication

AT commands

Special commands

The following commands are special commands.

AC (Apply Changes)

Immediately applies new settings without exiting Command mode.

Applying changes means that the device re-initializes based on changes made to its parameter values. Once changes are applied, the device immediately operates according to the new parameter values.

This behavior is in contrast to issuing the **WR** (Write) command. The **WR** command saves parameter values to non-volatile memory, but the device still operates according to previously saved values until the device is rebooted or you issue the **CN** (Exit AT Command Mode) or **AC** commands.

Parameter range

N/A

Default

N/A

FR (Force 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.

Note Digi recommends shutting down the cellular component before resetting or rebooting the device to allow the cellular module to detach from the network. The cellular component can be shut down by issuing the SD command.

Parameter range

N/A

Default

N/A

RE (Restore Defaults)

Restore device parameters to factory defaults.

The **RE** command does not write restored values to non-volatile (persistent) memory. Issue the **WR** (Write) command after issuing the **RE** command to save restored parameter values to non-volatile memory.

Parameter range

N/A

Default

N/A

SD (Shutdown)

Shuts down the device. When the shut down process is complete, the device returns **OK**. After the device responds **OK**, you can safely remove power from the device.

If the radio can't be fully shut down within two minutes, the device returns **ERROR**.

You can verify the state of the device using the AI command. After you issue the **SD** command and a response has been returned (either **OK** or **ERROR**), issue the AI command. If the shutdown was successful, **2D** is returned.

Parameter range

Parameter	Description
0	Shuts down the device. When the shut down process is complete, the device returns **OK**.
1	Reboots the module when the shut down completes.

Default

N/A

WR (Write)

Writes parameter values to non-volatile memory so that parameter modifications persist through subsequent resets.

Note Once you issue a **WR** command, do not send any additional characters to the device until after you receive the **OK** response.

Parameter range

N/A

Default

N/A

Cellular commands

The following AT commands are cellular configuration and data commands.

PH (Phone Number)

Reads the SIM card phone number.

Parameter range

N/A

Default

Set by the cellular carrier via the SIM card

S# (ICCID)

Reads the Integrated Circuit Card Identifier (ICCID) of the inserted SIM.

Parameter range

N/A

Default

Set by the SIM card

IM (IMEI)

Reads the device's International Mobile Equipment Identity (IMEI).

Parameter range

N/A

Default

Set in the factory

II (Subscriber identity)

Reads the IMSI (International Mobile Subscriber Identity) from the SIM inserted into the module.

Parameter range

N/A

Default

N/A

MN (Operator)

Reads the network operator on which the device is registered.

Parameter range

N/A

MV (Modem Firmware Version)

Read the firmware version string for cellular component communications. See the related VR (Firmware Version) command.

Parameter range

N/A

Default

Set in the currently loaded firmware

DB (Cellular Signal Strength)

Reads the absolute value of the current signal strength to the cell tower in dB. If **DB** is blank, the XBee Smart Modem has not received a signal strength from the cellular component.

DB0 only updates when the modem is registered with the cellular tower. It is updated periodically, and not when read.

Parameter range

Parameter	Description
0	Returns the most recent, cached RSSI signal value received.
1	Returns a fresh, uncached RSSI signal value.

Returned values

0x71 - 0x33 (-113 dBm to -51 dBm) [read-only]

Default

N/A

DT (Cellular Network Time)

Reads the current network-provided local time of the XBee device, as reported by the cellular tower.

If the time is not known, the response is empty. If the radio reported the time zone, that information is included in the response.

This value is synchronized with the network at least once per hour.

Note The time is provided by the network. If the time is not what you expect, contact your network provider.

Parameter range

0 - 1

Value	Description
0	The response is the number of seconds since 2000-01-01 00:00:00, as a 32-bit number. This is the default.
1	The response is the current date and time in ISO 8601 format. If the radio reported the time zone, that information is included in the response. Example (no time zone): 2022-12-25T22:00:05 Example (with time zone): 2022-05-23T15:45:46-05:00

Note You can also send **DT**, which acts like **DT=0**.

Default

0

AN (Access Point Name)

Specifies the packet data network that the modem uses for Internet connectivity. This information is provided by your cellular network operator. After you set this value, applying changes with AC (Apply Changes) or CN (Exit Command mode) triggers a network reset.

See Network connection issues if the XBee Smart Modem is not joining the network.

Parameter range

1 - 100 ASCII characters

Default

-

CP (Carrier Profile)

Configures the cellular component to select network operator settings (RF bands, packet data configuration) for various networks.

The **1 (No Profile)** setting should be used if the module is not able to join the network because the underlying cellular modem does not have a predefined profile that supports the inserted SIM card. The **1 (No Profile)** setting does not use any predefined profiles, which forces the module to attempt to join an appropriate network based on the module's current configuration. This configuration works in conjunction with the following commands: BN (Bandmask) (NB-IoT), BM (Bandmask) (LTE-M), and N# (Preferred Network Technology).

Changes to the value only take effect on boot so a reboot or power cycle is required for any changes to become active.

Parameter range

0 - 3

Value	Description
0	Autodetect from inserted ICCID (SIM). This is the default. Setting to **0** increases the boot time.
1	No Profile
2	AT&T
3	Verizon **Note** This value should only be used with Verizon home network SIM cards. Setting the value to 3 with other SIM cards may adversely affect network registration and activity.

Default

0

BM (Bandmask) (LTE-M)

Note This command is for use with only LTE-M.

Configures the enabled 4G LTE bands for LTE-M when CP is set to **1** (No Profile).

Changes to the value only take effect on boot so a reboot or power cycle is required for any changes to become active.

Note The actual set of enabled bands will be a subset of this bit field, depending on the limitations of the cellular component.

WARNING! If this value is configured incorrectly, the XBee module may be unable to locate a tower and join the network.

Parameter range

0 - 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF (bit field)

Bit	LTE Band
0	1
...	
127	128

Example

0x080080 (bits 7 and 19) enable LTE Bands 8 and 20.

Default

0xFFFFFFFFFFFFFFFF (bands 1-64 enabled)

BN (Bandmask) (NB-IoT)

Note This command is for use with only NB-IoT.

Configures the enabled 4G LTE bands for NB-IoT when the CP command is set to **1** (No Profile). If CP is not set to **1**, the BN command will return an error.

Changes to the value only take effect on boot so a reboot or power cycle is required for any changes to become active.

Note The actual set of enabled bands will be a subset of this bit field, depending on the limitations of the cellular component.

WARNING! If this value is configured incorrectly, the XBee module may be unable to locate a tower and join the network.

Parameter range

0 - 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF (bit field)

Bit	LTE Band
0	1
...	
127	128

Examples

0x080080 (bits 7 and 19) enable NB-IoT Bands 8 and 20.

Default

0xFFFFFFFFFFFFFFFF (bands 1-64 enabled)

AM (Airplane Mode)

When set, the cellular component of the XBee Smart Modem is fully turned off and no access to the cellular network is performed or possible.

Parameter range

0 - 1

0 = Normal operation

1 = Airplane mode

Default

0

N# (Preferred Network Technology)

Allows the XBee 3 Cellular LTE-M/NB-IoT module to select the preferred network technology. A reboot or power cycle is required for any parameter changes to become active.

Note For the **N#** command configuration to take effect, the CP command must be set to **1** (No Profile). Setting the **CP** command to **1** configures the enabled 4G LTE bands for NB-IoT. If CP is not set to **1**, the N# command value is ignored.

Range

0 - 3

Parameter	Description
0	LTE-M with NB-IoT fallback.
1	NB-IoT with LTE-M fallback.
2	LTE-M only.
3	NB-IoT only.

Default

0

SQ (Reference Signal Received Quality)

Returns the Reference Signal Received Quality (RSRQ) value.

The value returned is in hex, and should be converted by the user with the following formula:

RSRQ = -(<hex_value> / 0xA)

Example: The value returned from the command is 82:

RSRQ = -(0x82 / 0xA) = -13.0 dB

Example: The value returned is A0:

RSRQ = -(0xA0 / 0xA) = -16.0 dB

If the value cannot be retrieved for some reason, such as the device is not on the network yet, an empty string with **OK** after it is returned.

Parameter range

N/A

Default

N/A

SW (Reference Signal Received Power)

Returns the Reference Signal Received Power (RSRP) value.

The value returned is in hex, and should be converted by the user with the following formula:

RSRP = -(<hex_value> / 0xA)

Example: The value returned from the command is 384:

RSRP = -(0x384 / 0xA) = -90.0 dBm

Example: The value returned is A0:

RSRQ = -(0xA0 / 0xA) = -16.0 dB

If the value cannot be retrieved for some reason, such as the device is not on the network yet, an empty string with **OK** after it is returned.

Parameter range

N/A

Default

N/A

PN (SIM PIN)

Specifies the PIN when using a SIM.

This command is write-only.

Parameter range

4 to 8 ASCII digits or space character.

A value of a single space character (ASCII 0x20) acts as an empty value.

Default

0x20: A single ASCII space character that indicates there is no PIN.

PK (SIM PUK)

Specifies the PUK for unlocking a SIM. This is needed only if the wrong PIN was used and the SIM is locked out.

This command is write-only.

Parameter range

8 ASCII digits or space character

A value of a single space character (ASCII 0x20) acts as an empty value.

Default

0x20: A single ASCII space that indicates there is no PUK.

CU (Cellular user name)

Specifies the user name used when authenticating to the cellular network.

This command is write-only.

Parameter range

1 to 30 ASCII characters

A value of a single space character (ASCII 0x20) acts as an empty value.

Default

0x20: A single ASCII space that indicates there is no cellular user name.

CW (Cellular password)

Specifies the password used when authenticating to the cellular network.

This command is write-only.

Parameter range

1 to 30 ASCII characters

A value of a single space character (ASCII 0x20) acts as an empty value.

Default

0x20: A single ASCII space that indicates there is no cellular password.

OT (Operating Technology)

Reports the active technology of the current network connection.

A blank value (**OK** returned) indicates that the access technology is currently unknown.

Range

0x0 - 0xFFFF

Parameter	Description
7	LTE (M1)
9	E-UTRAN (NB1)

Default

N/A

FC (Frequency Channel Number)

Returns the EARFCN of the current cellular connection.

The EARFCN encodes the carrier frequency or frequencies that the cellular radio is using. Refer to the 3GPP specifications or various online tools or guides to determine the corresponding band number.

If the value cannot be retrieved for some reason, such as the device is not on the network, the response is empty. When in command mode and the value cannot be retrieved, **OK** is returned.

Parameter range

N/A

Default

N/A

Network commands

The following commands are network commands.

IP (IP Protocol)

Note For NB-IoT, TCP and SMS support is dependent on the network. Contact your network provider for details.

Sets or displays the IP protocol used for client and server socket connections in IP socket mode.

For TLS, the ublox SARA-R4 series AT Commands Manual provides the list of supported cipher suites under the **SSL/TLS security layer profile manager +USECPRF** heading.

Parameter range

0 - 4

Value	Description
0x00	UDP
0x01	TCP
0x02	SMS (Transparent mode)
0x03	Reserved
0x04	TLS over TCP

Default

0x01

TL (TLS Protocol Version)

Sets the TLS protocol version used for the TLS socket. If you change the **TL** value, it does not affect any currently open sockets. The value only applies to subsequently opened sockets.

Note Due to known vulnerabilities in prior protocol versions, we strongly recommend that you use the latest TLS version whenever possible.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Range

Value	Description
0x00	SSL v3
0x01	TLS v1.0
0x02	TLS v1.1
0x03	TLS v1.2

Default

0x03

$0 (TLS Profile 0)

Specifies the TLS certificate(s) to use in Transparent mode (when IP (IP Protocol) = **4**) or API mode (Transmit (TX) Request: IPv4 - 0x20 or Tx Request with TLS Profile - 0x23 with profile set to **0**).

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Format

server_cert;client_cert;client_key

Parameter range

From 1 through 127 ASCII characters.

Default

N/A

$1 (TLS Profile 1)

Specifies the TLS certificate(s) to use for Tx Request with TLS Profile - 0x23 transmissions with profile set to **1**.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Format

server_cert;client_cert;client_key

Parameter range

From 1 through 127 ASCII characters.

Default

N/A

$2 (TLS Profile 2)

Specifies the TLS certificate(s) to use for Tx Request with TLS Profile - 0x23 transmissions with profile set to **2**.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Format

server_cert;client_cert;client_key

Parameter range

From 1 through 127 ASCII characters.

Default

N/A

TM (IP Client Connection Timeout)

The IP client connection timeout. If there is no activity for this timeout then the connection is closed. If **TM** is **0**, the connection is closed immediately after the device sends data.

If you change the **TM** value while in Transparent Mode, the current connection is immediately closed. Upon the next transmission, the **TM** value applies to the newly created socket.

If you change the **TM** value while in API Mode, the value only applies to subsequently opened sockets.

TM does not apply to explicit sockets.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Parameter range

0 - 0xFFFF [x 100 ms]

Default

0xBB8 (5 minutes)

TS (IP Server Connection Timeout)

The IP server connection timeout. If no activity for this timeout then the connection is closed. When set to **0** the connection is closed immediately after data is sent.]

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Parameter Range

10 - 0xFFFF; (x 100 ms)

Default

0xBB8 (5 minutes)

DO (Device Options)

Enables and disables special features on the XBee Smart Modem.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Bitfield

Bit	Description
0	Enable Remote Manager Controls whether Remote Manager is enabled. Digi recommends that Remote Manager remains enabled.
2	Enable USB Direct Set bit 2 to enable USB direct mode. After setting, use WR (Write) to write all values to flash and use FR (Force Reset) to reset the device. **Note** Setting P0 (DIO10/PWM0 Configuration) to **6** overrides setting **DO** bit 2.
3	Enable PSM To enable PSM, set DO (Device Options) bit 3. See Power Saving Mode (PSM).
4	Enable the Low Voltage Shutdown feature Set bit 4 to enable the Low Voltage Shutdown feature. See Low voltage shutdown.
5	Enable eDRX Set bit 5 to request the eDRX feature from the network. The requested cycle length is defined by the DX command, and the network-provided cycle length can be read using D? command, once the device is registered on the network. When connected to the network, the AI command is set to 0.

Note We strongly recommend that you clear bit 0 (Enable Remote Manager) if you set bit 3. If not, the connection that the device retains with Remote Manager causes the cellular component to spend very little time in low power, negating the value of selecting that feature.

Default

1 (Bit 0 enabled)

DX (Requested eDRX cycle length)

The eDRX cycle length (in milliseconds) that will be requested from the network if DO command bit 5 is set. For best power characteristics, you should set this to the maximum receive latency your application is designed to tolerate.

The actual value obtained is provided by the network and is generally less than the value requested. The specific legal values depend on network access technology and carrier policy. The maximum value, where data is retained by the network without loss, also depends on carrier policy and this value should be selected in light of guidance provided by your carrier.

Parameter range

N/A

Default

0xea60 (60 seconds)

D? (Network-provided eDRX cycle length)

The value currently being used as the eDRX cycle length (in milliseconds). If eDRX is not active, or registration with the network has not yet been achieved, an empty response is returned.

Parameter range

N/A

Default

N/A

W? (Network-provided eDRX Paging Time Window length)

The value currently being used as the eDRX Paging Time Window length (in milliseconds). If eDRX is not active, or registration with the network has not yet been achieved, an empty response is returned.

Note This command was added in firmware version 11417.

Parameter range

N/A

Default

N/A

Addressing commands

The following AT commands are addressing commands.

SH (Serial Number High)

The upper digits of the unique International Mobile Equipment Identity (IMEI) assigned to this device.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

N/A

SL (Serial Number Low)

The lower digits of the unique International Mobile Equipment Identity (IMEI) assigned to this device.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

N/A

MY (Module IP Address)

Reads the device's IP address. This command is read-only because the IP address is assigned by the mobile network.

In API mode, the address is represented as the binary four byte big-endian numeric value representing the IPv4 address.

In Transparent or Command mode, the address is represented as a dotted-quad string notation.

Parameter range

0- 15 IPv4 characters

Default

0.0.0.0

P# (Destination Phone Number)

Sets or displays the destination phone number used for SMS when IP (IP Protocol) = **2** while in Transparent Operating mode. Phone numbers must be fully numeric, using ASCII digits, for example: 8889991234.

P# allows international numbers with or without the **+** prefix. If you omit **+** and are dialing internationally, you need to include the proper International Dialing Prefix for your calling region, for example, 011 for the United States.

Note For information on SMS transmissions in API mode, see Transmit (TX) SMS - 0x1F.

Range

Device firmware versions...	Range
Ending in *16 or later	4 - 20 ASCII digits, including an optional **+** prefix
Ending in *11 to *15	5 - 20 ASCII digits
Ending in *10 or earlier	7 - 20 ASCII digits

Default

N/A

N1 (DNS Address)

Displays the IPv4 address of the primary domain name server.

Parameter Range

Read-only

Default

0.0.0.0 (waiting on cellular connection)

N2 (DNS Address)

Displays the IPv4 address of the secondary domain name server.

Parameter Range

Read-only

Default

0.0.0.0 (waiting on cellular connection)

DL (Destination Address)

The destination IPv4 address or fully qualified domain name used by Transparent mode.

To set the destination address to an IP address, the value must be a dotted quad, for example **XXX.XXX.XXX.XXX**.

To set the destination address to a domain name, the value must be a legal Internet host name, for example **remotemanager.digi.com**

Parameter range

0 - 128 ASCII characters

Default

0.0.0.0

OD (Operating Destination Address)

Read the destination IPv4 address currently in use by Transparent mode. The value is **0.0.0.0** if no Transparent IP connection is active.

In API mode, the address is represented as the binary four byte big-endian numeric value representing the IPv4 address.

In Transparent or Command mode, the address is represented as a dotted-quad string notation.

Parameter range

-

Default

0.0.0.0

DE (Destination port)

Sets or displays the destination IP port number used in Transparent mode.

This command reads all input as hexadecimal. All values must be entered in hexadecimal with no leading 0x. For example, the destination port 9001 has the hexadecimal value of 0x2329. The command would be entered as **ATDE 2329**.

Parameter range

0x0 - 0xFFFF

Default

0x2616

C0 (Source Port)

The IP port used to listen for incoming connections (TCP/TLS) or incoming data (UDP) when using Transparent mode or API mode with implicit sockets.

As long as a network connection is established to this port (for TCP) data received on the serial port is transmitted on the established network connection.

IP (IP Protocol) sets the protocol used.

For more information on using incoming connections, see Socket behavior.

Parameter range

0 - 0xFFFF

Value	Description
0	Disabled
Non-0	Enabled on that port

Default

0

LA (Lookup IP Address of FQDN)

Performs a DNS lookup of the given fully qualified domain name (FQDN) and outputs its IP address.

When you issue **LA** in API mode, the IP address is formatted in binary four byte big-endian numeric value. In all other cases (for example, Command mode) the format is dotted decimal notation.

Range

Valid FQDN

Default

-

NI (Node Identifier)

Stores a string identifier. The register only accepts printable ASCII data.

Parameter range

A string of case-sensitive ASCII printable characters from 0 to 20 bytes in length.

Default

One ASCII space character (0x20)

Serial interfacing commands

The following AT commands are serial interfacing commands.

BD (Baud Rate)

Sets or displays the serial interface baud rate for communication between the device's serial port and the host.

Modified interface baud rates do not take effect until the XBee Smart Modem exits Command mode or you issue AC (Apply Changes). The baud rate resets to default unless you save it with WR (Write) or by clicking the **Write module settings** button in XCTU.

The device interprets any value between 0x4B0 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: 0x1 - 0xA

Non-standard baud rates: 0x4B0 - 0x0EC400

Note On XBee 3 Cellular firmware versions ending in *13 or earlier, the minimum baud rate is 2400 and the maximum is 230400.

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	230400 b/s
0x9	460800 b/s
0xA	921600 b/s

Default

0x3 (9600 b/s)

NB (Parity)

Set or read the serial parity settings for UART communications.

Parameter range

0x00 - 0x02

Parameter	Description
0x00	No parity
0x01	Even parity
0x02	Odd parity

Default

0x00

SB (Stop Bits)

Sets or displays the number of stop bits for the UART.

Parameter range

0 - 1

Value	Description
0	One (1) stop bit.
1	Two (2) stop bits.

Default

0

RO (Packetization Timeout)

Set or read the number of character times of inter-character silence required before transmission begins when operating in Transparent mode.

Set **RO** to **0** to transmit characters as they arrive instead of buffering them into one RF packet.

Parameter range

0 - 0xFF (x character times)

Default

3

TD (Text Delimiter)

The ASCII character used as a text delimiter for Transparent mode. When you select a character, information received over the serial port in Transparent mode is not transmitted until that character is received. To use a carriage return, set to **0xD**. Set to zero to disable text delimiter checking.

Parameter range

0 - 0xFF

Default

0x0

FT (Flow Control Threshold)

Set or display the flow control threshold.

The device de-asserts $\overline{\text{CTS}}$ when **FT** bytes are in the UART receive buffer.

Parameter range

0x9D - 0x82D

Default

0x681

AP (API Enable)

Enables the frame-based application programming interface (API) mode.

The API mode setting. The device can format the RF packets it receives into API frames and send them out the UART. When API is enabled the UART data must be formatted as API frames because Transparent mode is disabled. See Modes for more information.

Parameter range

0x00 - 0x05

Parameter	Description
0x00	API disabled (operate in Transparent mode)
0x01	API enabled
0x02	API enabled (with escaped control characters)
0x03	N/A
0x04	MicroPython REPL
0x05	Bypass mode (DEPRECATED. For diagnostic use only)

Default

0

IB (Cellular Component Baud Rate)

Note Digi does not recommend using bypass mode. You should use USB Direct mode instead.

Sets the serial interface baud rate for communication between the XBee CPU and the cellular component when in bypass mode. You can set bypass mode by setting the AP command to **5**. You must configure the cellular modem to use the same baud rate (AT+IPR) prior to changing this setting.

Parameter range

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	230400 b/s
0x9	460800 b/s
0xA	921600 b/s

Default

0x9 (460800 baud)

I/O settings commands

The following AT commands are I/O settings commands.

D0 (DIO0/AD0)

Sets or displays the DIO0/AD0 configuration (pin 20).

Parameter range

0, 2 - 6

Parameter	Description
0	Disabled
1	N/A

Parameter	Description
2	Analog input
3	Digital input
4	Digital output, default low
5	Digital output, default high
6	Cellular component mirror

Default

0

D1 (DIO1/AD1)

Sets or displays the DIO1/AD1 configuration (pin 19).

Parameter range

0 - 6

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{ATTN}}$
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high
6	I^{2}C SCL

Default

0

D2 (DIO2/AD2)

Sets or displays the DIO2/AD2 configuration (pin 18).

Parameter range

0 - 5, 6

	Description
0	Disabled
1	SPI_CLK

	Description
2	Analog input
3	Digital input
4	Digital output, default low
5	Digital output, default high
6	N/A
7	MicroPython UART1 RTS

Default

0

D3 (DIO3/AD3)

Sets or displays the DIO3/AD3 configuration (pin 17).

Parameter range

0 - 5, 7

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{SSEL}}$
2	Analog input
3	Digital input
4	Digital output, default low
5	Digital output, default high
6	N/A
7	MicroPython UART1 CTS

Default

0

D4 (DIO4)

Sets or displays the DIO4 configuration (pin 11).

Parameter range

0, 1, 3 - 5, 7

Parameter	Description
0	Disabled
1	SPI_MOSI
2	N/A
3	Digital input
4	Digital output, default low
5	Digital output, default high
6	N/A
7	MicroPython UART1 TX

Default

0

D5 (DIO5/ASSOCIATED_INDICATOR)

Sets or displays the DIO5/ASSOCIATED_INDICATOR configuration (pin 15).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Associated LED
2	N/A
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

D6 (DIO6/RTS)

Sets or displays the DIO6/$\overline{\text{RTS}}$ configuration (pin 16).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Flow control (input)
2	N/A
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

0

D7 (DIO7/CTS)

Sets or displays the DIO7/$\overline{\text{CTS}}$ configuration (pin 12).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Flow control (output)
2	N/A
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

0x1

D8 (DIO8/SLEEP_REQUEST)

Sets or displays the DIO8/$\overline{\text{DTR}}$/SLP_RQ configuration (pin 9).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	SLEEP_REQUEST input

Parameter	Description
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

D9 (DIO9/ON_SLEEP)

Sets or displays the DIO9/ON_SLEEP configuration (pin 13).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	ON/SLEEP output
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

P0 (DIO10/PWM0 Configuration)

Sets or displays the PWM/DIO10 configuration (pin 6).

This command enables the option of translating incoming data to a PWM so that the output can be translated back into analog form.

Parameter range

0 - 6

Parameter	Description
0	Disabled
1	RSSI PWM0 output
2	PWM0 output
3	Digital input

Parameter	Description
4	Digital output, low
5	Digital output, high
6	USB VBUS

Default

0

P1 (DIO11/PWM1 Configuration)

Sets or displays the DIO11 configuration (pin 7).

Parameter range

0, 1, 3 - 7

Parameter	Description
0	Disabled
1	Fan enable. Output is low when the XBee Smart Modem is sleeping, turning an attached fan off when the cellular component is in a power saving mode, and also during Airplane Mode
2	Enables PWM output
3	Digital input
4	Digital output, default low
5	Digital output, default high
6	I^2C SDA
7	USB direct

Default

0

P2 (DIO12 Configuration)

Sets or displays the DIO12 configuration (pin 4).

Parameter range

0, 1, 3 - 5, 7

Parameter	Description
0	Disabled

Parameter	Description
1	SPI_MISO
2	N/A
3	Digital input
4	Digital output, default low
5	Digital output, default high
6	N/A
7	MicroPython UART1 RX

Default

0

P3 (DIO13/DOUT)

Sets or displays the DIO13/DOUT configuration (pin 17).

Parameter range

0, 1

Parameter	Description
0	Disabled
1	UART DOUT enabled

Default

1

P4 (DIO14/DIN)

Sets or displays the DIO14/DIN configuration (pin 3).

Parameter range

0 - 1

Parameter	Description
0	Disabled
1	UART DIN enabled

Default

1

PD (Pull 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.

If the bit is not set in **PR**, the device uses **PD**.

Note Resistors are not applied to disabled lines.

Parameter range

0x0 – 0x7FFF on TH, 0x0 – 0xFFFFF on SMT

Default

0 - 0x7FFF on TH

0 - 0xFFFFF on SMT

PR (Pull-up/down Resistor Enable)

Sets or displays the bit field that configures the internal resistor status for the digital input lines. Internal pull-up/down resistors are not available for digital output pins, analog input pins, or for disabled pins.

Use the **PD** command to specify whether the resistor is pull-up or pull-down.

- 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.

The following table defines the bit-field map for both the **PR** and **PD** commands.

Bit	I/O line	Module pin
0	DIO4	pin 11
1	DIO3/AD3	pin 17
2	DIO2/AD2	pin 18
3	DIO1/AD1	pin 19
4	DIO0/AD0	pin 20
5	DIO6/$\overline{\text{RTS}}$	pin 16
6	DIO8/SLEEP_REQUEST	pin 9
7	DIO14/DIN	pin 3
8	DIO5/ASSOCIATE	pin 15
9	DIO9/On/$\overline{\text{SLEEP}}$	pin 13
10	DIO12	pin 4
11	DIO10	pin 6
12	DIO11	pin 7

Bit	I/O line	Module pin
13	DIO7/$\overline{\text{CTS}}$	pin 12
14	DIO13/DOUT	pin 17

Parameter range

0 - 0x7FFF (bit field)

Default

0x7FFF

M0 (PWM0 Duty Cycle)

Sets the duty cycle of PWM0 (pin 6) for **P0** = **2**, where a value of 0x200 is a 50% duty cycle.

Before setting the line as an output:

1. Enable PWM0 output (P0 (DIO10/PWM0 Configuration) = **2**).
2. Apply the settings (use CN (Exit Command mode) or AC (Apply Changes)).

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 command

Sets the duty cycle of PWM1 for **P1** = **2**, where a value of 0x200 is a 50% duty cycle.

Parameter range

0 - 0x3FF

Default

0

I/O sampling commands

The following AT commands configure I/O sampling parameters.

TP (Temperature)

Displays the temperature of the XBee Smart Modem in degrees Celsius. The temperature value is displayed in 16-bit two’s complement format. For example, **0x1A** = 26 °C, and **0xF6** = -10 °C.

Parameter range

0 - 0xFF which indicates degrees Celsius displayed in 8-bit two's complement format.

Default

N/A

IS (Force Sample)

When run, **IS** reports the values of all of the enabled digital and analog input lines. If no lines are enabled for digital or analog input, the command returns an error.

Command mode

In Command mode, the response value is a multi-line format, individual lines are delimited with carriage returns, and the entire response terminates with two carriage returns. Each line is a series of ASCII characters representing a single number in hexadecimal notation. The interpretation of the lines is:

- Number of samples. For legacy reasons this field always returns 1.
- Digital channel mask. A bit-mask of all I/O capable pins in the system. The bits set to **1** are configured for digital I/O and are included in the digital data value below. Pins D0 - D9 are bits 0 - 9, and P0 - P2 are bits 10 - 12.
- Analog channel mask. The bits set to **1** are configured for analog I/O and have individual readings following the digital data field.
- Digital data. The current digital value of all the pins set in the digital channel mask, only present if at least one bit is set in the digital channel mask.
- Analog data. Additional lines, one for each set pin in the analog channel mask. Each reading is a 10-bit ADC value for a 2.5 V voltage reference.

API operating mode

In API operating mode, **IS** immediately returns an **OK** response.

The API response is ordered identical to the Command mode response with the same fields present. Each field is a binary number of the size listed in the following table. Multi-byte fields are in big-endian byte order.

Field	Size
Number of samples	1 byte
Digital channel mask	2 bytes
Analog chanel mask	1 byte
Samples	2 bytes each

Parameter range

N/A

Default

N/A

Sleep commands

The following AT commands are sleep commands.

SM (Sleep Mode)

Sets or displays the sleep mode of the device.

The sleep mode determines how the device enters and exits a power saving sleep.

In addition to the sleep modes listed below, sleep can be controlled directly by a MicroPython program. See the Power management in MicroPython section of the *Digi MicroPython Programming Guide*. With direct control additional power savings can be realized through intelligent sleep upon immediate completion of application tasks and dynamic sleep intervals can be used to respond to changing needs.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Normal. In this mode the device never sleeps.
1	Pin Sleep. In this mode the device honors the SLEEP_RQ pin. Set D8 (DIO8/SLEEP_REQUEST) to the sleep request function: **1**.
4	Cyclic Sleep. In this mode the device repeatedly sleeps for the value specified by **SP** and spends **ST** time awake.
5	Cyclic Sleep with Pin Wake. In this mode the device acts as in Cyclic Sleep but does not sleep if the SLEEP_RQ pin is inactive, allowing the device to be kept awake or woken by the connected system.

Default

0

SP (Sleep Period)

Sets or displays the time to spend asleep in cyclic sleep modes. In Cyclic sleep mode, the node sleeps with CTS disabled for the sleep time interval, then wakes for the wake time interval.

Parameter range

0x1 - 0x83D600 (x 10 ms)

Default

0x7530 (5 minutes)

ST (Wake Time)

Sets or displays the time to spend awake in cyclic sleep modes.

Parameter range

0x1 - 0x36EE80 (x 1 ms)

Default

0xEA60 (60 seconds)

PA (Requested Active Timer)

The requested Active Timer for PSM.

Note This is related to 3GPP timer T3324.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Parameter range

0 - 0xFFFF (0 - 65535 [* 1 s])

Default

0xa (10 s)

PU (Requested Tracking Area Update Timer)

The requested Active Timer for PSM.

Note This is related to 3GPP timer T3412.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Parameter range

0 - 0x35683200 (* 1 s)

Default

0x8ca00 (576000 s)

Command mode options

The following commands are Command mode option commands.

CC (Command Sequence Character)

The character value the device uses to enter Command mode.

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.

Parameter range

0 - 0xFF

Default

0x2B (the ASCII plus character: **+**)

CT (Command Mode Timeout)

Sets or displays the Command mode timeout parameter. If a device does not receive any valid commands within this time period, it returns to Idle mode from Command mode.

Parameter range

2 - 0x1770 (x 100 ms)

Default

0x64 (10 seconds)

CN (Exit Command mode)

Immediately exits Command Mode and applies pending changes.

Note Whether Command mode is exited using the **CN** command or by **CT** timing out, changes are applied upon exit.

Parameter range

N/A

Default

N/A

GT (Guard Times)

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.

Parameter range

0x2 - 0x6D3 (x 1 ms)

Default

0x3E8 (one second)

MicroPython commands

The following commands relate to using MicroPython on the XBee Smart Modem.

PS (Python Startup)

Sets whether or not the XBee Smart Modem 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 Smart Modem using MicroPython. **PY** is a command with sub-commands. These sub-commands are arguments to **PY**.

Note You can use the **PY** command options to control MicroPython from Digi Remote Manager. Refer to the *Digi MicroPython Programming Guide*.

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)
bundled: 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.
- **bundled**: 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.

PYR (Soft Reset)

PYR performs a MicroPython soft reset which stops any currently executing code. If Python Startup is enabled (PS1) then the stored Python code is run.

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

Firmware version/information commands

The following AT commands are firmware version/information commands.

VR (Firmware Version)

Reads the firmware version on the device.

Parameter range

0 - 0xFFFFFF [read-only]

Default

Set in firmware

VL (Verbose Firmware Version)

Shows detailed version information including the application build date and time.

Parameter range

N/A

Default

Set in firmware

HV (Hardware Version)

Read the device's hardware version. Use this command to distinguish between different hardware platforms. The upper byte returns a value that is unique to each device type. The lower byte indicates the hardware revision.

Parameter range

0 - 0xFFFF [read-only]

Default

Set in firmware

HS (Hardware Series)

Read the device's hardware series number.

Parameter range

N/A

Default

Set in the firmware

%C (Hardware/Software Compatibility)

Specifies what firmware is compatible with this device's hardware. Firmware images with a compatibility value lower than the value returned by **%C** cannot be loaded onto the XBee.

The compatibility number for each firmware image can be found in the corresponding XCTU XML definition file, as the **compatibility_number** field.

Parameter range

Read-only (programmed at manufacturing)

Default

N/A

CK (Configuration CRC)

Displays the cyclic redundancy check (CRC) of the current AT command configuration settings.

Parameter range

0 - 0xFFFFFFFF

Default

N/A

AI (Association Indication)

Reads the Association status code to monitor association progress. The following table provides the status codes and their meanings.

Status code	Meaning
0x00	Connected to the Internet.
0x22	Registering to cellular network.
0x23	Connecting to the Internet.
0x24	The cellular component is missing, corrupt, or otherwise in error. The cellular component requires a new firmware image.
0x25	Cellular network registration denied.
0x2A	Airplane mode.
0x2B	USB Direct active.
0x2C	Cellular component is in PSM.
0x2D	Modem shut down. See SD (Shutdown).
0x2E	Low voltage shut down.
0x2F	Bypass mode active.
0x30	An update is in process.

Status code	Meaning
0x31	Regulatory testing has been enabled. See Regulatory testing commands and %# (Enable/disable test mode).
0xFF	Initializing.

Parameter range

0 - 0xFF [read-only]

Default

N/A

FI (FTP OTA Update Indication)

Reports the result of the previous FTP OTA operation.

Status code	Meaning
0x0	Last update succeeded.
0x1	Update file transfer failed.
0x2	Update image rejected by cellular component.
0x10	A problem processing the update request occurred.
0x11	Update was blocked by XBee sleep.
0x12	One or more update parameters were invalid.
0xFE	An update is currently in progress.
0xFF	No update status to report.

Parameter range

N/A

Default

N/A

FO (FTP OTA command)

The FO command allows for the initiation of a cellular component FTP OTA from an AT command interface.

The FO command has sub-commands that either set or read a parameter, initiate the FTP OTA (ATFOI) or clears the parameters (ATFOC).

The table below shows the FTP OTA parameters that can be set and their default values.

Note Any of the parameter commands in the table below will return ERROR if the entered parameter is invalid or if an FTP OTA has already been initiated.

Command	Parameter	Default
ATFOS	Server	ftp1.digi.com
ATFOP	Port	21
ATFOU	Username	anonymous
ATFOW	Password	fota@digi.com
ATFOD	Directory	support/ublox
ATFOF	Filename	

ATFOI

ATFOI initiates an FTP OTA with the set parameters. To check the status of an initiated FTP OTA , check ATFI to get the status of the last FTP OTA operation.

This can return ERROR immediately if there are invalid parameters set or another FTP OTA already in progress.

ATFOC

ATFOC clears all parameters back to their defaults as listed in the table above.

Example usage

Setting a parameter

```
ATFOSmyftp.server.com
OK
```

Reading a parameter

```
ATFOS
myftp.server.com
```

Initiating FTO OTA

```
ATFOI
OK
```

Diagnostic interface commands

The following AT commands are diagnostic interface commands.

DI (Remote Manager Indicator)

Displays the current Remote Manager status for the XBee.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

Range

Value	Description
0x00	Connected, but without TLS or authentication.
0x01	Before connection to the Internet.
0x02	Remote Manager connection in progress.
0x03	Disconnecting from Remote Manager.
0x04	Not configured for Remote Manager.
0x05	Connected over TLS.
0x06	Connected over TLS with authenticated server.

Default

N/A

CI (Protocol/Connection Indication)

Displays information regarding the last IP connection when using Transparent mode (**AP** = **0**).

The following table provides the parameter's meaning when **IP** = **0** for UDP connections.

Parameter	Description
0x00	The socket is open.
0x01	Tried to send but could not.
0x02	Invalid parameters (bad IP/host).
0x10	Not registered to the cell network.
0x11	Cellular component not identified yet.
0x12	DNS query lookup failure.
0x13	Socket leak
0x20	Bad handle.
0x21	User closed.
0x22	Unknown server - DNS lookup failed.
0x23	Connection lost.
0x24	Unknown.
0xFF	No known status.

The following table provides the parameter's meaning when **IP** = **1** or **4** for TCP connections.

Parameter	Description
0x00	The socket is open.
0x01	Tried to send but could not.
0x02	Invalid parameters (bad IP/host).
0x03	TCP not supported on this cellular component.
0x10	Not registered to the cell network.
0x11	Cellular component not identified yet.
0x12	DNS query lookup failure.
0x13	Socket leak
0x20	Bad handle.
0x21	User closed.
0x22	No network registration.
0x23	No internet connection.
0x24	No server - timed out on connection.
0x25	Unknown server - DNS lookup failed.
0x26	Connection refused.
0x27	Connection lost.
0x28	Unknown.
0x2F	TLS Socket Authentication Error
0xFF	No known status.

The following table provides the parameter's meaning when **IP** = **2** for SMS connections.

Parameter	Description
0x00	SMS successfully sent.
0x01	SMS failed to send.
0x02	Invalid SMS parameters - check P# (Destination Phone Number).
0x03	SMS not supported.
0x10	No network registration.
0x11	Cellular component stack error.
0x12	A modem update is in-progress. Try again after its completion.
0xFF	No SMS state to report (no SMS messages have been sent).

Parameter range

0 - 0xFF (read-only)

Default

-

AS (Active scan for network environment data)

Scans for mobile cells in the vicinity and returns information about the cells in the service area of the device. When you run the command, the cell module waits until all other communication is idle and then performs the scan.

The information that can be reported by this command varies based on the network technology of the module that you are using.

In both AT and API mode the command returns line-based records mapping key-value pairs. The record for the serving cell begins with the capital letter S, and keys for the fields are MCC, MNC, Area, CID, and Signal. Each line describes a particular cell and only those values determined during a single scan are reported.

Example

```
atas

S MCC:311 MNC:480 Area:48707
CID:48825632 Signal:-88
CID:48825612 Signal:-95
CID:48825603 Signal:-68
CID:48825601 Signal:-71
```

Parameter range

0-1

Value	Description
0 or no value	Scans for mobile cells in the vicinity and returns information about the cells in the service area of the module. When you run the command, the cell module waits until all other communication is idle and then performs the scan.

Parameter range

N/A

Default

N/A

Execution commands

The location where most AT commands set or query register values, execution commands execute an action on the device. Execution commands are executed immediately and do not require changes to be applied.

NR (Network Reset)

NR resets the network layer parameters. The XBee Smart Modem tears down any TCP/UDP sockets and resets Internet connectivity.

The XBee Smart Modem responds immediately with an **OK** on the UART and then causes a network restart.

Parameter range

0

Default

N/A

!R (Modem Reset)

Forces the cellular component to reboot.

CAUTION! This command is for advanced users, and you should only use it if the cellular component becomes completely stuck while in Bypass mode. Normal users should never need to run this command. See the FR (Force Reset) command instead.

Range

N/A

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 is a command with sub-commands. These sub-commands are arguments to **FS**.

For **FS** commands, you have to type **AT** before the command, for example **ATFS PWD**, **ATFS LS** and so forth.

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 capital **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 capital **E**.

ATFS (File System)

When sent without any parameters, **FS** prints a list of supported commands.

ATFS PWD

Prints the current working directory, which always starts with **/** and defaults to **/flash** at startup.

ATFS CD *directory*

Changes the current working directory to **directory**. Prints the current working directory or an error if unable to change to **directory**.

ATFS MD *directory*

Creates the directory **directory**. Prints **OK** if successful or an error if unable to create the requested directory.

ATFS 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 filesize 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. Secure files end with a hash mark (**#**) and you cannot download them.

```
<DIR> ./
<DIR> ../
<DIR> cert/
<DIR> lib/
   32 test.txt
 1234 secure.bin#
```

ATFS PUT *filename*

Starts a YMODEM receive on the XBee Smart Modem, storing the received file to filename and ignoring the filename that appears in block 0 of the YMODEM transfer. The XBee Smart Modem 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.

ATFS XPUT filename

Similar to the **PUT** command, but stores the file securely on the XBee Smart Modem. See Secure files for details on what this means.

If the command is incorrect, the reply will be an error as described in Error responses.

ATFS 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**.

ATFS GET *filename*

Starts a YMODEM send of **filename** on the XBee device. When it is ready to send, the XBee Smart Modem 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.

ATFS MV *source_path dest_path*

Moves or renames the selected file or directory **source_path** to the new name or location **dest_path**. This command fails with an error if **source_path** does not exist, or **dest_path** already exists.

Note Unlike a computer's command prompt which moves a file into the **dest_path** if it is an existing directory, you must specify the full name for **dest_path**.

ATFS 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.

ATFS 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
```

ATFS FORMAT confirm

Reformats the file system, leaving it with a default directory structure. Pass the word **confirm** as the first parameter to confirm the format. The XBee Smart Modem responds with **Formatting...**, adds a period every second until the format is complete and ends the response with a carriage return.

BLE commands

The following AT commands are BLE commands.

BI (Bluetooth Identifier)

A human-friendly name for the device. This name appears in BLE advertisement messages.

If set to the default (a single ASCII space character), the Bluetooth identifier displays as the device name, such as XBee 3 Cellular LTE-M/NB-IoT.

If you are 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)

BL (Bluetooth MAC address)

The BL command reports the EUI-48 Bluetooth device address (BLE MAC 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

BP (Bluetooth Advertisement Power Level)

Sets or displays the output power level that will be used for Bluetooth advertisements.

Parameter range

0x0 - 0x3

Bit	Description
0	-20 dBm
1	-10 dBm
2	0 dBm
3	8 dBm

Default

3 (8 dBm)

BT (Bluetooth enable)

Enables or disables the Bluetooth functionality.

Parameter range

Bit	Description
0	Bluetooth functionality is disabled.
1	Bluetooth functionality is enabled.

Default

0

$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 SRP (Secure Remote Password) Salt is a 32-bit number used to create an encrypted password for the XBee device. The **$S** command is used in conjunction with the **$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 device.

The salt is configured 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 XBee device does not allow for 0 to be valid salt. If the value is 0, SRP is disabled and you will not be able to authenticate using Bluetooth.

Parameter range

0 - FFFFFFFF

Default

0

$V, $W, $X, $Y (SRP password verifier)

Note You should only use these commands if you have already configured a password on the XBee device and the salt verifier values correspond to the password.

The **$V**, **$W**, **$X**, and **$Y** commands are used in conjunction with the **$S** command used to create an encrypted password for the XBee device. Together with the **$S** command, these commands authenticate the client for the BLE API Service without storing the XBee password on the XBee device.

The salt is configured 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 valule.

Parameter range

1 - 32 bytes (1-64 hexadecimal characters in command mode)

Default

0

Remote Manager commands

The following commands are used with Remote Manager.

MO (Remote Manager Options)

Configures the connection to Remote Manager.

Note When bit 0 is set to 0, you should manage the Remote Manager keepalive interval, which may otherwise result in excessive data usage. See Configure Remote Manager keepalive interval.

Parameter range

0, 1, 7

Bit	Description
0	Maintains a persistent TCP connection to Remote Manager. If the XBee Smart Modem cannot establish a connection with Remote Manager, it waits 30 seconds before trying again. On each successive connection failure, the wait time doubles (60 seconds, 120, 240, and so on) up to a maximum of 1 hour. This time resets to 30 seconds once the connection to Remote Manager succeeds or if the device is reset.
1	TCP connection uses TLS. This is the default.
2-6	Reserved for future use.
7	Sets up a constant TLS connection with SM/UDP still enabled.

Default

6 (Bits 1 and 2 are enabled by default.)

DF (Remote Manager Status Check Interval)

Defines the number of minutes between polls for Remote Manager activity.

Parameter range

1 to 0x10E0

Default

1440

EQ (Remote Manager FQDN)

Sets or display the fully qualified domain name of the Remote Manager server.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Range

From 0 through 63 ASCII characters.

Default

my.devicecloud.com

K1 (Remote Manager Server Send Keepalive)

Specify the Remote Manager Server Send Transmit Keepalive Interval value in seconds. The XBee device considers a Remote Manager connection to have failed after 3 missed keepalives.

This command works with the K2 command to limit data usage. See Configure Remote Manager keepalive interval.

Note Changing this value causes any currently active Remote Manager connections to be closed and recreated.

Parameter range

10 - 7200 (x 1 s)

Default

0x258 (600 seconds)

K2 (Remote Manager Device Send Keepalive)

Specify the Remote Manager Device Send Transmit Keepalive Interval value in seconds. The Remote Manager considers a connection to have failed after 3 missed keepalives.

This command works with the K1 command to limit data usage. See Configure Remote Manager keepalive interval.

Note Changing this value causes any currently active Remote Manager connections to be closed and recreated.

Parameter range

10 - 7200 (x 1 s)

Default

0x258 (600 seconds)

$D (Remote Manager certificate)

Defines the TLS Remote Manager certificate.

Parameter range

N/A

Default

/flash/cert/digi-remote-mgr.pem

RI (Remote Manager Service ID)

Sets the Remote Manager service ID for the XBee.

See Configure SMS messaging in Remote Manager for more information.

Range

-

Default

idgp

DP (Remote Manager Phone Number)

Sets the Remote Manager phone number for the XBee device. This code must match the phone number option in the **SMS Configuration** dialog.

See Configure SMS messaging in Remote Manager for more information.

Range

-

Default

32075

HF (Health Metrics Reporting Frequency)

Reports the time between attempts to upload metrics. The time is measured in minutes. Metrics which cannot be collected or reported at any particular time are skipped until the next attempt.

Parameter range

1 to 0xFFFF

Value	Description
0x3c	One hour.

Default

0x3c

HM (Health Metrics)

Sets the Health Metrics to report. This is a bit-mask of values. Each bit set in the mask represents a metric which is reported to Remote Manager.

Parameter range

N/A

Bit	Description
0	Signal Strength. Set bit 0 to enable reporting of signal strength metrics. If available on your device the following metrics are reported: ■ "*metrics/signal_strength*": Uncached RSSI signal value. This is the same value as reported by the DB command with parameter 1, in dbM. ■ "*metrics/signal_receive_power*": Reference Signals Received Power (RSRP). This is the same value as reported by the SW command, in dBm. ■ "*metrics/signal_receive_quality*": Reference Signal Received Quality (RSRQ). This is the same value as reported by the SQ command, in dB.
1	Module Temperature in Celsius. This is reported to the "*metrics/temperature*" Data Stream in Remote Manager for the devices.

Bit	Description
2	TCP data estimated transfer counters. Set bit 2 to enable reporting of data counters for TCP traffic. The data reported is application data and does not include the overhead of the protocol. Data not counted includes headers and retransmissions which may be used by providers to calculate billed amounts. The metrics are set to 0 after reset or after the metrics are reported to Remote Manager. The reported metrics are as follows: ■ "*metrics/tcp/sent*": TCP data sent from the device. ■ "*metrics/tcp/received*": TCP data received.
3	UDP data estimated transfer counters. Set bit 3 to enable reporting of data counters for UDP traffic. The data reported is application data and does not include the overhead of the UDP protocol including headers. The metrics are set to 0 after reset or after the metrics are reported to Remote Manager. The reported metrics are as follows: ■ "*metrics/udp/sent*": UDP data sent from the device. ■ "*metrics/udp/received*": UDP data received.
4	Link Deactivations. Set bit 4 to enable reporting the number of internet link deactivations since the last reset or reporting. The metrics are set to 0 after reset or after the metrics are reported to Remote Manager. This is reported to the "*metrics/link_deactivations*" Data Stream in Remote Manager for the devices.
5	Sleep metrics counter. Set bit 5 to enable reporting of the number of times the device has slept since the last report. This works in conjunction with the HF command. The value is reset to 0 after reset or after the metrics are reported to Remote Manager.
7	Enable reporting of information about the current serving cell. The reported metrics include: ■ **Mobile Country Code**: metrics/cellular/1/sim1/mcc ■ **Mobile Network Code**: metrics/cellular/1/sim1/mnc ■ **Location Area Code**: metrics/cellular/1/sim1/lac ■ **Cell ID**: metrics/cellular/1/sim1/cid

Default

0x0

ER (Remote Manager TCP Port Override)

Use this command to specify a TCP port other than the default Remote Manager TCP port. The defaults are 0xC7D when unencrypted and 0xC7F when TLS is enabled.

- Value is 0: The default Remote Manager TCP port is used.
- Value is non-zero: Specify the TCP port that should be used. The default Remote Manager TCP port is overridden.

Parameter range

0x0 - 0xFFFF

Default

0x0

ES (Remote Manager UDP Port Override)

Use this command to specify a UDP port other than the default Remote Manager UDP port.

- Value is 0: The default Remote Manager UDP port is used.
- Value is non-zero: Specify the UDP port that should be used. The default Remote Manager UDP port is overridden. The default UDP port is 0xCE1.

Parameter range

0x0 - 0xFFFF

Default

0x0

MT (Remote Manager Idle Timeout)

Specify the length of time (in minutes) that a TCP connection to Remote Manager can be idle. When the time limit is met the TCP connection is closed.

For example, you can use this command to adjust the desired timeout when a TCP connection is used without a persistent connection to Remote Manager. This command can be used in conjunction with devices that use SM/UDP or SM/SMS and scheduled tasks within Remote Manaer after a request connect task is performed to connect on demand. For more information on situations where this command applies, see Configure Remote Manager features using automations.

This command works in conjunction with the MO command. If MO bit 0 is set (to maintain a persistent TCP connection to Remote Manager), the configuration for the MT command is ignored.

Parameter range

0x1 - 0x5A0

Default

0xA

System commands

The following commands are used to assign descriptors to the XBee Smart Modem, which distinguish the devices from each other in Remote Manager.

KL (Device Location)

Sets or displays a user-defined physical location for the XBee displayed in Remote Manager.

Range

Up to 20 ASCII characters

Default

One ASCII space character (0x20).

KP (Device Description)

Sets or displays a user-defined description for the XBee displayed in Remote Manager.

Range

Up to 20 ASCII characters

Default

One ASCII space character (0x20)

KC (Contact Information)

Sets or displays user-defined contact information for the XBee displayed in Remote Manager.

Range

Up to 20 ASCII characters

Default

One ASCII space character (0x20).

Socket commands

The following AT commands are socket commands.

SI (Socket Info)

Lists either information about a given socket or lists the socket IDs of all active (open) sockets on the modem in a human-readable format.

When the **SI** command is issued without a parameter, the XBee outputs a list of socket IDS in hex, separated by carriage returns (<CR>). After the last socket ID has been printed the list is terminated with an additional carriage return.

In both API and command mode the payload (output) will have the following format:

```
ID<CR>
ID<CR>
. . .
ID<CR>
<CR>
```

In the list of socket IDs, an asterisk (*) displays after the socket ID for non-Extended API Sockets (which are sockets created implicitly when using IPv4 TX API frames). In the example below, the 0x00 socket is an IPv4 TX/RX socket, and the 0x01 and 0x02 sockets are both Extended API sockets. The socket IDs are displayed in ascending order, from smallest socket value to the largest.

```
0x00*
0x01
0x02
```

Note When sending AT commands for API frames it is standard to send the command as ASCII text and the parameters for that command as binary.

When the **SI** command is issued with a socket ID, specified in hex, the response is a list of information about the socket. The list is separated by carriage returns (<CR>) and terminated with an additional carriage return.

In both API and command mode the payload/output will have the following format:

```
ID<CR>
STATE<CR>
PROTOCOL<CR>
LOCAL_PORT<CR>
REMOTE_PORT<CR>
REMOTE_ADDRESS<CR>
<CR>
```

Field	Description
ID	The socket ID.
STATE	The state of the socket: ■ ALLOCATED ■ CONNECTING ■ CONNECTED ■ LISTENING ■ BOUND ■ CLOSING
PROTOCOL	The protocol of the socket: ■ UDP ■ TCP ■ TLS
LOCAL_PORT	The local port of the socket. This is 0 unless the socket is explicitly bound to a port.
REMOTE_PORT	The remote port of the socket.
REMOTE_ADDRESS	The remote IPv4 address for the given socket. This is 0.0.0.0 for an unconnected socket.

Parameter range

0x00 - 0xFE

Default

-

Power measurement commands

The following commands enable you to access voltage readings and manage a value for minimum allowed operating voltage.

%V command

Measures the supply voltage of the XBee VCC pin for the device in mV units.

Parameter range

N/A

Default

N/A

%L (Low voltage shutdown base threshold)

Sets the voltage threshold in millivolts at which the XBee enters a shutdown state. You must enable this feature by setting the DO command bit 4. See Low voltage shutdown.

Parameter range

0xA28 - 0xC80 mV

Default

0xBB8 mV

%M (Low voltage shutdown reset offset)

The voltage offset in millivolts above %L command (Low voltage shutdown base threshold) at which the XBee recovers from a shutdown state by resetting. You must enable this feature by setting the DO command bit 4. See Low voltage shutdown.

Parameter range

0x64 - 0x2BC mV

Default

0xC8 mV

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 firmware, so build 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).
AP = 3	N/A
AP = 4	MicroPython REPL
AP = 5	Bypass mode. This mode is for direct communication with the underlying chip and is only for advanced users.

The API data frame structure differs depending on what mode you choose.

API frame format

An API frame consists of the following:

- Start delimeter
- 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 reason for 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	Checksum
			Data	
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	Checksum
			Data	
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:

	Length		**Frame data**								
				Data							
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB		Data							

- **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 Smart Modem 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

API frames

The following sections describe the API frames.

AT Command - 0x08

Description

Use this frame to query or set parameters on the local device. Changes this frame makes to device parameters take effect after executing the AT command.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x08	Byte	
Frame ID		Byte	Identifies the data frame for the host to correlate with a subsequent ACK. If set to **0**, the device does not send a response.
AT command		Byte	Command name: two ASCII characters that identify the AT command.
Parameter value		Byte	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the register.

AT Command: Queue Parameter Value - 0x09

Description

This frame allows you to query or set device parameters. In contrast to AT Command - 0x08, this frame queues new parameter values and does not apply them until you issue either:

- The AT Command (0x08) frame
- The **AC** command

When querying parameter values, the 0x09 frame behaves identically to the 0x08 frame. The device returns register queries immediately and not does not queue them. The response for this command is also an AT Command Response frame (0x88).

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x09	Byte	
Frame ID		Byte	Identifies the data frame for the host to correlate with a subsequent ACK. If set to **0**, the device does not send a response.
AT command		Byte	Command name: two ASCII characters that identify the AT command.
Parameter value		Byte	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the register.

Transmit (TX) SMS - 0x1F

Description

Transmit an SMS message. The frame allows international numbers with or without the **+** prefix. If you omit **+** and are dialing internationally, you need to include the proper International Dialing Prefix for your calling region, for example, 011 for the United States.

Note For NB-IoT, SMS support is dependent on the network. Contact your network provider for details.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x1F	Byte	
Frame ID		Byte	Reference identifier used to match status responses. **0** disables the TX Status frame.
Options		Byte	Reserved for future use.
Phone number		20 byte string	String representation of phone number terminated with a null (0x0) byte. Use numbers and the **+** symbol only, no other symbols or letters.
Payload		Variable (160 characters maximum)	Data to send as the body of the SMS message.

Transmit (TX) Request: IPv4 - 0x20

Description

A TX Request message causes the device to transmit data in IPv4 format. A TX request frame for a new destination creates a network socket. After the network socket is established, data from the network that is received on the socket is sent out the device's serial port in the form of a Receive (RX) Packet frame.

When you specify protocol **4** (TLS), the profile configuration specified by $0 (TLS Profile 0) is used to form the TLS connection.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x20	Byte	
Frame ID		Byte	Reference identifier used to match status responses. **0** disables the TX Status frame.
Destination address		32-bit big endian	
Destination port		16-bit big endian	
Source port		16-bit big endian	If the source port is **0**, the device attempts to send the frame data using an existing open socket with a destination that matches the destination address and destination port fields of this frame. If there is no matching socket, then the device attempts to open a new socket. If the source port is non-zero, the device attempts to send the frame data using an existing open socket with a source and destination that matches the source port, destination address, and destination port fields of this frame. If there is no matching socket, the TX Status frame returns an error.

Field name	Field value	Data type	Description
Protocol		Byte	0 = UDP 1 = TCP 4 = SSL/TLS **Note** For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.
Transmit options		Byte bitfield	Bit fields are offset 0 Bit field 0 - 7. Bits 0, and 2-7 are reserved, bit 1 is not. BIT 1 = **1** - Terminate the TCP socket after transmission is complete **0** - Leave the socket open. Closed by timeout, see TM (IP Client Connection Timeout). Ignore this bit for UDP packets. All other bits are reserved and should be **0**.
Payload		Variable	Data to be transferred to the destination, may be up to 1500 bytes. UDP is limited to 512 bytes.

Tx Request with TLS Profile - 0x23

Description

The frame gives greater control to the application over the TLS settings used for a connection.

A TX Request with TLS Profile frame implies the use of TLS and behaves similar to the TX Request (0x20) frame, with the protocol field replaced with a TLS Profile field to choose from the profiles configured with the $0, $1, and $2 configuration commands.

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x23	Byte	
Frame ID		Byte	Reference identifier used to match status responses. **0** disables the TX Status frame.

Field name	Field value	Data type	Description
Destination address		32-bit big endian	
Destination port		16-bit big endian	
Source port		16-bit big endian	If the source port is **0**, the device attempts to send the frame data using an existing open socket with a destination that matches the destination address and destination port fields of this frame. If there is no matching socket, then the device attempts to open a new socket. If the source port is non-zero, the device attempts to send the frame data using an existing open socket with a source and destination that matches the source port, destination address, and destination port fields of this frame. If there is no matching socket, the TX Status frame returns an error.
TLS profile		Byte	Zero-indexed number that indicates the profile as specified by the corresponding **$<*num*>** command.
Transmit options		Byte bitfield	Bit fields are offset 0 Bit field 0 - 7. Bits 0, and 2-7 are reserved, bit 1 is not. BIT 1 = **1** - Terminate the TCP socket after transmission is complete **0** - Leave the socket open. Closed by timeout, see TM (IP Client Connection Timeout). Ignore this bit for UDP packets. All other bits are reserved and should be **0**.
Payload		Variable	Data to be transferred to the destination, may be up to 1500 bytes.

AT Command Response - 0x88

Description

A device sends this frame in response to an AT Command (0x08) frame. Some commands send back multiple frames.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x88	Byte	
Frame ID		Byte	Identifies the data frame for the host to correlate with a subsequent ACK. If set to **0**, the device does not send a response.
AT command		Byte	Command name: two ASCII characters that identify the AT command.
Status	##	Byte	0 = OK 1 = ERROR 2 = Invalid command 3 = Invalid parameter
Parameter value		Byte	Register data in binary format. If the register was set, then this field is not returned.

Transmit (TX) Status - 0x89

Description

Indicates the success or failure of a transmit operation.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x89	Byte	
Frame ID		Byte	Refers to the frame ID specified in a previous transmit frame
Status		Byte	Status code (see the table below)

The following table shows the status codes.

Code	Description
0x0	Successful transmit
0x20	Connection not found
0x21	Failure to transmit to cell network
0x22	Not registered to cell network

Code	Description
0x2c	Invalid frame values (check the phone number)
0x31	Internal error
0x32	Resource error (retry operation later). See Socket limits in API mode for more information.
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 - 0x2D.
0x7D	Interface not accepting frames. See User Data Relay - 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
0x8B	TLS Socket Authentication Error

Modem Status - 0x8A

Description

Cellular component status messages are sent from the device in response to specific conditions.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x8A	Byte	
Status	##	Byte	0 = Hardware reset or power up 1 = Watchdog timer reset 2 = Registered with cellular network 3 = Unregistered with cellular network 0x0E = Remote Manager connected 0x0F = Remote Manager disconnected 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

Note The BLE Connect and BLE Disconnect events are reported over the UART/SPI interface in API mode when a valid Bluetooth connection has been made and API mode has been unlocked, and also when an unlocked connection disconnects.

Receive (RX) Packet: SMS - 0x9F

Description

This XBee Smart Modem uses this frame when it receives an SMS message.

Note For NB-IoT, SMS support is dependent on the network. Contact your network provider for details.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame Type	0x9F	Byte	
Phone number		20 byte string	String representation of the phone number, padded out with null bytes (0x0).
Payload		Variable	Body of the received SMS message.

Receive (RX) Packet: IPv4 - 0xB0

Description

The XBee Smart Modem uses this frame when it receives RF data on a network socket that is created by a TX request frame or configuring C0 (Source Port).

Note For NB-IoT, TCP support is dependent on the network. Contact your network provider for details.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Frame data fields	Offset	Description
Frame type	3	0xB0
IPv4 32-bit source address	MSB 4	The address in the example below is for a source address of **192.168.0.104**. 32-bit big endian.
	5	
	6	
	7	
16-bit destination port	MSB 8	The port that the packet was received on. 16-bit big endian.
	LSB 9	
16-bit source port	MSB 10	The port that the packet was sent from. 16-bit big endian.
	LSB 11	
Protocol	MSB 12	0 = UDP 1 = TCP 4 = SSL over TCP
Status	13	Reserved
Payload	14	Data received from the source. The maximum size is 1500 bytes.
	15	
	16	
	17	
	18	

User Data Relay - 0x2D

Description

Allows for data to be sent to an interface with a designation of a target interface for the data to be output on. The frame can be sent or received from any of the following interfaces: MicroPython (internal interface), UART, and BLE. This frame is used in conjunction with User Data Relay Output - 0xAD.

You can send and receive User Data Relay Frames from MicroPython. See Send and receive User Data Relay frames in the *MicroPython Programming Guide*.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x2D	Byte	
Frame ID			Reference identifier used to match TX Status frames (type 0x89) sent for errors. A value of **0** disables the TX Status frame.
Destination interface		Byte	0 = Serial port (SPI, or UART when in API mode) 1 = BLE 2 = MicroPython
Data		Variable	

Error cases

The Frame ID is used to report error conditions in a method consistent with existing transmit frames. The error codes are mapped to statuses. The following conditions result in an error that is reported in a TX Status frame, referencing the frame ID from the 0x2d request.

- **Invalid interface** (0x7c) : The user specified a destination interface that does not exist.
- **Interface not accepting frames** (0x7d): 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.

Example use cases

These examples show you can use this frame.

- You can use the 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.

User Data Relay Output - 0xAD

Description

Allows for data to be received on an interface with a designation of the target interface for the data to be output on. The frame can be sent or received from any of the following interfaces: MicroPython (internal interface), UART, and BLE. This frame is used in conjunction with User Data Relay - 0x2D.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xAD	Byte	
Source interface		Byte	0 = Serial port (SPI, or UART when in API mode) 1 = BLE 2 = MicroPython
Data		Variable	

BLE Unlock API - 0x2C

Description

The XBee Smart Modem uses this frame to authenticate a connection on the Bluetooth interface and unlock the processing of AT command frames. This frame is used in conjunction with the Response (0xAC) frame.

The unlock process is an implementation of the SRP (Secure Remote Password) algorithm using the RFC5054 1024-bit group and the SHA-256 hash algorithm . The SRP identifying user name, commonly referred to as *I*, is fixed to the value `apiservice`.

Upon completion, each side will have derived a shared session key which is used to communicate in an encrypted fashion with the peer. Additionally, a Modem Status - 0x8A with the status code 0x32 (Bluetooth Connected) is sent through the UART (if AP=1 or 2). When an unlocked connection is terminated, a Modem Status Frame with the status code 0x33 (Bluetooth Disconnected) is sent through the UART.

The following implementations are known to work with the BLE SRP implementation:

- https://github.com/cncfanatics/SRP

You will need to modify the hashing algorithm to SHA256 and the values of N and g to use the RFC5054 1024-bit group.

- https://github.com/cocagne/csrp
- https://github.com/cocagne/pysrp

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Frame data fields	Offset	Description
Frame type	3	0x2C = Request 0xAC = Response
Step	4	Indicates the phase of authentication and interpretation of payload data. See 1 = Client presents *A* value 2 = Server presents *B* and *salt* 3 = Client present *M1* session key validation value 4 = Server presents *M2* session key validation value and two 12-byte nonces See the phase tables below for more information. Step values greater than 0x80 indicate error conditions. 0x80 = Unable to offer B (cryptographic error with content, usually due to A mod N == 0) 0x81 = Incorrect payload length 0x82 = Bad proof of key 0x83 = Resource allocation error 0x84 = Request contained a step not in the correct sequence
Payload	5	Payload structure varies by Frame ID value. Descriptions are in the tables, below.

The tables below give more information about the phase of authentication and interpretation of payload data.

Phase 1 (Client presents A)

If the *A* value is zero, the server will terminate the connection.

Frame data field	Offset in frame	Length
A	5	128 bytes

Phase 2 (Server presents B and salt)

Frame data field	Offset in frame	Length
salt	5	4 bytes
B	9	128 bytes

Phase 3 (Client presents M1)

Frame data field	Offset in frame	Length
M1	5	Hash algorithm digest length (32 bytes for SHA256)

Phase 4 (Server presents M2)

Frame data field	Offset in frame	Length
M2	5	Hash algorithm digest length (32 bytes for SHA256)
TX nonce	37	12-byte (96-bit) random nonce, used as the constant prefix of the counter block for encryption/decryption of data transmitted to the API service by the client
RX nonce	49	12-byte (96-bit) random nonce, used as the constant prefix of the counter block for encryption/decryption of data received by the client from the API service

Upon completion of *M2* verification, the session key has been determined to be correct and the API service is unlocked and will allow additional API frames to be used. Content from this point will be encrypted using AES-256-CTR with the following parameters:

- **Key**: The entire 32-byte session key.
- **Counter**: 128 bits total, prefixed with the appropriate nonce shared during authentication. Initial remaining counter value is 1.

  The counter for data sent into the XBee API Service is prefixed with the *TX nonce* value (see the **Phase 4** table, above), and the counter for data sent by the XBee to the client is prefixed with the *RX nonce* value.

Example sequence to perform AT Command XBee API frames over BLE

1. Discover the XBee 3 device through scanning for advertisements.
2. Create a connection to the GATT Server.
3. Optional, but recommended, request a larger MTU for the GATT connection.
4. Turn on indications for the API Response characteristic.
5. Perform unlock procedure using unlock frames. See BLE Unlock API - 0x2C.
6. Once unlocked, AT Command (0x8) frames may be sent and AT Command Response frames received.
    a. For each frame to send, form the API Frame, and encrypt through the stream cipher as described in the unlock procedure. See BLE Unlock API - 0x2C.
    b. Write the frame using one or more Write operations.
    c. When successful, the response arrives in one or more indications. If your stack does not do it for you, remember to acknowledge each indication as it is received. Note that you are expected to process these indications and the response data is not available if you attempt to perform a read operation to the characteristic.
    d. Decrypt the stream of content provided through the indications, using the stream cipher as described in the unlock procedure. See BLE Unlock API - 0x2C.

BLE Unlock Response - 0xAC

Description

The XBee Smart Modem uses the **BLE Unlock API - 0x2C** frame to authenticate a connection on the Bluetooth interface and unlock the processing of AT command frames. This frame is used in conjunction with the **Response (0xAC)** frame.

For details, see BLE Unlock API - 0x2C.

Socket Create - 0x40

Description

Use this frame to create a new socket with the following protocols: TCP, UDP, or TLS.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x40	Byte	
Frame ID		Byte	Reference identifier used to match status responses. A response is required and will be sent regardless of the frame ID.
Protocol		Byte	0 = UDP 1 = TCP 4 = SSL over TCP

Socket Create Response - 0xC0

Description

The device sends this frame in response to a Socket Create (0x40) frame. It contains a socket ID that should be used for future transactions with the socket and a status field.

If the status field is non-zero, which indicates an error, the socket ID will be set to 0xFF and the socket will not be opened.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xC0	Byte	
Frame ID		Byte	A reference identifier used to match status responses.
Socket ID		Byte	A unique socket ID to address the socket. This field is 0xFF if the value in the status field is non-zero.
Status		Byte	Status code. See table below.

The following table shows the status codes.

Code	Description
0x0	Successful open
0x22	Not registered to cell network
0x31	Internal error

Code	Description
0x32	Resource error: retry the operation later See Socket limits in API mode.
0x7B	Invalid protocol
0x7E	A modem update is in process. Try again after its completion.
0x85	Unknown error
0x86	Invalid TLS configuration

Socket Option Request - 0x41

Description

Use this frame to modify the behavior of sockets to change their behavior to be different than the normal default behavior. If the Option Data field is zero-length the request acts as a query, and the Socket Option Response frame (0xC1) reports the current effective value.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x41	Byte	
Frame ID		Byte	A reference identifier used to match status responses. Requests made with Frame ID 0 will not send a response.
Socket ID		Byte	The socket ID to modify.
Option ID		Byte	Identifier of the parameter to change.
Option Data		Variable	Variable length field based on option type. If zero length, the current effective value will be returned in the response frame.

Options

Option ID	Option Name	Data Type	Default Value	Description
0x00	TLS Profile	Byte	0x00	Determines the TLS profile to be used: $0 - $2. This is valid only for TLS sockets.

Socket Option Response - 0xC1

Description

Reports the status of requests made with the Socket Option Request (0x41) frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xC1	Byte	
Frame ID		Byte	Identifier provided in request.
Socket ID		Byte	The socket ID for which modification was requested.
Option ID		Byte	Identifier of the parameter requested.
Status		Byte	0x00: Success 0x01: Invalid parameters 0x02: Failed to retrieve option value 0x20: Bad socket ID
Option Data		Variable	Current effective value of the option. This field is only present if the corresponding request was a query (empty value).

Socket Connect - 0x42

Description

Use this frame to connect a socket to the given address and port.

For a UDP socket, this filters out any received responses that are not from the specified remote address and port.

Two frames occur in response:

1. Socket Connect Response frame: Arrives immediately and confirms the request.
2. Socket Status frame: Indicates if the connection was successful.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x42	Byte	
Frame ID		Byte	A reference identifier used to match status responses. If set to **0**, the device does not send a response.
Socket ID		Byte	ID of the socket to connect.
Destination port		16-bit big endian	
Destination address type		Byte	0: Indicates the destination address field is a **binary** IPv4 address in network byte order. 1: Indicates the destination address field is a string containing either a dotted quad value or a domain name to be resolved.
Destination address		Variable	

Socket Connect Response - 0xC2

Description

The device sends this frame in response to a Socket Connect (0x42) frame. The frame contains a status regarding the initiation of the connect.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xC2	Byte	
Frame ID		Byte	A reference identifier used to match status responses.
Socket ID		Byte	ID of the socket that will be connected.
Status		Byte	Status code. See the table below.

The following table shows the status codes.

Code	Description
0x00	Successfully started the connection process
0x01	Invalid destination address type
0x02	Invalid parameter: address or port
0x03	Connection already in progress
0x04	Already connected
0x05	Unknown error
0x20	Invalid socket ID

Socket Close - 0x43

Description

Use this frame to close an Extended API socket with a specified Socket ID or to close all currently open Extended API sockets.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x43	Byte	
Frame ID		Byte	A reference identifier used to match status responses. If set to **0**, the device does not send a response.
Socket ID		Byte	The following options can be used: ■ ID of the socket to be closed. ■ 0xFF: Close all Extended API sockets that are currently open.

Socket Close Response - 0xC3

Description

The device sends this frame in response to a Socket Close (0x43) frame. Since a close will always succeed for a socket that exists, the status can be only one of two values: Success or Bad socket ID.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xC3	Byte	
Frame ID		Byte	A reference identifier used to match status responses.
Socket ID		Byte	ID of the socket that has been closed.
Status		Byte	0x00 = Success 0x20 = Bad socket ID

Socket Send (Transmit) - 0x44

Description

A Socket Send message causes the device to transmit data using the current connection. For a non-zero frame ID, this will elicit a Transmit (TX) Status - 0x89 frame.

This frame requires a successful Socket Connect - 0x42 frame first. For a socket that is not connected, the device responds with a Transmit (TX) Status - 0x89 frame with an error. To send data from a UDP socket that is not connect, use a Socket SendTo - 0x45 frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x44	Byte	
Frame ID		Byte	A reference identifier used to match status responses. If set to **0**, the Transmit (TX) Status - 0x89 frame is disabled.
Socket ID		Byte	ID of the socket to send on.
Transmit options		Byte bit-field	Reserved
Payload		Variable	Data to be transferred to the destination, up to 1500 bytes.

Socket SendTo (Transmit Explicit Data): IPv4 - 0x45

Description

A Socket SendTo (Transmit Explicit Data) message causes the device to transmit data using an IPv4 address and port. For a non-zero frame ID, this will elicit a Transmit (TX) Status - 0x89 frame.

If this frame is used with a TCP, SSL, or a connected UDP socket, the address and port fields are ignored.

You must perform a Socket Bind/Listen - 0x46 frame for a UDP connection before you attempt a SendTo in order to assign a source port.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x45	Byte	
Frame ID		Byte	A reference identifier used to match status responses. If set to **0**, the Transmit (TX) Status - 0x89 frame is disabled.
Socket ID		Byte	ID of the socket to send on.
Destination address		32-bit big endian	
Destination port		16-bit big endian	
Transmit options		Byte bit-field	Reserved
Payload		Variable	Data to be transferred to the destination, up to 1500 bytes.

Socket Bind/Listen - 0x46

Description

Opens a listener socket that listens for incoming connections.

When there is an incoming connection on the listener socket, a Socket New IPv4 Client - 0xCC frame is sent, indicating the socket ID for the new connection along with the remote address information.

For a UDP socket, this frame binds the socket to a given port. A bound UDP socket can receive data with a Socket Receive From: IPv4 - 0xCE frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0x46	Byte	
Frame ID		Byte	A reference identifier used to match status responses. If set to **0**, the device does not send a response.
Socket ID		Byte	The socket ID to listen on.
Source port		16-bit big endian	The port to listen on.

Socket Listen Response - 0xC6

Description

The device sends this frame in response to a Socket Bind/Listen (0x46) frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xC6	Byte	
Frame ID		Byte	Resource identifier used to match status responses.
Socket ID		Byte	The socket ID of the socket that has started listening.
Status		Byte	Status code. See table below.

The following table shows the status codes.

Code	Description
0x00	Success
0x01	Invalid port
0x02	Error

Code	Description
0x03	Already bound or listening
0x20	Invalid socket ID

Socket New IPv4 Client - 0xCC

Description

The XBee Cellular modem generates this frame when an incoming connection is accepted on a listener socket.

This frame contains the original listener's socket ID and a new socket ID of the incoming connection, along with the connection's remote address information.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xCC	Byte	
Socket ID		Byte	The socket ID of the listener socket.
Client Socket ID		Byte	The socket ID of the new connection.
Remote address		32-bit big endian	
Remote port		16-bit big endian	

Socket Receive - 0xCD

Description

The XBee Cellular modem uses this frame when it receives RF data on the specified socket.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xCD	Byte	

Field name	Field value	Data type	Description
Frame ID		Byte	(Optional) This field allows for solicited reads to be in the future.
Socket ID		Byte	ID of the socket that the data has been received on.
Status		Byte bit-field	Reserved
Payload		Variable	Data received from the destination. It may be up to 1500 bytes.

Socket Receive From: IPv4 - 0xCE

Description

The XBee cellular modem uses this frame when it receives RF data on the specified socket. This frame is sent only for UDP sockets that have not used a Socket Connect - 0x42 frame to connect, providing addressing information about the source.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Field value	Data type	Description
Frame type	0xCE	Byte	
Frame ID		Byte	Optional: This field allows for solicited reads to be in the future.
Socket ID		Byte	ID of the socket that the data has been received on.
Source address		32-bit big endian	
Source port		16-bit big endian	
Status		Byte bit-field	Reserved
Payload		Variable	Data to be transferred to the destination, up to 1500 bytes.

Socket Status - 0xCF

Description

This frame is sent out the device's serial port to indicate the state related to the socket.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Field name	Size	Description
Frame type	1	Socket Status frame type (0xCF)
Socket ID	1	Socket ID for status reported
Status	1	0x00 = Connected All values other than **0x00 = Connected** are fatal and the Socket ID is closed and invalid after receipt. 0x01 = Failed DNS lookup 0x02 = Connection refused 0x03 = Transport closed 0x04 = Timed out 0x05 = Internal error 0x06 = Host unreachable 0x07 = Connection lost 0x08 = Unknown error 0x09 = Unknown server 0x0A = Resource error 0x0C = RST Close by peer 0x0D = Closed due to inactivity timeout 0x0E = PDP context deactivated by network

File system API frames

Local File System Request - 0x3B

Description

Access the XBee module's file system.

The frame content varies based on the File System Command sent in the request. Payloads for each command and their respective responses are included.

For more information about the file system, see File system.

Note The XBee modules responds to these requests with Local File System Response - 0xBB.

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 File System Request - 0x3B
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	File System Command	See File System Commands for valid command values.
6-n	variable	Request Parameters	Variable content based on File System Command.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

File System Commands

Value	Command
0x01	File Open
0x02	File Close
0x03	File Read
0x04	File Write
0x08	File Hash

Value	Command
0x10	Directory Create
0x11	Directory Open
0x12	Directory Close
0x13	Directory Read
0x1C	Get Path ID
0x2F	Delete
0x40	Volume Info
0x4F	Volume Format

Notes

- Multiple commands take a 16-bit Path ID, which allows the use of relative pathnames (using "/" as the path separator and using ".." to refer to a parent directory) as command parameters. The default of 0x0000 refers to the root directory (/). See the Get Path ID - 0x1C command for details on creation and use of temporary values in order to use relative pathnames.
- For the Directory Open and Get Path ID commands, using an empty Pathname field is equivalent to using "." – both refer to the directory designated by the Path ID.
- Request and Success Response describe the frame contents starting with the **File System Command** field (and excluding the **Checksum** field).
- Success Response lists the fields following the **Status** byte when 0 (indicating a successful operation), and is only listed for commands with additional fields after the **Status** byte.
- See Local File System Response - 0xBB for non-zero (error) **Status** values in the **Response**.
- Variable-length names are NOT null terminated. The frame length determines the length of the field.

File Open - 0x01

Description

Open a file for reading and/or writing.

- Requests must have at least READ or WRITE bit set in the **Options** field.
- Use the SECURE bit (0x80) of the **Options** byte to upload a write-only file (one that cannot be downloaded or viewed). This is useful for protecting MicroPython source code on the device.
- The SECURE bit is only valid when also setting the WRITE bit and either creating a new file (CREATE + EXCLUSIVE) or replacing an existing file (TRUNCATE).

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File Open - 0x01
6	16-bit	**Path ID**	See Get Path ID - 0x1C for a description.
8	8-bit	**Options**	Bitfield with the following values: ■ 0x01 CREATE: Create if file doesn't exist. ■ 0x02 EXCLUSIVE: Error out if file exists. ■ 0x04 READ: Open file for reading. ■ 0x08 WRITE: Open file for writing. ■ 0x10 TRUNCATE: Truncate file to 0 bytes. ■ 0x20 APPEND: Append to end of file. ■ 0x40 UNUSED: Unused, set to 0. ■ 0x80 SECURE: Create a secure write-only file.
9-n	variable	**File Name**	Pathname relative to Path ID.

Success Response

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File open - 0x01
6	8-bit	**Status**	Success - 0x00
7	16-bit	**File Handle**	Value used to reference file in later requests. Expires and becomes invalid if not referenced for over 2 minutes.
9	32-bit	**File Size**	File's size or `0xFFFFFFFF` if unknown.

File Close - 0x02

Description

Close an open file and release its File Handle.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File Close - 0x02
6	16-bit	**File Handle**	Value returned from File Open - 0x01 response.

File Read - 0x03

Description

Read the file.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File Read - 0x03
6	16-bit	**File Handle**	Value returned from File Open - 0x01 response.
8	32-bit	**Read Offset**	File position for read, or 0xFFFFFFFF to use the current position.
12	16-bit	**Bytes To Read**	Number of bytes to read from file, or 0xFFFF to read as many as possible (limited by file size or maximum response frame size).

Success Response

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File Read - 0x03
6	8-bit	**Status**	Success - 0x00
7	16-bit	**File Handle**	Value sent in request.
9	32-bit	**Data Offset**	Actual offset of data read from file.
13-n	variable	**Data**	Data read from the file.

File Hash - 0x08

Description

Returns a SHA256 hash to verify a file's contents without downloading the entire file. On XBee Cellular modules, there is a response delay in order to calculate the hash of a non-secure file.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File Hash - 0x08
6	16-bit	**Path ID**	See Get Path ID - 0x1C for a description.
8-n	variable	**File Name**	Pathname relative to **Path ID**.

Success Response

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File Hash - 0x08
6	8-bit	**Status**	Success - 0x0
7-38	32-bytes	**SHA256 Hash**	Hash used to verify file contents.

File Write - 0x04

Description

Write to the file.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File Write - 0x04
6	16-bit	**File Handle**	Value returned from File Open - 0x01 response.
8	32-bit	**Write Offset**	File position for write, or 0xFFFFFFFF to use the current position.
12-n	variable	**Data**	Data to write to file. If empty, frame just refreshes the **File Handle** timeout to keep the file open.

Success Response

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	File Write - 0x04
6	8-bit	**Status**	Success - 0x00
7	16-bit	**File Handle**	Value sent in request.
9	32-bit	**Current Offset**	Current offset of file after writing **Data** from **Request**.

Directory Create - 0x10

Description

Create a directory.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Directory Create - 0x10
6	16-bit	**Path ID**	See command Get Path ID - 0x1C for description.
8-n	variable	**Directory Name**	Pathname relative to **Path ID**. The parent directory of the directory to create must exist, for example, you must create all intermediate directories via separate requests.

Directory Open - 0x11

Description

Used with Directory Read to list files and directories in a given directory. To get a listing of entries in a directory:

1. Send a **Directory Open Request**.
2. Parse multiple entries from the **Response**.
3. If the last entry has the ENTRY_IS_LAST flag set, the listing is complete and the **Directory Handle** was automatically released.
4. If the listing is not complete, do one of the following:
   - Send a Directory Read Request to get additional directory entries
   - Send a Directory Close Request to release the Directory Handle.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Directory Open 0x10
6	16-bit	**Path ID**	See command Get Path ID - 0x1C for description.
8-n	variable	**Directory Name**	Pathname relative to **Path ID**, or empty to get a file listing for the **Path ID**.

Success Response

A **Directory Open Request** sends a response identical to a Directory Read Request. An empty directory returns a single entry with only the ENTRY_IS_LAST flag set, and a 0-byte **Entry Name**. A response ending with an ENTRY_IS_LAST flag automatically closes the Directory Handle.

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Directory Read - 0x13 or Directory Open - 0x11, depending on request.
6	16-bit	**Status**	Success - 0x00
7	16-bit	**Directory Handle**	Value returned in initial Directory Open Response.
9	32-bit	**File Size/Entry Flags**	File's size in lower 24 bits, combined with the following flags: ■ 0x80000000 (ENTRY_IS_DIR): Entry is a directory. ■ 0x40000000 (ENTRY_IS_SECURE): File is secure (write-only). ■ 0x01000000 (ENTRY_IS_LAST): This is the last entry. ■ Other flags in the top 8 bits (0x3E) are currently reserved and set to zero.
13-n	variable	**Entry Name**	File or directory name.
If there is enough room in the frame, there may be additional entries after the first.			
n+1	8-bit	**Null Terminator**	0x00 byte to separate entries
n+2	32-bit	**File Size and Flags**	Refer to description above.
n+6	variable	**Entry Name**	Refer to description above.

Process the entries in a **Directory Open Response** or Directory Read Response as follows:

- Split the **File Size and Flags** field into separate **File Size** and **Flags**.
- Look for a null terminator after the **File Size and Flags** field.
- Extract **Entry Name** as bytes after **File Size and Flags** and before either the null terminator or the end of the frame.
- Repeat this sequence if **Entry Name** had a null terminator and the packet contains unprocessed entries.
- If the final entry of the frame does not have ENTRY_IS_LAST set, send another Directory Read Request to get additional entries.

Directory Close - 0x12

Description

The host can send this frame to indicate that it is done reading the directory and no longer needs the **Directory Handle**. Note that the **Directory Handle** is automatically closed and no longer valid after receiving a **Response** with the ENTRY_IS_LAST flag set.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Directory Close - 0x12
6	16-bit	**Directory Handle**	Value returned in initial **Directory Open Response**.

Directory Read - 0x13

Description

Read entries from the directory.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Directory Read - 0x13
6	16-bit	**Directory Handle**	Value returned from previous Directory Open Response or **Directory Read Response**.

Success Response

A **Directory Read Request** sends a response identical to a Directory Open Request.

Get Path ID - 0x1C

Description

Many commands include a 16-bit field for the **Path ID**. If set to 0x0000, pathnames in the frame are relative to the root directory of the filesystem (/). Use the **Get Path ID** request to generate a **Path ID** for any subdirectory of the file system to allow the use of shorter relative pathnames in later requests.

- If the **Path ID** field of a Request is 0x0000, the **Response** contains a newly-allocated **Path ID** for use in later **Requests**.
- If the **Path ID** field of a **Request** is non-zero (such as one returned in a previous **Get Path ID Response**), the XBee module updates the path for that ID.
- To release a **Path ID** when no longer needed (instead of waiting for a timeout), send a **Request** with the **Path ID** and a single slash ("/") as the **Pathname**. Any **Get Path ID Request** that resolves to the root directory will release the **Path ID** and return a 0x0000 ID.
- Allocated **Path ID** values expire after 2 minutes if not used. You can refresh that timeout by sending a **Get Path ID** request with the **Path ID** and an empty or single period (".") **Pathname**.
- The full, absolute path of the **Path ID** is included in the Response only if can fit. Any code used to process the response needs to take that into account and handle an empty **Full Pathname** field.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Get Path ID - 0x1C
6	16-bit	**Path ID**	Either 0x0000 to create a new **Path ID**, or an existing **Path ID** to update its location.
8-n	variable	**Pathname**	Pathname relative to **Path ID**.

Success Response

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Get Path ID x 0x1C
6	8-bit	**Status**	0x00 - Success
7	16-bit	**Path ID**	Value to use in later **File System Requests** with relative pathnames.
9-n	variable	**Full Pathname**	If short enough to fit in the frame, the full pathname (starting with "/flash"). Deep subdirectories may return an empty field instead of their **Full Pathname**. The **Full Pathname** will never exceed 255 characters.

Delete - 0x2F

Description

Delete files or a directory. The entry must delete all files in a directory before you can delete the directory.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Delete - 0x2F
6	16-bit	**Path ID**	See Get Path ID - 0x1C for description
8-n	variable	**Path Name**	**Pathname** of file or empty directory to delete.

Volume Info - 0x40

Description

Get volume information: used space, available space, and unusable bytes on volume.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Volume Info - 0x40
6-n	variable	**Volume Name**	Name of volume to report on. Currently /flash is the only supported value.

Success Response

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Volume Info - 0x40
6	16-bit	**Status**	Success - 0x00
7	32-bit	**Used Bytes**	Used space on volume.
11	32-bit	**Free Bytes**	Available space on volume.
15	32-bit	**Bad Bytes**	Unusable bytes on volume.

Volume Format - 0x4F

Description

Format the space allocated to file storage. This command sends a **Volume Info Success Response** when the format completes.

Request

Offset	Size	Frame Field	Description
5	8-bit	**File System Command**	Volume Format - 0x4F
6-n	variable	**Volume Name**	Name of volume to format. Currently /flash is the only supported value.

Local File System Response - 0xBB

Description

The XBee module sends this frame in response to a Local File System Request (0x3B) frame sent with a non-zero **Frame ID**. The contents of the variable-length **Response Data** field appear in the documentation for each **File System Command**.

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 File System Response - 0xBB
4	8-bit	**Frame ID**	Frame ID value from the corresponding Local File System Request.
5	8-bit	**File System Command**	See File System Commands for valid command values.
6	8-bit	**Status**	See Status Values for description.
7-n	variable	**Response Data**	Variable content based on **File System Command**. Only present if **Status** is 0 and the command has additional data to provide.
EOF	8-bit	**Checksum**	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Status Values

Value	Command
0x00	Success
0x01	Error
0x02	Invalid File System Command
0x03	Invalid command parameter
0x50	Access denied
0x51	File/Directory already exists
0x52	File/Directory does not exist
0x53	Invalid name
0x54	File operation on directory
0x55	Cannot delete non-empty directory
0x56	Attempt to read past EOF (end of file)

Value	Command
0x57	Hardware failure
0x58	Volume offline/format required
0x59	Volume full
0x5A	Operation timed out
0x5B	Busy (wait for prior command to complete then try again)
0x5C	Resource failure (memory allocation failed, try again)

Regulatory firmware

You can install a regulatory firmware version onto your XBee for regulatory compliance testing of your Bluetooth and cellular radio components.

Note This firmware is to be used only for regulatory compliance testing. When you install the regulatory firmware, most XBee Cellular features are disabled, as this firmware is NOT meant to be a full-featured firmware used in production, and use of the regulatory firmware version in production is not supported.

When you install the regulatory firmware on your XBee, the current device firmware is overwritten. After regulatory testing is complete, you will have to reinstall the device firmware to return to full functionality.

The table below shows a list of features that are supported in the regulatory firmware.

Feature	Description
Firmware upgrade	Use XCTU or Digi Remote Manager to upgrade the device to or from the regulatory testing firmware.
Command mode	Use +++ to switch between bypass mode, DTM protocol, and other configurations.
Bypass mode	Bypass mode is available through the primary UART when configured with ATAP=5.
USB Direct	You can configure USB Direct for use with ATP1=7. In addition, you must enable VBUS (ATDO=5).
Bluetooth DTM Protocol	To be able to issue DTM Commands over the primary UART, configure the module with ATAP=0.

Install the regulatory firmware

You can install the regulatory firmware from either XCTU or Remote Manager.

Install regulatory firmware using XCTU

You can install the regulatory firmware on your XBee using XCTU.

Note After you have completed your testing using the regulatory firmware, you should re-install the device firmware.

1. Add your XBee device to XCTU if you haven't already done so.
2. From within XCTU, click the **Configuration working modes** button .
3. From the **Radio Modules** list, select the device that you want to update.
4. Click **Update firmware**. The **Update the radio module firmware** dialog appears and displays the available and compatible device firmware for the selected XBee module.
5. Select the product family XBXC3, the function set including the name Regulatory, and then the newest firmware listed.
6. Click **Update** to update the device firmware.
7. Once the regulatory firmware is loaded, configure the XBee for the testing required.
   - Configure regulatory firmware for testing the Bluetooth radio.
   - Configure regulatory firmware for testing the cellular component.
8. After you have completed your testing using the regulatory firmware, you should re-install the device firmware.

Install regulatory firmware using Remote Manager

You can install the regulatory firmware on your XBee from Remote Manager.

Note After you have completed your testing using the regulatory firmware, you should re-install the device firmware.

Note Digi Remote Manager requires TCP and will not work with NB-IoT, unless the carrier supports TCP.

WARNING! The firmware version 1140F reorganizes the product's flash memory and upgrades the product to version 11410. You cannot downgrade to a version earlier than 11410 after installing 1140F/11410.

To perform a firmware update:

1. Download the updated firmware file for your device from Digi's support site.
   a. Go to the Digi XBee 3 Cellular LTE-M support page.
   b. Scroll down to the **Firmware Updates** section.
   c. Locate and click **Digi XBee 3 Cellular LTE-M/NB-IoT Regulatory Firmware** to download the zip file.
   d. Unzip the file.
2. Log into Remote Manager.
3. Click the arrow next to your user name and select **Open Classic Remote Manager**.
4. In your Remote Manager account, click **Device Management > Devices**.
5. Select the first device you want to update. To select multiple devices (must be of the same type), press the Control key and select additional devices.

6. Click **More** in the **Devices** toolbar and select **More** > **Update** > **Update Firmware**. The **Update Firmware** dialog appears.
7. Click **Browse** to select the file that you unzipped earlier.
8. Click **Update Firmware**. The updated devices automatically reboot when the updates are complete.

   Note The update is immediately rejected and an error is returned if the device is going into sleep mode or is being shut down. See Clean shutdown.

9. When all changes are complete, disconnect the device from Remote Manager.
10. Once the regulatory firmware is loaded, configure the XBee for the testing required.
    - Configure regulatory firmware for testing the Bluetooth radio.
    - Configure regulatory firmware for testing the cellular component.
11. After you have completed your testing using the regulatory firmware, you should re-install the device firmware.

Configure regulatory firmware for testing the Bluetooth radio

In XCTU or command mode, set the following configurations:

1. Turn on Bluetooth by setting BT=1.
2. Set the API mode to DTM Protocol by setting AP=0.
3. Specify cellular modem as on or off:
   - If the cellular modem should be off during testing set AM=1.
   - If the cellular modem should be on during testing set AM=0.
4. Verify that the serial interface settings are set up as expected for testing.

Configure regulatory firmware for testing the cellular component

You must configure the regulatory firmware in order to use the regulatory test commands for testing the cellular component.

Prerequisite

A SIM card must be installed in the XBee.

Configure the regulatory firmware

In XCTU or command mode, set the following configurations:

1. If Bluetooth is not required, turn off Bluetooth by setting BT=0.
2. Disable USB Direct mode, if it is currently enabled. Set P1 = 0.
3. Disable airplane mode, if it is currently enabled. Set AM = 0.
4. Configure the XBee in transparent mode. Set AP = 0.

Bluetooth DTM protocol

The Bluetooth DTM protocol is implemented as specified in Volume 6 part F of the Bluetooth 5 specification.

All commands are two bytes long (16-bits) and receive a response, which is also two bytes long. All multi-byte sequences are big endian.

The protocol has also been extended with the two commands shown in the table below to allow changing the transmit power and to override the packet type to use an unmodulated carrier.

Description	Command (2-bits)	Control (6-bits)	Parameter (6-bits)	DC (2-bits)
Set the transmit power. The response is the actual transmit power that is set, which may be less than the requested if the radio does not support the requested transmit power.	Setup 0	6	Transmit power in dBm (only 1 dB resolution available) ranging from 0 to 17 dBm.	N/A leave as 0
Override the packet type. This will supersede the packet type specified in the transmitter/receiver test commands.	Setup 0	7	Packet Type Override ■ 0: No Override ■ 1: Packet will be an unmodulated carrier	N/A leave as 0

Example

Example of a typical test sequence.

Description	Command	Response
Set transmit power to 10 dBm	06 28	00 14
Set PHY to LE 1M	02 04	00 00
Override the packet type	07 04	00 00
Start TX test at 2402 MHz	80 FD	00 00
End Test	80 00	80 00

Regulatory testing commands

The regulatory commands are used for regulatory compliance testing of your Bluetooth and cellular radio components. The commands work in conjunction with regulatory firmware which you must install onto your XBee before you use these commands.

Use the commands

Before you can use these commands for regulatory compliance testing, you must do the following:

1. Install a regulatory firmware version onto your XBee. See Regulatory firmware.
2. Enable test mode. See %# (Enable/disable test mode).
3. Start test mode. See %1 (Start test mode).
4. Perform regulatory tests, using the regulatory commands.
5. Stop test mode. See %2 (Stop test mode)
6. Disable test mode. See %# (Enable/disable test mode).

Regulatory command reference

As you use the commands, be aware of the following:

- After each test command %(1-D) the status command %? should be queried until it returns the expected result, and testing should not proceed until the module reports that it is in the desired mode. For entering Receive Mode this can take up to two minutes while the module is reconfigured. During this time, you may see the module in the error (5) state temporarily.
- If the error state persists, or the status value persistently changes between 1 and 5, double-check that the channel number (AT%8) and power (AT%A) settings are appropriate for the module and test being performed. For example, you should not attempt to use a downlink channel for the transmit test, as the cellular component will not successfully enter test mode.

%# (Enable/disable test mode)

Use this command to enable and disable test mode. When disabled, many non-regulatory testing features are also disabled, such as the ability to create sockets. For a list of the features that are available, see Regulatory firmware.

Prior to enabling test mode, it is recommended that you set the module to factory default settings to ensure best results. When test mode is enabled, the module will report an AI value of 0x31.

Parameter range

Value	Description
0	Disables test mode.
1	Enables test mode.

Default

Disabled

Examples

Enable test mode:

```
AT%#1
```

Disable test mode:

```
AT%#0
```

%1 (Start test mode)

Use this command to start test mode. You must perform this command at least once before you perform any other regulatory command.

Examples

Start test mode:

```
AT%1
```

%2 (Stop test mode)

Use this command to stop test mode. Any active test operation currently in progress is stopped.

If you do not stop test mode after you have completed regulatory testing, normal cellular component features will not be available.

Example

Stop test mode:

```
AT%2
```

%5 (Start modulated transmit)

Use this command to start modulated transmit using the EARFCN and power specified by AT%7 and AT%9.

This command works in conjunction with %6 (Stop transmit).

Examples

Start modulated transmit:

```
AT%5
```

%6 (Stop transmit)

Stop modulated transmit. This command works in conjunction with %5 (Start modulated transmit).

Example

Stop transmit:

```
AT%6
```

%7 (Set EARFCN)

Use this command to set the EARFCN (Absolute Radio Frequency Channel Number).

Parameter range

0 - 65535

This is specified in decimal to conform to standard representations found in specifications without need for translation.

Default

N/A

Examples

Set EARFCN to 5110:

```
AT%75110
```

Set EARFCN to 23010:

```
AT%723010
```

%8 (Get the EARFCN)

Use this command to get the EARFCN (Absolute Radio Frequency Channel Number) that was set using AT%7.

Parameter range

N/A

Example

Get the channel number:

```
AT%8
```

%9 (Set transmit power)

Use this command to set the transmit power.

Parameter range

0-FFF hexadecimal

Variant range

-40 to 24 dBm

Value is in sixteenth dBm (1/16) fixed point and is represented as a 12-bit twos-complement integer. The XBee 3 LTE-M module only accepts whole integer dBm and will truncate the fractional portion.

Default

N/A

Examples

Set transmit power to 0 dBm:

```
AT%9000
```

Set transmit power to -1 dBm:

```
AT%9FF0
```

%A (Get transmit power)

Use this command to get the transmit power value set using AT%9.

Parameter range

N/A

Example

Get transmit power:

```
AT%A
```

%D (Start receive mode)

Use this command to start receive mode on the EARFCN channel specified using AT%7.

Parameter range

N/A

Examples

Start receive mode:

```
AT%D
```

%? (Query test state)

Use this command to query test state.

Parameter range

Value	Description
0	Inactive (Test mode not yet started.)
1	Transition (Attempting to activate test mode.)
2	Off (Test mode started, but no active test.)
3	Receive mode
4	Transmit mode
5	An error occurred

Example

Query test state:

```
AT%?
```

Troubleshooting

This section contains troubleshooting steps for the XBee Smart Modem.

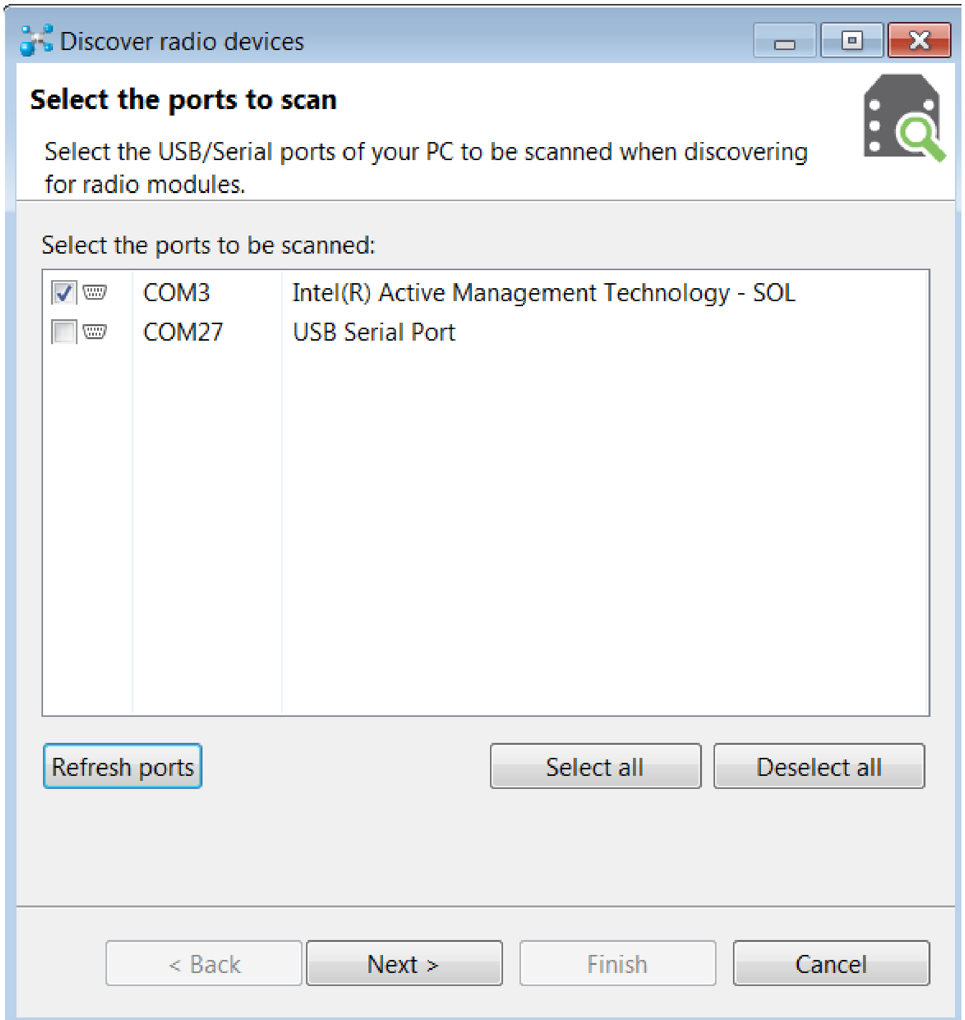
Cannot find the serial port for the device

Condition

In XCTU, the serial port that your device is connected to does not appear.

Solution

1. Click the **Discover radio modules** button .
2. Select all of the ports to be scanned.
3. Click **Next** and then **Finish**. A dialog notifies you of the devices discovered and their details.



4. Remove the development board from the USB port and view which port name no longer appears in the **Discover radio devices** list of ports. The port name that no longer appears is the correct port for the development board.

Other possible issues

Other reasons that the XBee Smart Modem is not discoverable include:

1. If you accidentally have the loopback pins jumpered.
2. You may not have a driver installed. If you do not have a driver installed, the item will have an exclamation point icon next to it in the Windows Device Manager.
3. You may not be using an updated FTDI driver.
   a. Click here to download the drivers for your operating system.
   b. This may require you to reboot your computer.
   c. Disconnect the power and USB from the XBIB-CU-TH board and reconnect it
4. If you have a driver installed and updated but still have issues, on Windows 10 you may have to enable VCP on the driver; see Enable Virtual COM port (VCP) on the driver.

Enable Virtual COM port (VCP) on the driver

On Windows 10 computers, if XCTU does not see the devices you have attached to a PC, you may need to enable VCP on the USB driver.

To enable VCP:

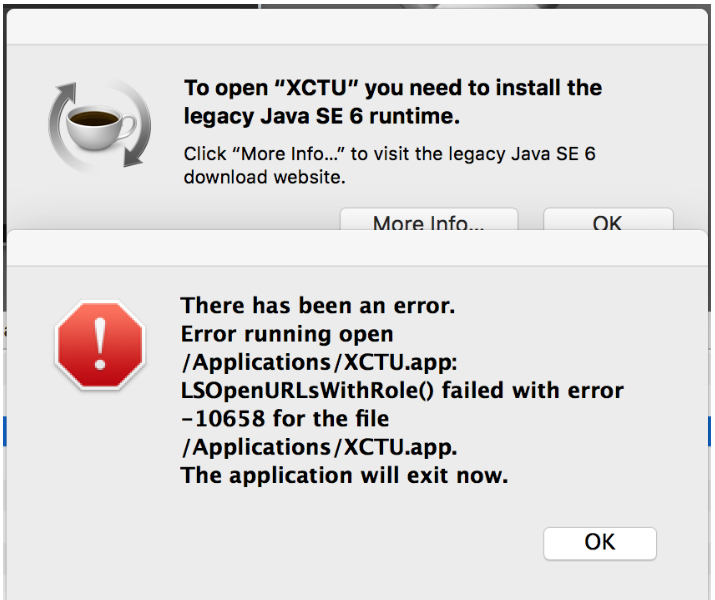
1. Click the **Search** button.
2. Type **Device Manager** to search for it.
3. Click **Universal Serial Bus controllers**.
4. If it displays more than one USB controller, unplug the XBee Smart Modem and plug it back in to make sure you choose the correct one.
5. Right-click the USB controller and select **Properties**; a dialog displays.
6. Select the **Advanced** tab.
7. Check **Load VCP**.
8. Click **OK**.
9. Unplug the board and plug it back in.

Correct a macOS Java error

When you use XCTU on macOS computer, you may encounter a Java error.

Condition

When opening XCTU for the first time on a macOS computer, you may see the following error:



Solution

1. Click **More info** to open a browser window.
2. Click **Download** to get the file javaforosx.dmg.
3. Double-click on the downloaded javaforosx.dmg.
4. In the dialog, double-click the JavaForOSX.pkg and follow the instructions to install Java.

Unresponsive cellular component in Bypass mode

When in Bypass mode, the XBee Smart Modem does not automatically reset or reboot the cellular component if it becomes unresponsive.

Condition

In Bypass mode, the XBee Smart Modem does not respond to commands.

Solution

1. Query the AI (Association Indication) parameter to determine whether the cellular component is connected to the XBee Smart Modem software. If **AI** is **0x2F**, Bypass mode should work. If not, look at the status codes in AI (Association Indication) for guidance.
2. Ensure that you set DO (Device Options) bit 3 to **0** before entering Bypass mode.
3. You can send the !R (Modem Reset) command to reset only the cellular component.

Syntax error at line 1

You may get a **syntax error at line 1** error after pasting example MicroPython code and pressing **Ctrl**+**D**.

Solution

This commonly happens when you accidentally type a character at the beginning of line 1 before pasting the code.

Error Failed to send SMS

In MicroPython, you consistently get **Error Failed to send SMS** messages.

Note For NB-IoT, SMS support is dependent on the network. Contact your network provider for details.

Solution

Your device cannot connect to the cell network. The reason may be:

1. The antenna is improperly or loosely connected.
2. The device is at a location where cellular service cannot reach. If the device is connected to the network, the red LED blinks about twice in a second. If it is not connected it does not blink; see Associate LED functionality.
3. You SIM card is out of SMS text quota.
4. The device is not getting enough current, for example if power is being supplied only by USB to the XBIB development board, rather than using an additional external power supply.

Network connection issues

Condition

The XBee Smart Modem is not joining the network, AI (Association Indication) is cycling between **0xFF** (Initalizing), **0x22** (Registering to Cellular Network) and **0x25** (Cellular Network Registration Denied).

Solution

Some things to check are:

- The antennas are connected correctly to the device.
- The SIM card is seated properly in the device.
- Use ATRJ to view the reject cause.
- APN is set correctly.
- If you are using a SIM card not provided by Digi, or a SIM card capable of roaming between multiple networks, you may need to adjust the modem's network configuration. Set ATCP to **1 (No Profile)**, and set ATBM and/or ATBN to select only the bands supported by the carrier you wish to connect to. Consult with your carrier to determine the necessary bands.
- If you had to change a modem parameter, check to make sure you have reset the module using the FR command or by pressing the reset button, especially if you have changed settings such as CP, N#, BM, or BN.

Note The default APN configured in the kit should allow the XBee Smart Modem to get on the network with the SIM included in the kit. However, you may program the APN explicitly to **10569.mcs** if you are having difficulty registering with the network.

Hardware flow control in Bypass mode

Support for hardware flow control in Bypass mode is available with version L0.0.00.00.05.08,A.02.04 of the SARA-R410M cellular component. Digi recommends that you update to at least firmware version L0.0.00.00.05.08,A.02.04 of the cellular component (using a serial port or from Remote Manager) and firmware version 11414 of the XBee Smart Modem.

Note Earlier versions of the SARA-R410M firmware did not support hardware flow control. The lack of hardware flow control may lead to data loss while in Bypass operating mode during periods of high UART traffic.

Baud rate in Bypass mode

If you change the AT+IPR setting of the cellular component away from its default you will lose communication with the cellular component while in Bypass mode.

In firmware version *14 and later, the IB (Cellular Component Baud Rate) command was added to control the baud rate to the cellular component. If you change the baud rate of the cellular component using the AT+IPR setting you will need to match it with the IB setting to maintain communication.

Note Digi does not recommend using bypass mode. You should use USB Direct mode instead.

Socket leaks

This applies to devices with the u-blox SARA-R410 cellular component used in the XBee 3 Cellular LTE-M Global Smart Modem.

Condition

There are instances where a socket leaks when closing a connection while there is pending RX data. This state is reported whenever the number of leaked sockets causes an inability to create a new connection.

For Transparent and API modes the XBee Smart Modem returns specific status responses to describe that you need to perform a hard reset to recover the leaked sockets. These statuses are:

Mode	Response
Transparent mode	CI (Protocol/Connection Indication) reports **0x13** Socket leak
API mode	Transmit (TX) Status - 0x89 status **0x32** Resource error (retry operation later)
MicroPython	When attempting to make a connection to a server you receive **errno.ENFILE (7023)**

Solution

To recover the leaked sockets and make them available again in all modes, you should perform a reset of the cellular component (hard or soft).

Modem fails to go into PSM dormant state

Note This behavior does not impact configurations which do not use PSM.

Condition

The modem may fail to enter the PSM dormant state when there are open TLS sockets. This includes any connection to Digi Remote Manager, if ATMO bit 0 is set.

This situation may happen only when the XBee firmware version is earlier than 1141B and the modem firmware version is earlier than L0.0.00.00.05.12,A.02.21.

Solution

Close any TLS connections when PSM dormant state is desired.

Regulatory Information

Antenna regulatory information: FCC and ISED

The equipment can be installed using antennas and cables constructed with non-standard connectors (RPSMA, RPTNC, and so forth). An adapter cable may be necessary to attach the XBee connector to the antenna connector.

The modules are approved by FCC and ISED for fixed base station and mobile applications for the channels indicated in the tables below. If the antenna is mounted at least 21 cm from nearby persons, the application is considered a mobile application.

Note For the Global LTE-M/NB-IOT (LowPower) variant: If the antenna is mounted at least 20 cm from nearby persons, the application is considered a mobile application.

The antennas below have been approved for use with this module. Digi does not carry all of these antenna variants. Contact Digi Sales for available antennas.

Bluetooth antennas

The following antennas are approved for use with the Bluetooth radio by the FCC and by ISED.

Part number	Type (description)	Gain	Application
31000020-02	Integral antenna	-2.5 dBi	Fixed/Mobile
29000095	Dipole (Half-wave articulated RPSMA - 4.5")	2.1 dBi	Fixed/Mobile
A24-HASM-450	Dipole (Half-wave articulated RPSMA-4.5")	2.1 dBi	Fixed/Mobile
A24-HABSM	Dipole (Articulated RPSMA)	2.1 dBi	Fixed
A24-HABUF-P5I	Dipole (Half-wave bulkhead mount U.FL w/ 5" pigtail)	2.1 dBi	Fixed
A24-HASM-525	Dipole (Half-wave articulated RPSMA-5.25")	2.1 dBi	Fixed/Mobile
29000812	Flexible PCB, U.FL w/ 200mm pigtail	4.4 dBi	Fixed/Mobile
FXP74.07.0100A	Taoglas FXP74 Black Diamond 2.4GHz Band Antenna	4.0 dBi	Fixed/Mobile
W3921B0100	Pulse 2400-2500MHz FPC dipole	1 dBi	Fixed/Mobile
W3525B039	Pulse 2.4-2.5GHz PCB Antenna	1.5 dBi	Fixed/Mobile

Cellular antennas

The gain of the system antenna (i.e. the combined transmission line, connector, cable losses and radiating element gain) must not exceed the values below for mobile and fixed or mobile operating configurations.

u-blox SARA-R410M-01B

- 3.67 dBi in 700 MHz, i.e. LTE FDD-12 band
- 4.10 dBi in 850 MHz, i.e. LTE FDD-5 band
- 6.74 dBi in 1700 MHz, i.e. LTE FDD-4 band
- 7.12 dBi in 1900 MHz, i.e. LTE FDD-2 band

u-blox SARA-R410M-02B

- 3.66 dBi in 700 MHz, i.e. LTE FDD-12 band
- 3.94 dBi in 750 MHz, i.e. LTE FDD-13 band
- 4.41 dBi in 850 MHz, i.e. LTE FDD-5 band
- 6.75 dBi in 1700 MHz, i.e. LTE FDD-4 band
- 7.00 dBi in 1900 MHz, i.e. LTE FDD-2 band
- 9.40 dBi in 1900 MHz, i.e. LTE FDD-25 band

United States (FCC)

FCC requirements

XBee Smart Modems comply with Part 15 of the FCC rules and regulations. Compliance with the labeling requirements, FCC notices and antenna usage guidelines is required.

To fulfill FCC Certification, the OEM must comply with the following regulations:

1. The system integrator must ensure that the text on the external label provided with this device is placed on the outside of the final product.
2. RF Modules may only be used with antennas that have been tested and approved for use with the modules.

OEM labeling requirements

WARNING! As an Original Equipment Manufacturer (OEM) you must ensure that FCC labeling requirements are met. You must include a clearly visible label on the outside of the final product enclosure that displays the following content:

Required FCC Label for OEM products containing the XBee 3 Cellular LTE-M RF Module

Contains FCC ID: MCQ-XB3M1

Contains FCC ID: XPY2AGQN4NNN

The enclosed device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (i.) this device may not cause harmful interference and (ii.) this device must accept any interference received, including interference that may cause undesired operation.

FCC notices

IMPORTANT: XBee modules have been certified by the FCC for use with other products without any further certification (as per FCC section 2.1091). Modifications not expressly approved by Digi could void the user's authority to operate the equipment.

IMPORTANT: OEMs must test final product to comply with unintentional radiators (FCC section 15.107 & 15.109) before declaring compliance of their final product to Part 15 of the FCC Rules.

IMPORTANT: The RF module has been certified for remote and base radio applications. If the module will be used for portable applications, the device must undergo SAR testing.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures: Re-orient or relocate the receiving antenna, Increase the separation between the equipment and receiver, Connect equipment and receiver to outlets on different circuits, or Consult the dealer or an experienced radio/TV technician for help.

RF exposure

If you are an integrating the XBee 3 RF Module into another product, you must include the following Caution statement in OEM product manuals to alert users of FCC RF exposure compliance:

Global LTE-M/NB-IOT

CAUTION! To satisfy FCC RF exposure requirements for mobile transmitting devices, a separation distance of 25 cm or more should be maintained between the antenna of this device and persons during device operation. To ensure compliance, operations at closer than this distance are not recommended. The antenna used for this transmitter must not be co-located in conjunction with any other antenna or transmitter.

FCC publication 996369 related information

In publication 996369 section D03, the FCC requires information concerning a module to be presented by OEM manufacturers. This section assists in answering or fulfilling these requirements.

2.1 General

No requirements are associated with this section.

2.2 List of applicable FCC rules

This module conforms to FCC Parts 27(cellular).

2.3 Summarize the specific operational use conditions

Certain approved antennas require attenuation for operation. For the XBee Smart Modem, see Antenna regulatory information: FCC and ISED.

Host product user guides should include the antenna table if end customers are permitted to select antennas.

2.4 Limited module procedures

Not applicable.

2.5 Trace antenna designs

While it is possible to build a trace antenna into the host PCB, this requires at least a Class II permissive change to the FCC grant which includes significant extra testing and cost. If an embedded trace or chip antenna is desired contact a Digi sales representative for information on how to engage with a lab to get the modified FCC grant.

2.6 RF exposure considerations

For RF exposure considerations see RF exposure.

Host product manufacturers need to provide end-users a copy of the “RF Exposure” section of the manual: RF exposure.

2.7 Antennas

A list of approved antennas is provided for the XBee Smart Modem. See Antenna regulatory information: FCC and ISED.

2.8 Label and compliance information

Host product manufacturers need to follow the sticker guidelines outlined in OEM labeling requirements.

2.9 Information on test modes and additional testing requirements

Contact a sales representative for information on how to configure test modes for the XBee Smart Modem.

2.10 Additional testing, Part 15 Subpart B disclaimer

All final host products must be tested to be compliant to FCC Part 15 Subpart B standards. While the XBee Smart Modem was tested to be complaint to FCC unintentional radiator standards, FCC Part 15 Subpart B compliance testing is still required for the final host product. This testing is required for all end products, and XBee Smart Modem Part 15 Subpart B compliance does not affirm the end product’s compliance.

See FCC notices.

Europe (CE)

If the XBee Smart Modems is incorporated into a product, the manufacturer must ensure compliance of the final product with articles 3.1a and 3.1b of the Radio Equipment Directive. A Declaration of Conformity must be issued for each of these standards and kept on file as described in the Radio Equipment Directive.

Furthermore, the manufacturer must maintain a copy of the XBee Smart Modem user guide documentation and ensure the final product does not exceed the specified power ratings, antenna specifications, and/or installation requirements as specified in the user guide.

Maximum power and frequency specifications

When using the Bluetooth low energy technology RF physical layer:

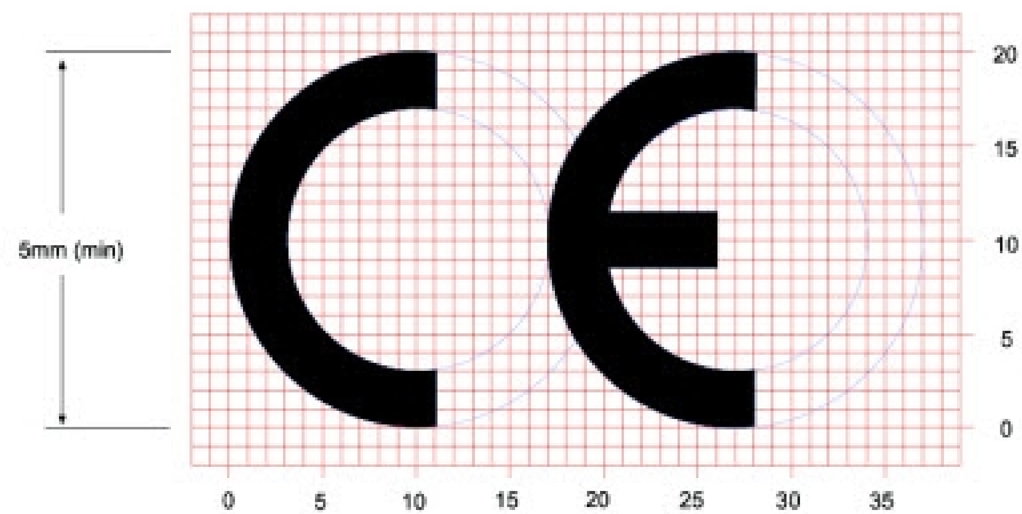
- Maximum power: 9.02 mW (9.55 dBm) Equivalent Isotropically Radiated Power (EIRP).
- Frequencies: 2 MHz channel spacing, beginning at 2402 MHz and ending at 2480 MHz.

CE and UKCA OEM labeling requirements

The CE and UKCA markings must be clearly visible and legible when you affix it to the product. If this is not possible, you must attach these marks to the packaging (if any) or accompanying documents.

CE labeling requirements

The “CE” marking must be affixed to a visible location on the OEM product. The following figure shows CE labeling requirements.



The CE mark shall consist of the initials “CE” taking the following form:

- If the CE marking is reduced or enlarged, the proportions given in the above graduated drawing must be respected.
- The CE marking must have a height of at least 5 mm except where this is not possible on account of the nature of the apparatus.
- The CE marking must be affixed visibly, legibly, and indelibly.

UK Conformity Assessed (UKCA) labeling requirements

See guidance/using-the-ukca-marking for further details.

You must make sure that:

- If you reduce or enlarge the size of your marking, the letters forming the UKCA marking must be in proportion to the version set out below.
- The UKCA marking is at least 5 mm in height – unless a different minimum dimension is specified in the relevant legislation.
- The UKCA marking is easily visible, legible (from 1 January 2023 it must be permanently attached).
- The UKCA marking can take different forms (for example, the color does not have to be solid), as long as it remains visible, legible and maintains the required proportions.

Declarations of conformity

Digi has issued Declarations of Conformity for the XBee Smart Modem concerning emissions, EMC, and safety. For more information, see www.digi.com/resources/certifications.

Important note

Digi customers assume full responsibility for learning and meeting the required guidelines for each country in their distribution market. Refer to the radio regulatory agency in the desired countries of operation for more information.

Approved antennas

Testing for use of the **Digi XBee® 3 Cellular LTE-M/NB-IoT Global Smart Modem** (Bluetooth) in the European markets was performed with a 2.1 dBi dipole antenna.

Use of an antenna with a gain of 2.1 dBi or less will ensure compliance with the spectrum requirements of the RED (CE) and UKCA.

The following antennas have been tested for use with the **Digi XBee® 3 Cellular LTE-M/NB-IoT Global Smart Modem**.

Consult with an antenna manufacturer for an equivalent option.

Antenna	Description
2900009	Dipole (Half-wave articulated RPSMA - 4.5”)
A24-HASM-450	Dipole (Half-wave articulated RPSMA-4.5")

Antenna	Description
A24-HABSM	Dipole (Articulated RPSMA)
A24-HABUF-P5I	Dipole (Half-wave bulkhead mount U.FL w/ 5" pigtail)
A24-HASM-525	Dipole (Half-wave articulated RPSMA-5.25")

Innovation, Science and Economic Development Canada (ISED)

Labeling requirements

Labeling requirements for Industry Canada are similar to those of the FCC. A clearly visible label on the outside of the final product enclosure must display the following text.

Contains IC: 1846A-XB3M1

Contains IC: 8595A-2AGQN4NNN

The integrator is responsible for its product to comply with IC ICES-003 & FCC Part 15, Sub. B - Unintentional Radiators. ICES-003 is the same as FCC Part 15 Sub. B and Industry Canada accepts FCC test report or CISPR 32 test report for compliance with ICES-003.

This device complies with Industry Canada licence-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.

Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes: (1) l'appareil ne doit pas produire de brouillage, et (2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

RF Exposure

CAUTION! This equipment is approved for mobile and base station transmitting devices only. Antenna(s) used for this transmitter must be installed to provide a separation distance of at least 25 cm from all persons and must not be co-located or operating in conjunction with any other antenna or transmitter.

ATTENTION! Cet équipement est approuvé pour la mobile et la station base dispositifs d'émission seulement. Antenne(s) utilisé pour cet émetteur doit être installé pour fournir une distance de séparation d'au moins 25 cm à partir de toutes les personnes et ne doit pas être situé ou fonctionner en conjonction avec tout autre antenne ou émetteur.

Transmitters with Detachable Antennas

This radio transmitter has been approved by Industry Canada to operate with the antenna types listed in Antenna regulatory information: IC (Canada) with the maximum permissible gain and required antenna impedance for each antenna type indicated. Antenna types not included in this list, having a gain greater than the maximum gain indicated for that type, are strictly prohibited for use with this device.

Le présent émetteur radio a été approuvé par Industrie Canada pour fonctionner avec les types d'antenne énumérés ci-dessous et ayant un gain admissible maximal et l'impédance requise pour chaque type d'antenne. Les types d'antenne non inclus dans cette liste, ou dont le gain est supérieur au gain maximal indiqué, sont strictement interdits pour l'exploitation de l'émetteur.

Detachable Antenna

Under Industry Canada regulations, this radio transmitter may operate using only an antenna of a type and maximum (or lesser) gain approved for the transmitter by Industry Canada. To reduce potential radio interference to other users, the antenna type and its gain should be so chosen that the equivalent isotropically radiated power (EIRP) is not more than that necessary for successful communication.

Conformément à la réglementation d'Industrie Canada, le présent émetteur radio peutfonctionner avec une antenne d'un type et d'un gain maximal (ou inférieur) approuvépour l'émetteur par Industrie Canada. Dans le but de réduire les risques de brouillageradioélectrique à l'intention des autres utilisateurs, il faut choisir le type d'antenne etson gain de sorte que la puissance isotrope rayonnée équivalente (p.i.r.e.) ne dépassepas l'intensité nécessaire àl'établissement d'une communication satisfaisante.

Brazil (ANATEL)

The XBee® 3 Cellular LTE-M/NB-IoT Smart Modem radio modules comply with the requirements of ANATEL to be used in Brazil. The customer's product is subject to testing for conformity to the same Brazilian standards.

Modelo: XB3M1

Para maiores informações, consulte o site da ANATEL www.anatel.gov.br.

Este equipamento não tem direito à proteção contra interferência prejudicial e não pode causar interferência em sistemas devidamente autorizados.

ANATEL Agência Nacional de Telecomunicações

ANATEL: 18681-20-01209

ANATEL: 07889-19-05903

Mexico (IFETEL)

Mexico does not have a "modular" approval for radio products.

If XBee Modules are integrated into a product, the manufacturer must ensure complicance of the final product with all the requirements of Mexican regulations.

Mexico IFETEL

Manufacturer: Digi International

Country: USA

Brand: Digi

Tariff Code (HS): 8517-62-01

Model: XB3M1

IFETEL (IFT) number RCPDIX21-1453 applies to this XBee3 radio:

- XB3-C-A2-UT-001

OEM labeling requirements

WARNING! The Original Equipment Manufacturer (OEM) must ensure that Mexico IFT labeling requirements are met.

The IFETEL number for the Digi XBee® 3 Cellular LTE-M/NB-IoT product must be listed either on the end product, on the packaging, in the manual, or in the software with the following phrase:

“Este equipo contiene el módulo XB3M1 con Número IFETEL: RCPDIXB20-2543”

or

“Este equipo contiene el módulo XB3M1 con IFT #: RCPDIXB20-2543”

The following paragraph must also be present in the User Manual for the end product:

“La operación de este equipo está sujeta a las siguientes dos condiciones: (1) es posible que este equipo o dispositivo no cause interferencia perjudicial y (2) este equipo o dispositivo debe aceptar cualquier interferencia, incluyendo la que pueda causar su operación no deseada.”

Deprecated kit (PN XK3-C-A2-UT-U or XK3-C-N1-UT-E)

This section includes hardware and connection information for the deprecated kit, Digi XBee 3 Cellular LTE-M/NB-IoT (part numbers XK3-C-A2-UT-U or XK3-C-N1-UT-E). The deprecated kit includes the XBIB-U-DEV development board.

Note Information about how to configure and manage the XBee on the XBIB-U-DEV development board can be found in this guide. The XBee configuration is the same for both kits and development boards.

To connect the hardware using the XBIB-U-DEV development board, you should perform all of the steps below in the order shown.

1. Identify the kit contents
2. Determine cellular service and acquire a SIM card
3. Connect the hardware
4. Install and upgrade XCTU
5. Update the device and cellular firmware using XCTU
6. Configure your module for cellular connectivity
7. Check for cellular registration and connection
8. Use one of the following methods to verify your cellular connection:
   - Connect to the Echo server
   - Connect to the ELIZA server
   - Connect to the Daytime server

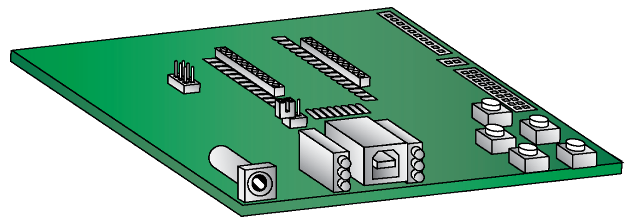
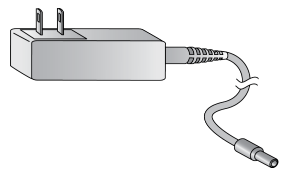
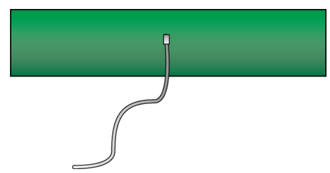
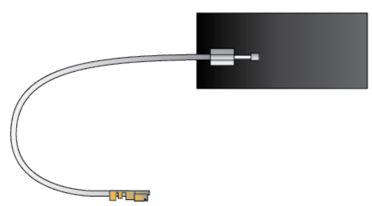
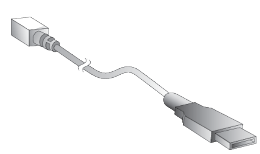
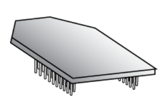
Optional steps

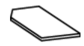
You can review the information in these steps for more XBee connection examples and examples of how to use MicroPython.

1. Review additional connection examples to help you learn how to use the device. See XBee connection examples.
2. Review introductory MicroPython examples. You can use MicroPython to enhance the intelligence of the XBee to enable you to do edge-computing by adding business logic in MicroPython, rather than using external components.
   - Example: hello world
   - Example: Turn on an LED

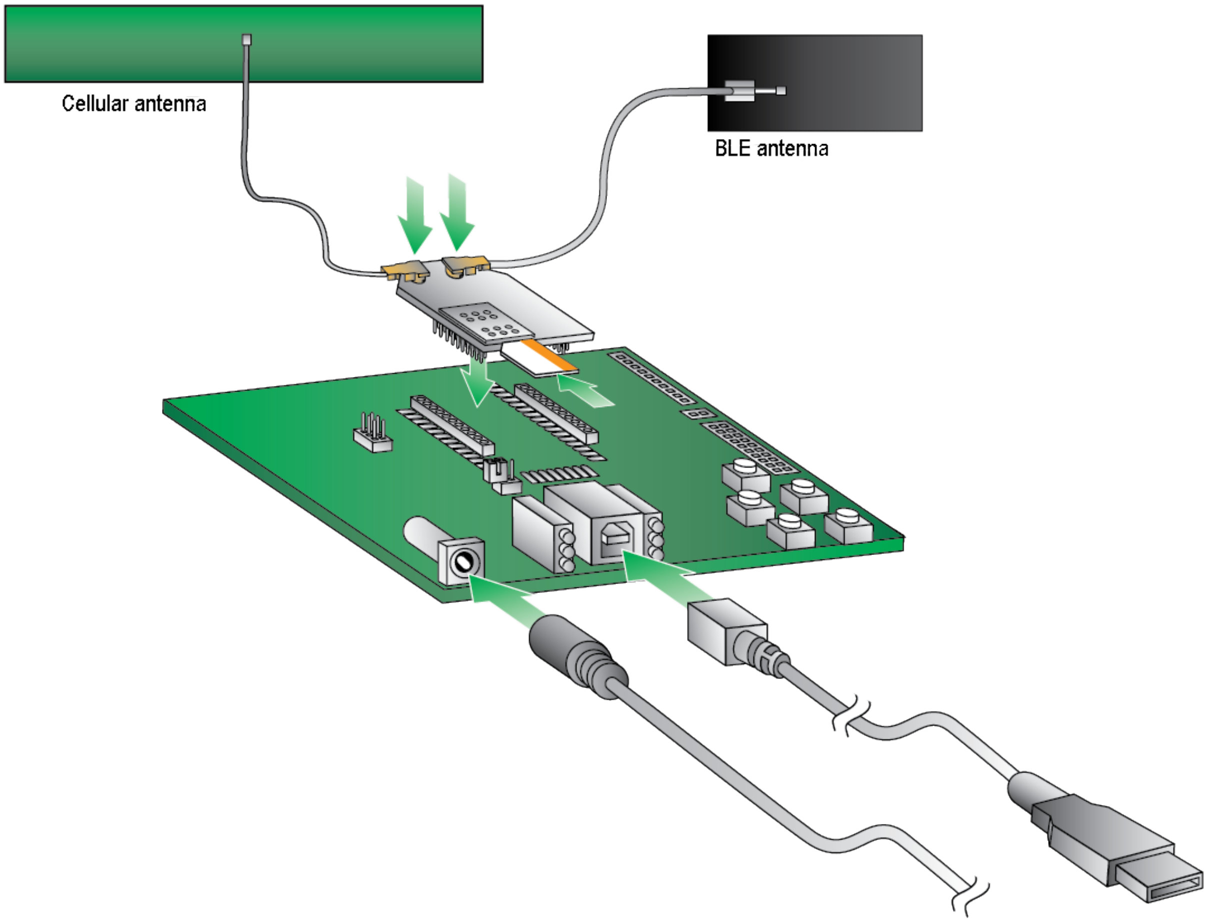
Identify the kit contents

The Developer's kit includes the following:

Item	Description
One XBIB-U-DEV board	
One 12 V power supply	
One cellular antenna with U.FL connector	
One Bluetooth Low Energy (BLE) antenna	
One USB cable	
One XBee Smart Modem **Note** The XBee Smart Modem comes attached to the board in ESD wrap.	

Item	Description
One SIM card **Note** NB-IoT kits (Digi product number XK3-C-N1-UT-E) do not include a SIM card. Contact your NB-IoT mobile carrier provider to obtain a SIM card and service. See Determine cellular service and acquire a SIM card.	

Connect the hardware



1. The XBee Smart Modem should already be plugged into the development board. For more information about development boards, see Development boards.
2. If a SIM card is included with the kit, the card is inserted into the XBee. If a SIM card is not included, install the SIM card into the XBee before attaching the XBee device to the board.

> **Note** Some kits do not include a SIM card. Contact your mobile carrier provider to obtain a SIM card and service. See Determine cellular service and acquire a SIM card.

> **WARNING!** Never insert or remove the SIM card while the device is powered!

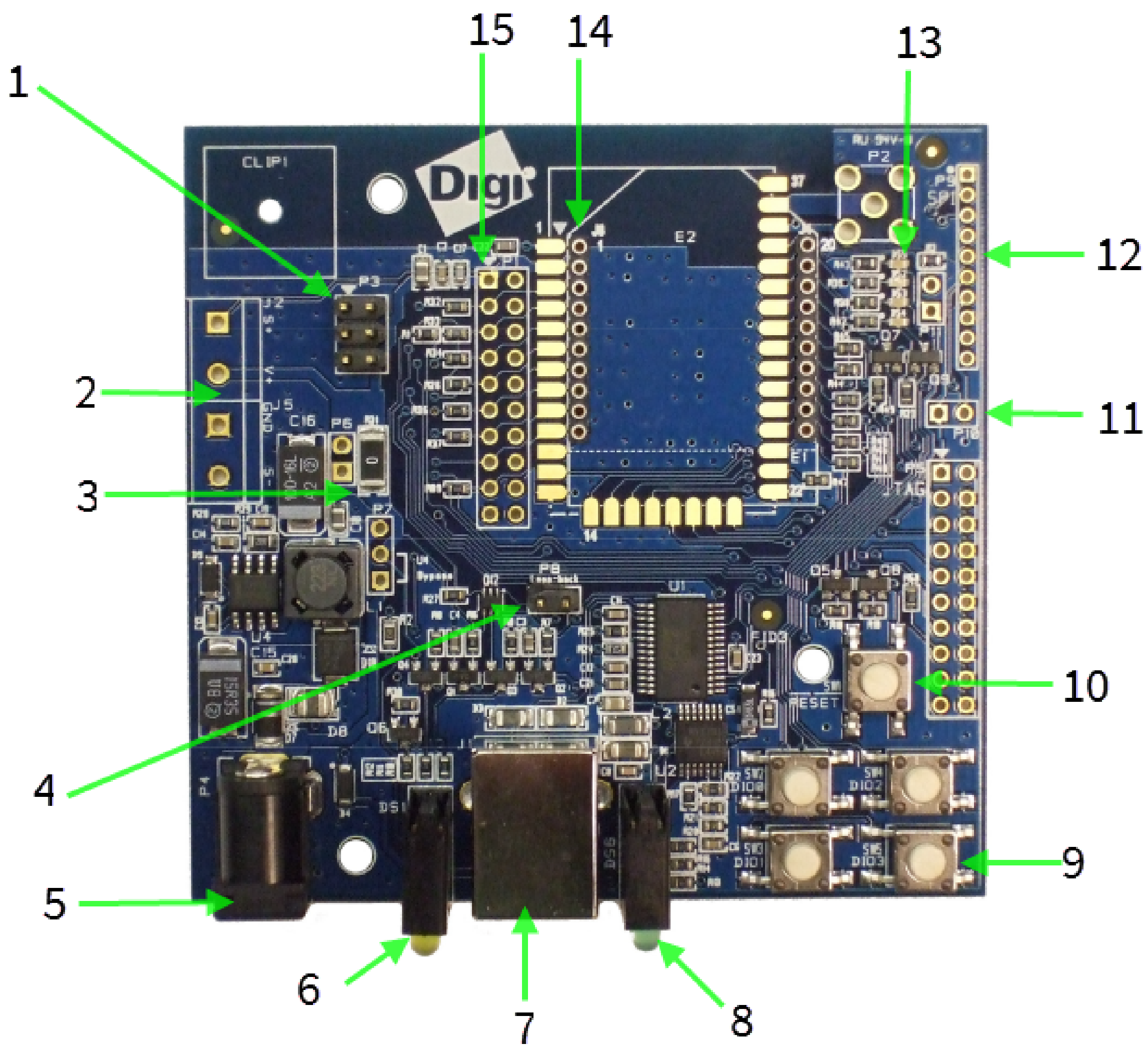
3. Attach the XBee device to the board.
4. Connect the antennas.
   a. Connect the cellular antenna.
   b. Connect the BLE antenna if you are using BLE functionality. If you are not, you do not have to connect the BLE antenna.

Note Align the U.FL connectors carefully, then firmly press straight down to seat the connector. You should hear a snap when the antenna attaches correctly. U.FL is fragile and is not designed for multiple insertions, so exercise caution when connecting or removing the antennas. We recommend using a U.FL removal tool.

5. Plug the 12 V power supply to the power jack on the development board. The LED indicator blinks when the board is powered.
6. Connect the USB cable from a PC to the USB port on the development board. The computer searches for a driver, which can take a few minutes to install. If you have trouble downloading the USB driver, see Other possible issues.

XBIB-U-DEV reference

This picture shows the XBee USB XBIB-U-DEV development board and the table that follows explains the call-outs in the picture.



Number	Item	Description
1	Programming header	Header used to program XBee programmable devices.
2	Self power module	Advanced users only—voids the warranty. Depopulate R31 to power the device using V+ and GND from J2 and J5. You can connect sense lines to S+ and S- for sensing power supplies. CAUTION: Voltage is not regulated. Applying the incorrect voltage can cause fire and serious injury.[1]
3	Current testing	Depopulating R31 allows a current probe to be inserted across P6 terminals. The current though P6/R31 powers the device only. Other supporting circuitry is powered by a different trace.
4	Loopback jumper	Populating P8 with a loopback jumper causes serial transmissions both from the device and from the USB to loopback.
5	DC barrel plug: 6-20 V	Greater than 500 mA loads require a DC supply for correct operation. Plug in the external power supply prior to the USB connector to ensure that proper USB communications are not interrupted.
6	LED indicator	Yellow: Modem sending serial/UART data to host. Green: Modem receiving serial/UART data from host. Red: Associate.
7	USB	Connects to your computer.
8	RSSI indicator	See RSSI PWM. On the XBIB-U, more lights are better.
9	User buttons	Connected to DIO lines for user implementation.
10	Reset button	Press the reset button to reset the device to the default configuration.
11	SPI power	Connect to the power board from 3.3 V.
12	SPI	Only used for surface-mount devices.
13	Indicator LEDs	DS5: ON/$\overline{\text{SLEEP}}$ DS2: DIO12, the LED illuminates when driven low. DS3: DIO11, the LED illuminates when driven low. DS4: DIO4, the LED illuminates when driven low.
14	Through-hole XBee sockets	Connects the Digi XBee 3® Cellular LTE-M/NB-IoT modem to the XBee USB development board.

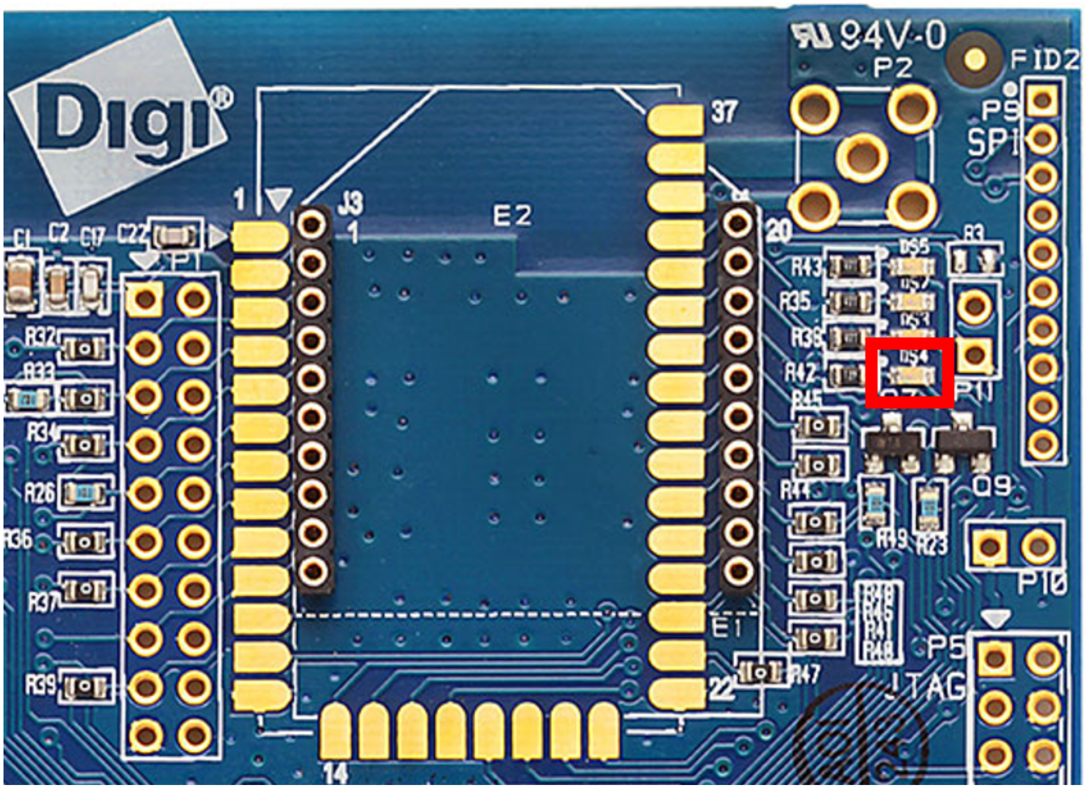
1Powering the board with J2 and J5 without R31 removed can cause shorts if the USB or barrel plug power are connected. Applying too high a voltage destroys electronic circuitry in the device and other board components and/or can cause injury.

Number	Item	Description
15	20-pin header	Maps to standard through-hole XBee pins. Male, Samtec header, part number: TSW-110-26-L-D. 2.54 mm / .100" pitch and row spacing.

Example: Turn on an LED

Note This example is only for kits that use the XBIB-U-DEV development board. For an example that uses the XBIB-CU-TH development board, see Example: Turn on an LED.

1. Note the **DS4** LED on the XBIB board. The following image highlights it in a red box. The LED is normally off.



2. At the MicroPython **>>>** prompt, type the commands below, pressing **Enter** after each one. After entering the last line of code, the LED illuminates. Anything after a **#** symbol is a comment, and you do not need to type it.

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.

```
import machine
from machine import Pin
led = Pin("D4", Pin.OUT, value=0)  # Makes a pin object set to output 0.
# One might expect 0 to mean OFF and 1 to mean ON, and this is normally the
case.
# But the LED we are turning on and off is setup as what is# known as
"active low".
# This means setting the pin to 0 allows current to flow through the LED and
then through the pin, to ground.
```

3. To turn it off, type the following and press **Enter**:

```
led.value(1)
```

You have successfully controlled an LED on the board using basic I/O.