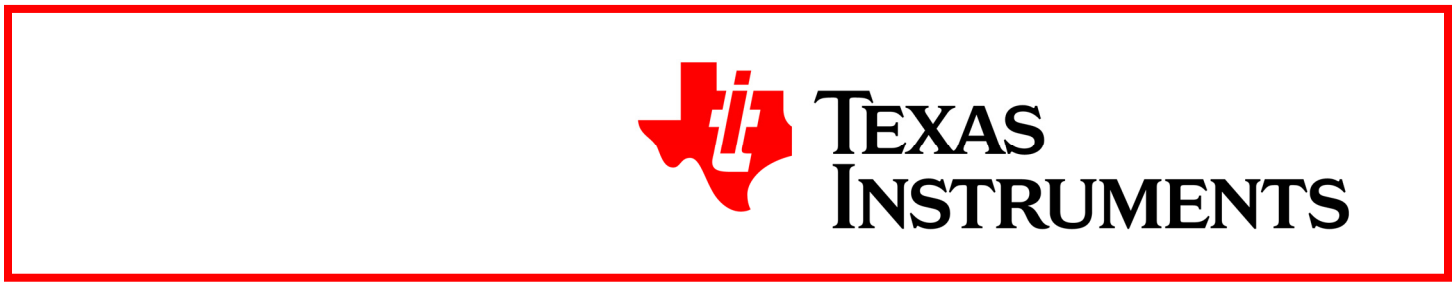
SimpliciTI



Developers Notes
Simple Modular RF Network

Texas Instruments, Inc.

www.ti.com/simpliciti

Texas Instruments
Low-Power RF

Table of Contents

Table of Tables

Table of Figures

i Revision History

Version	Description	Date
0.99	Pre-release draft	05/25/2007
1.00	Initial Release	07/02/2007
1.10	Revised to reflect SimpliciTI release 1.0	08/31/2007
1.20	Revised to reflect SimpliciTI release 1.0.4	02/01/2008
1.30	Revised to reflect SimpliciTI 1.0.6.	08/01/2008
1.40	Revised to reflect SimpliciTI 1.1.0.	03/24/2009
1.50	Added SimpliciTI timer calibration procedure.	10/27/2009
1.60	Included UART Driver and FHSS API information.	08/13/2011

ii References

[1] SimpliciTI Specification, Texas Instruments, 2007

[2] Application Note on SimpliciTI Frequency Agility, 2008

[3] SimpliciTI Application Programming Interface

[4] Application Note on SimpliciTI Security

[5] SimpliciTI Sample Application Guide

[6] Application Note on SimpliciTI Compatible UART Driver

[7] Application Note on Frequency Hopping

iii Definitions

- API Application Programming Interface.
- BSP Board Support Package
- CCA Clear Channel Assessment
- GPIO General Purpose Input Output
- ISR Interrupt Service Routine
- LED Light Emitting Diode
- LQI Link Quality Indication.

- LRU Least Recently Used
- MAC Medium Access Control.
- PHY Physical layer.
- RSSI Received Signal Strength Indicator
- FHSS Frequency Hopping Spread Spectrum

iv Nomenclature

Fixed pitch fonts **`such as this`** refer to program source code or source file names.

1. Introduction

This document provides information necessary to effectively use the SimpliciTI protocol support. Frequent references are made to source code files since source code is provided for this support.

There are some differences in the implementation depending on the specific radio being used. The references to the firmware supporting the protocol itself, however, are not hardware dependent.

2. Overview

SimpliciTI is a connection-based peer-to-peer protocol. It supports 2 basic topologies: strictly peer-to-peer and a star topology in which the star hub is a peer to every other device. The Access Point is used primarily for network management duties. It supports such features and functions as store-and-forward support for sleeping End Devices, management of network devices in terms of membership permissions, linking permissions, etc. The Access Point can also support End Device functionality, i.e., it can itself instantiate sensors or actuators in the network. In the star topology the Access Point acts as the hub of the network.

The protocol support is realized in a small number of API calls. These APIs support Customer application peer-to-peer messaging. The association between two applications, called linking, is done at run time. The linking process creates a connection based object through which the application peers can send messages. When a connection is established it is a bi-directional connection. There are provisions for a basic commissioning mechanism as well in which the connection context is populated directly with application layer calls.

3. Hardware configuration

3.1. MCU Interface

The CC1101/CC2500 radios specify a radio-MCU interface. In addition to the SPI communications the interface provides for up to two additional lines that when connected to MCU GPIO pins can be configured to generate interrupts. Though many of the reference designs provide for both of these to be connected, the implementation herein assumes only one of these (GDO0) is connected.

The CC2520 radio-MCU interface also specifies a GPIO line usable for interrupts and an SPI interface for command strobes and data transfer.

The SoC solutions depend on memory-mapped registers for data transfer and command strobes and they supply an interrupt vector location for presenting interrupts.

3.2. Radio configuration

The source file from which the initialization values for the radio are taken was generated by the SmartRF Studio7 tool from TI. The exported register configuration is included in the code distribution.

In addition in some cases additional register settings are added. These consist of setting registers that are not exported explicitly by the SmartRF Studio tool.

4. Architecture overview

4.1. Protocol layers

The protocol is in service to an application layer in which the main focus is peer-to-peer communication. The peers would typically be sensor-controller and actuator-controller objects. Direct sensor-actuator peers are supported as well. The protocol makes no distinctions here.

The goal of the protocol from the implementation perspective is to make it simple to link together various arbitrary peer applications.

A schematic of the layering is shown in the following Figure:

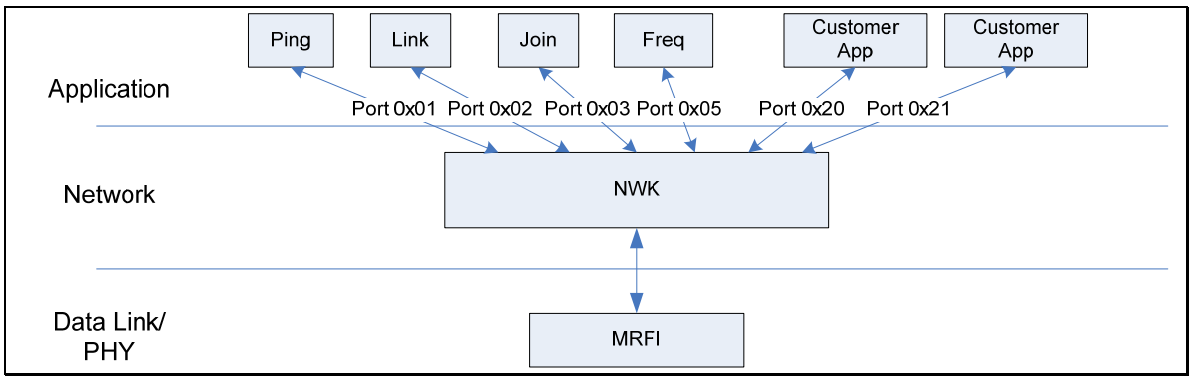


Figure 1: SimpliciTI logical layers

There is no formal PHY or Data Link (MAC/LLC) Layer. The data are received directly from the radio already framed so the radio performs these functions. The MRFI (Minimal RF Interface) layer encompasses whatever support is necessary to interact with the radio.

There is also an entity (not shown) called Board Support Package (BSP) to abstract the SPI interface from the NWK layer calls that interact with the radio. It is not intended to support a general hardware abstraction in service to the applications. Only those services (such as the SPI interface) that are in direct support of the NWK-radio interface are supported. As a convenience it also supports LEDs and button/switch peripherals attached to GPIO pins. But no other services are provided such as UART drivers, LCD drivers, or timer services.

The NWK layer manages the Rx and Tx queues and dispatches frames to their destination. The destination is always an application designated by a Port number. The NWK layer does no frame processing on behalf of applications.

The Ports are similar in spirit to the notion of a TCP/IP port. It is conceptually an extension of the address. The network frame overhead is stripped off and the remaining payload is dispatched to the application that lives on the designated Port.

The NWK layer applications are on “well known” ports. These all have values ≤ 0x1F. They are used by the NWK layer itself to manage the network. These ports are not intended to be accessed directly by Customer applications[1]. The NWK layer applications are not connection-based.

The Customer application Ports are assigned during the Link process by the NWK. As a result of a successful link transaction the application receives a handle called the Link IDs. The mapping from Link ID to address is done by the NWK. This is similar to the Sockets approach. The application has no responsibility with respect to assigning and maintaining the Port objects.

4.2. NWK applications

The NWK applications support network management. Except possibly for Ping these applications are not intended to be part of the Customer’s development environment. They are described here to provide a more comprehensive understanding of how SimpliciTI protocol supports the communication s environment.

Application	Port	Description
Ping	0x01	Just like the TCP/IP application. Echoes the received payload back to the sender. Direct addressing only.
Link	0x02	Used to associate two peers on different devices.
Join	0x03	Used when an Access Point is present to gain access to peers.
Security	0x04	Used to exchange security information.
Freq	0x05	Used to manage frequency migration to support frequency agility.
Mgmt	0x06	General use NWK management application. Used for example as the poll port.

Table 1: NWK applications

[1] The Ping application is an exception to this. This application is intended primarily for debugging, as it is awkward to use otherwise. The destination address must be known to use this Port (just like the IP Ping) and typically this isn’t known at the application layer.

4.3. Peer layer characteristics

There are essentially two SW peer layers in this architecture: NWK and Application. As seen in Figure 1 the Application layer is partitioned into two parts: NWK applications and Customer applications. Characteristics of each are described in Table 2:

Layer	Connection Type	Acknowledgment
NWK	Connectionless	No
NKW Applications	Connectionless	Yes[2]
Customer applications	Connected. The connections are bi-directional.	Optional depending on Customer's implementation

Table 2: Peer layer characteristics

For development purposes it is important to note that SimpliciTI in its basic form does not support acknowledgements. The main consequence of this is the need for applications themselves to provide support for the following:

- Segmentation and reassembly for messages larger than the maximum application payload
- Missing data (no NWK guaranteed delivery in the form of Transport layer)
- Redundant data (no NWK recognition of duplicate frames)

4.4. Message acknowledgements

With SimpliciTI Release 1.1.0 some support for acknowledgement is provided. When an application sends a message requesting an acknowledgment, a success (i.e., acknowledgement received) means only that the peer's NWK layer received the frame. It does not imply that the peer application itself received the frame.

This is not peer-to-peer acknowledgement and should not be use as sole determination of guaranteed delivery.

See [3] for details on the use of this feature.

5. Protocol overview

The functionality provided by the protocol is simply to provide support for connection-based peer-to-peer communications. The intent is to wrap the fundamental radio portion and remove that domain from the Customer's concern during development.

The functionality is realized in a simple set of API calls available to the Customer's application. The simplicity comes with the price of flexibility. The vision is that the use of this simple, small footprint protocol will be in scenarios that require only limited flexibility.

The following discussion summarizes the mechanisms that are implemented in support of the protocol.

[2] There are some exceptions to this. In general NWK application frames are acknowledged but some are not. In any case these exchanges are not visible to the Customer applications.

5.1. Topology

The protocol addresses only peer-to-peer topologies. With this protocol there is no formal routing mechanism supported. The Access Point (if present) supplies the support needed to manage the network. This support includes network access and network management functions such as frequency agility.

A star topology is realized by setting the hub to be a peer of each other device on the network.

5.2. SimpliciTI objects

SimpliciTI objects are SW objects. Three SimpliciTI objects are supported: End Devices, Access Points, and Range Extenders. Each is a logical construct so that multiple objects can be realized on a single hardware platform. For example, a platform that contains an Access Point can also support an End Device. However, most End Devices will likely occupy a hardware device either alone or with other End Devices.

5.2.1. End Device

These are the simplest devices. They are the locus of most of the sensors/actuators in the network. The End Devices host the application peers. A hardware platform hosting only End Devices may be battery powered.

5.2.2. Access Point

The Access Point, when present, can act as the star hub in the network. These are always-on devices. Only one Access Point per network is permitted. This can be enforced using constructs discussed below.

Access Points may coexist with End Devices on the same hardware platform. They can host peer applications that realize sensors or actuators, or both on the network.

Access Points run in promiscuous mode will receive all packets within range. In addition to infrastructure support Access Points will replay frames not destined for itself to help extend the range of End Devices.

5.2.3. Range Extender

The Range Extenders are intended to extend the radio range on a network. These are always-on devices. The main function is to repeat frames effectively extending the sphere of influence of the frame sender. Currently networks are limited to 4 Range Extenders.

Although not seen as a common use, Range Extenders may coexist with End Devices on the same hardware platform. They can host peer applications that realize sensors or actuators, or both on the network.

Range Extenders run in promiscuous mode will receive all packets within range.

5.3. Address namespace

A network addresses consist of two parts: a platform hardware address and an application address (Port).

As distributed the hardware address for each device is assigned at build time. A capability is provided to assign the address at run time as well. It must be unique for each device in the network. The management of the hardware address space is up to the Customer. There is no address resolution protocol.

Currently the hardware address space is that spanned by a four byte quantity treated as an array of unsigned characters.

The application address (Port) is either well-known and fixed or assigned at run time during the device Link procedure. It is not under Customer SW control.

5.4. Network discipline

In this Section will be a brief overview discussion of how peer-to-peer connections are made and how network member ship is controlled.

More information on the details of the frames involved in the connection processes can be found in [1].

5.4.1. Linking

Linking, supported by the SimpliciTI API, is the means by which an application peer-to-peer connection is established. Links are established in pairs. One application of the pair listens for a link message from the application with which it should be paired. Which application listens and which sends the link message is arbitrary, as the resulting connection is bi-directional.3 Typically a link session would be instigated by an end user action such as a button press or other physical intervention.

The link message contains a link token (currently a 4 byte object) which is used by the listener to validate the peer. There is a default link token set by the Customer at build time. Additional constraints can be added if the network has an Access Point. See Section 5.4.2.

Links are logical entities. A single platform may support multiple peer-to-peer links. These may be multiple links between the same two applications, multiple application pairs with single links, or any combination. The pairs may be on any two distinct platforms. SimpliciTI does not support links between two applications on the same platform.

The number of links on a device is limited only by RAM and the port address space.

5.4.2. Joining

The Join action is supported only when the network contains an Access Point. Joining is not supported by a specific SimpliciTI API but is a side effect of the initialization call. It is the process by which a platform gains access to the network from an Access Point.

[3] If one of the applications is on a Tx-only platform the assignment of roles becomes constrained.

Joining is the first action on the part of a device after initialization and before any other actions are taken.

The platform that wants to join a network sends a Join message as part of the SimpliciTI initialization. The message contains a Join token (currently a 4 byte object) which is set by the Customer at build time. The Join token can be used to ensure that two Access Points do not both respond to a new device trying to join a network. If the Join token matches the token expected by the Access Point the Access Point will reply with network information that the platform will require in order to interact properly on the network. Currently this information includes the Link token for the network (see Section 5.4.1).

The use of the unique Join token might be awkward if after-sale devices are added to the network. It would require the new device to somehow know the network's Join token at the point of installation. To deal with this scenario there is an additional capability to set the Join context in the Access Point to be either active or inactive. The context could be activated on the Access Point in the same way a link context is set (button press, etc.) and a default Join token could be used. By default the Join context is always active.

The act of joining a network does not provide anything other than infrastructure information such as the Link token to the joining device. The Access Point does not track joining devices except in the case of a polling device for which the Access Point must supply store-and-forward support (See Section 5.4.3).

5.4.3. Sleeping End Devices

Sleeping End Devices can take one of two approaches to getting messages. One type polls the Access Point to see if any messages are waiting. The second type listens for activity and if there is activity it stays awake and looks for frames destined for it.

The type of sleeping device is configured at build time.

5.4.3.1. Polling devices

If a sleeping End Device is configured as a polling device it is recognized as such by the Access Point when the device joins the network. At this time the Access Point reserves resources to support the device. All messages addressed to the sleeping device on a non-network Port are held by the Access Point. Broadcast messages are not held.

When the sleeping device awakens and does a receive call the call results in a polling message to the Access Point. This polling message specifies the Port being queried and is sent on the Management Port. The Access Point sends the oldest frame on the queried Port destined for the polling device. If none are being held the Access Point sends a frame with no payload to the queried port.

In the current implementation each Port must be polled separately and each Port must be polled until there are no more messages on that Port.

6. Application programming

This section presents information on the SimpliciTI implementation that will aid in the proper construction of a Customer application using the SimpliciTI capabilities.

6.1. Development environment

There are two development environments for SimpliciTI. First is IAR Embedded Workbench. this environment can be used for both the 8051 core SoC targets and the dual chip (MSP430 + radio) solutions. The Texas instruments Code Composer Essentials v. 3.1 (CCE) IDE also supports SimpliciTI for the dual chip (MSP430 + radio) targets.

Except for IDE-dependent files (such as the linker script which is the default for the target) there are no known IDE-dependent constructs or keywords used in the source code.

6.2. Hardware abstraction

Only a bare minimum hardware abstraction is provided with the distribution. This abstraction (BSP) supports the radio interface. It also includes an abstraction to support up to 8 LEDs and 8 switches on the target board.

There is no support for UARTs, timers, LCDs or other potential peripherals or on-chip resources. These implementations are left completely to the Customer.

6.3. MCU Resources

Other than flash for code space and constants, and RAM, the SimpliciTI implementation uses no other resources. It does not currently depend on dynamic memory allocation so it uses no heap memory. All memory allocations are either static or are represented as automatics and therefore use the stack. The const memory type is used when possible to save RAM. The implementation does optionally use one timer resource though the least intrusive timer resource on the MCU is chosen when the hardware timer resource is sued.

The run-time context is not stored in any persistent memory on either the MCU or the radio. With some minor exceptions that are handled in the code, the initial radio targets do not lose run-time context when in sleep mode. Most of the SRAM is maintained and those critical values that are not maintained are restored.

6.4. Threading model

In general the SimpliciTI API is designed to run in conjunction with the threading model of the Customer application. The operation of the protocol is not constructed as an independent task requiring its own context. Since the code runs in the existing threading context the implementation requires no access to a scheduler or any other OS construct.

The SimpliciTI API should not be considered thread-safe or re-entrant.

SimpliciTI provides a general callback capability for received frames. This callback runs in the receive ISR thread so efficient use of the callback is encouraged.

6.4.1. Peer-to-peer I/O

After initialization the peer applications are linked using a pair of sequenced calls in the API. The result of these calls on each end is a Link ID. This is a handle, like a socket

except that the binding is done automatically. The Link ID is used to reference the connection subsequently. After the linking step the may read from and write to the peer by referencing the Link ID Basically the API consists of a read (Receive) call and a write (Send) call.

6.4.1.1. Input/receive

On devices that do not sleep the receipt of a frame by the radio triggers an interrupt to the MCU. The frame is read out of the radio Rx FIFO and stored in user space in the MCU in a queue of received frames. If the input frame queue is full when a frame is received the oldest frame in the queue is dropped. The interrupt thread is then released. This is a relatively efficient process.

When the application level read is done on the specified Link ID the input frame queue is examined for any frames waiting for that Link ID. If a frame is waiting the application payload is returned to the caller. Otherwise the caller receives an indication that there are no data. In essence a read is a poll of the input frame queue for any frames waiting for the designated Link ID. Payloads are returned in FIFO order.

If the device is a sleeping/polling device the application layer read kicks off a poll query to the Access Point. This is invisible to the caller. The thread is then held and waits for the reply from the Access Point. If the Access point forwards a frame it is then passed back to the caller. If the Access point sends a frame with no payload it is interpreted as a simple acknowledgement and the return to the caller looks like a poll that returns with no payload. Because this scenario holds the thread until the reply from the Access Point it is less efficient than the non-sleeping case that simply examines the input queue for data.

6.4.1.2. Output/send

The sending scenario is implemented as a synchronous call that does not return until the frame is transmitted by the radio. This design prevents unintentional termination a transmit sequence by removing radio power before the frame is sent.

If the Tx thread cannot get access to the radio channel to transmit the frame the caller receives a return code accordingly and can retry later. There is some robustness in the network layer, as there is some degree of recovery attempted if access to the channel cannot be obtained immediately. But this is minimal. It is left to the Customer application to decide how to deal with this scenario since the SimpliciTI NWK does not support guaranteed delivery (see Table 2). Only the application knows how important it is to send the frame. The power consumption vs. communications reliability tradeoff is left to the Customer application.

6.4.2. NWK application threads

Since the NWK application threads (those that service the NWK application ports) are not part of the Customer application context and use no scheduling services they must be handled differently.

The Join, Link, and Mgmt polling frames each require a response from the target device. At the application layer this makes them acknowledged frames. To ensure that the application has the opportunity to complete the task (i.e., joining or linking) the application will wait for the acknowledgment to arrive. Since callbacks are not used and

there is no scheduler involved the sending application holds the calling thread until the reply is received. So, for joining, linking, and polling the sending device may take a “long time” (milliseconds) before completing.

The receiving device for these same applications also handles everything in one thread. In this case, the receive and acknowledge (Tx) are done in the same thread. In this case, the thread is an ISR thread so it is possible for interrupts to be disabled for a “long time” (milliseconds) when handling such a frame.

Note that while these threads can be “long” they do not occur very often. Joining occurs only at startup. Linking typically occurs at startup, though can occur anytime. Polling may occur more often but is still relatively infrequent.

6.5. Object model

SimpliciTI has no formal object model. There are no profiles or other abstractions that define peer application characteristics. The focus of the protocol is to set up peer-to-peer connections and support messaging over air to and from each peer. The object model is created by the Customer, as it is implicit in the application layer protocols set up between peers.

6.6. Sample SimpliciTI transactions

The following is a sample SimpliciTI session. It is intended to convey a general sense of how SimpliciTI peers are linked. The presence of an Access Point is optional. If it is not present the default Link tokens must match in the two End Devices or the listener will reject the Link message by simply not responding.

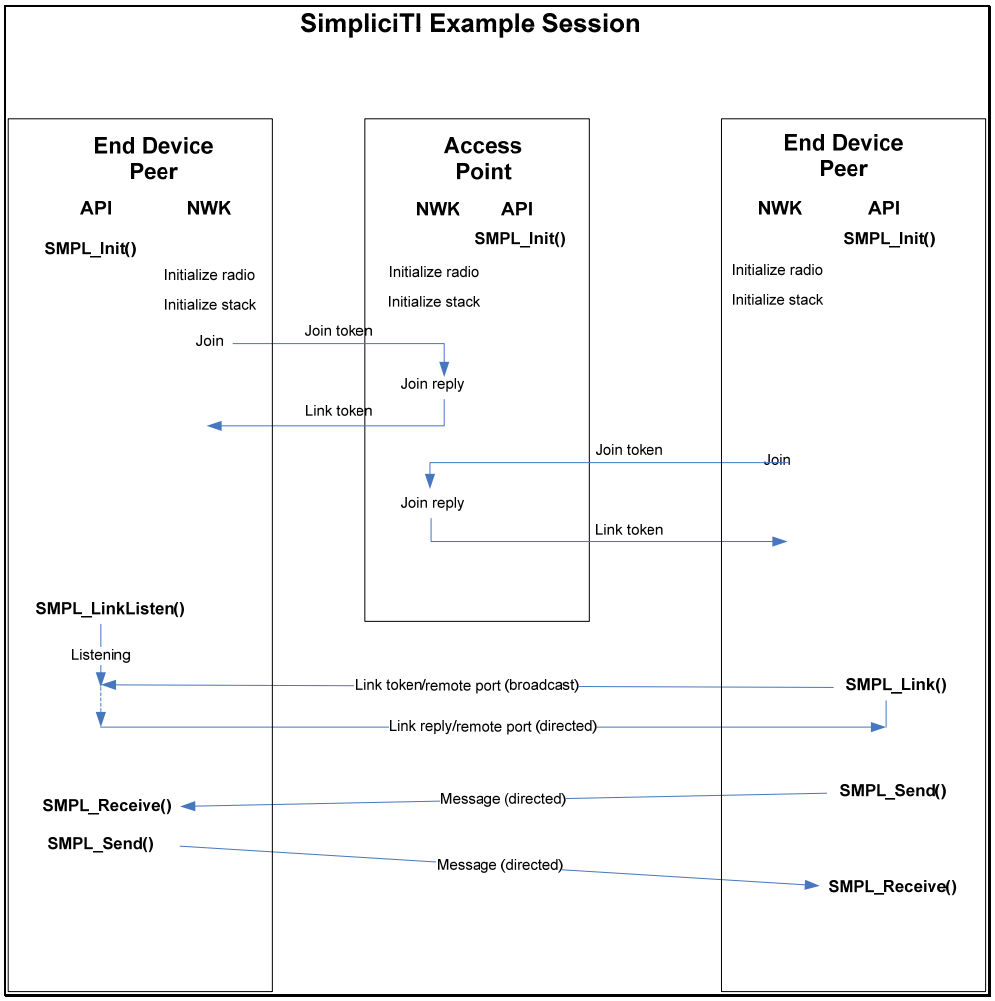


Figure 2: Sample SimpliciTI peer-to-peer session

7. Network access control

There are various means of controlling access to specific network. There is likely a need to guard against rogue devices whether their presence is purposeful or accidental

There are a number of mechanisms built into SimpliciTI to allow various forms of access control. Two of these take advantage of the added control that is possible when an Access Point is part of the configuration. Each of these is discussed below.

7.1. Join token

If an Access Point is part of the configuration then it supports the Join NWK application. The main purpose of this application is to provide hints to the remaining devices joining the network as to parameters of that network. This would include Link token and encryption key, for example. The idea is that without properly joining the network (i.e., knowing the Link token and key) a rogue device could not interact on the network. Devices need to know the Join token for the specific network to find out the other network parameters.

Customers can configure networks to have different Join tokens, or perhaps, different tokens for different products or types of applications. Join response could be modified as well. For example, the Access Point could assign different Link tokens to different network devices. The Join NWK application payload could be modified to provide more or different information than the default.

7.2. Access Point Join context

It might be necessary to actively invoke the Join context on an Access Point. For example if two unrelated Access Points are in proximity to a joining device (e.g., two neighboring houses) additional control might be required to constrain which network it joins.

By default Access Points will always accept a Join frame with the correct Join token. As an added control on access a Customer application can set the Join context of an Access Point. That is, it can be set to reject Join frames unless they are set to actively permit such frames. This is done using the SMPL_Ioctl() API.

7.3. Link token

If an Access Point is part of the configuration then it supports the Join NWK application. The main purpose of this application is to provide hints to the remaining devices joining the network as to parameters of that network. One of these network parameters is the Link token. The idea is that for each network the Link token would be unique, or at least controlled. Two devices subsequently linking must use the correct token or they cannot link.

If there is no Access Point a link token is still required for establishing a connection between two peers. In this case the link transaction is governed by the default link token, the value supplied at build time.

Using the Link token scheme provides an alternative to explicit physical binding in which the physical co-location of two devices provides the security needed to exclude unwanted devices.

7.4. Encryption

Encryption can be used to augment network access control. Encryption is implemented as of SimpliciTI Release 1.1.0. See Reference [4] on SimpliciTI Security implementation.

8. SimpliciTI SW Timer Calibration

SimpliciTI design allows the user to select a software timer mode so that the hardware timer can be used for other purposes. This is done by modifying the SW_TIMER directive in the smpl_nwk_config.dat file. For IAR projects, remove the x from -DxSW_TIMER to enable the SW timer mode. For CCS projects, insert a # before --define=SW_TIMER to disable the SW timer mode. Note that some configurations come “out of the box” with SW Timer enabled, others with SW Timer disabled.

The SimpliciTI SW timer can be calibrated for improved accuracy. The SimpliciTI SW timer can be calibrated with the following iterative procedure:

[1] Find APP_USEC_VALUE in mrfi_radio.c:

[2] #define APP_USEC_VALUE 496

[3] Follow instructions in reference [5] to setup the “Two-device peer-to-peer” example.

[4] When running the “Two-device peer-to-peer” examples, the talker sends messages at intervals, repeating the pattern of 1 second, 2 seconds, 3 seconds, and 4 seconds.

[5] Use a SimpliciTI sniffer to measure the actual delay interval patterns (see Figure 3).

[6] Adjust APP_USEC_VALUE as needed to obtain message intervals on even one-second boundaries.

[7] Repeat steps 3, 4 and 5 until the desired delay interval pattern is observed from the sniffer.

Figure 3 shows a sample sniffer screen capture. The talker device (Source Address 7A 56 34 12) is sending data in roughly 1S, 2S, 3S, 4S, 1S, 2S, etc. repeated delay patterns.

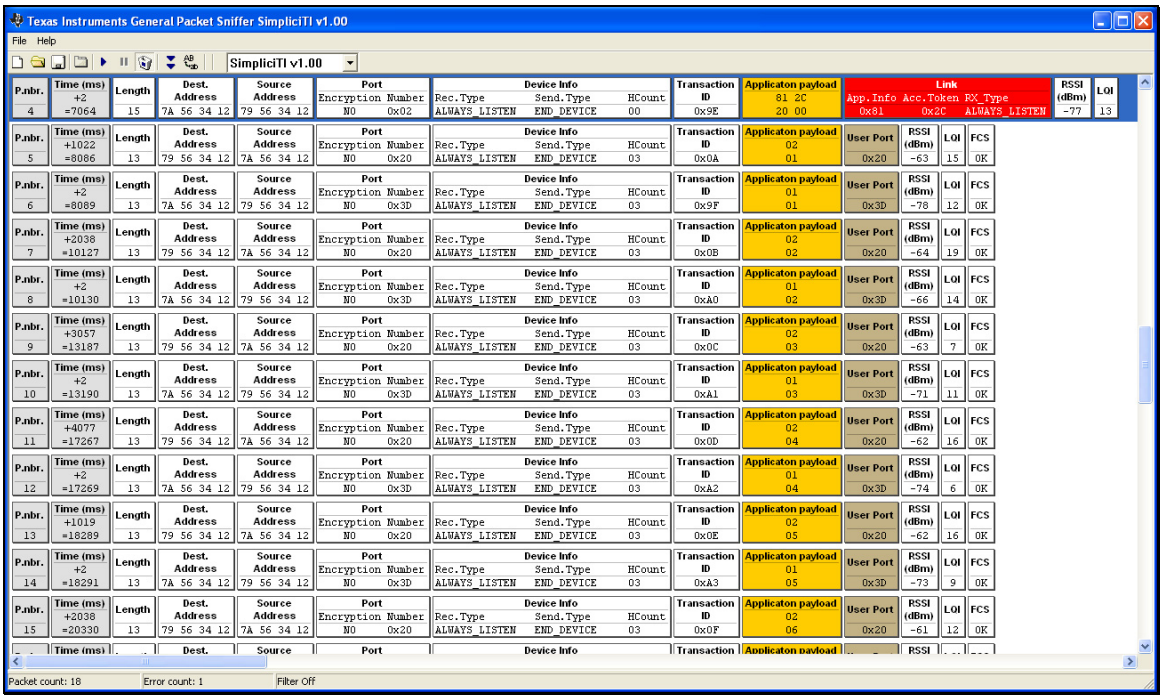


Figure 3: SimpliciTI two-device peer-to-peer sniffer capture

9. System configuration

Reference [1] summarizes the build-time configurable values needed by SimpliciTI. These are repeated here with additional discussion

There are two configuration file types used when building SimpliciTI devices. One file applies to the network in general and supplies macro values for all devices built for that particular network. The other file is device –specific. Each file type is discussed below.

9.1. General network configuration

The file smpl_nwk_config.dat contains the following network-wide macro definitions:

Macro	Default value	Comments
MAX_HOPS	3	Frames replayed only 3 times maximum
MAX_HOPS_FROM_AP	1	Maximum distance from an AP. Used by poll and poll response to limit network replay traffic.
MAX_APP_PAYLOAD	10	Absolute maximum is based on size of Rx and Tx FIFOs (50 for CC2500/CC1100 class including SoCs and 111 for IEEE radios CC2430/CC2520).
DEFAULT_JOIN_TOKEN	0x01020304	Change this to provide some access security
DEFAULT_LINK_TOKEN	0x05060708	Should be changed by APs on AP networks. Customer should modify nwk_join.c:generateLinkToken()
FREQUENCY_AGILITY	Not defined	When defined enabled support for Frequency Agility. Otherwise only the first entry in the channel table is used.
APP_AUTO_ACK	Not defined	Adds support for application level auto acknowledgment. Requires Extended API
EXTENDED_API	Not defined	Enables SMPL_Unlink(), SMPL_Ping() and SMPL_Commission().
NVOBJECT_SUPPORT	Not defined	Adds support for gerting and setting connection context for saving across resets.
SMPL_SECURITY	Not defined	Enables security infrastructure

Table 3: General network configuration macros

9.2. Device specific configuration

Macro	Default value Comments
NUM_CONNECTIONS	4 Number of connections supported. Each connection supports bi-directional communication. Access Points and Range Extenders can set this to 0 if they do not host End Devices.
SIZE_INFRAME_Q	2 Two is probably enough for an End Device. Little bit larger for REs and APs. AP needs larger input frame queue if it is supporting store-and-forward clients because the forwarded messages are held here.
SIZE_OUTFRAME_Q	2 The output frame queue can be small since Tx is done synchronously. If an Access Point device is also hosting an End Device that sends to a sleeping peer the output queue should be larger because the waiting frames in this case are held here. Actually 1 is probably enough otherwise.
THIS_DEVICE_ADDRESS	{0x78,0x56,0x34,0x12} This device's address. The first byte is used as a filter on the CC1100/CC2500 radios so THE FIRST BYTE MUST NOT BE either 0x00 or 0xFF. Also, for these radios on End Devices the first byte should be the least significant byte so the filtering is maximally effective. Otherwise the frame has to be processed by the MCU before it is recognized as not intended for the device. APs and REs run in promiscuous mode so the filtering is not done. This macro initializes a static const array of unsigned characters of length NET_ADDR_SIZE (found in nwk_types.h).
RANGE_EXTENDER	Must be defined for devices of this type Device type declaration for Range Extenders
END_DEVICE	Must be defined for devices of this type Device type declaration for End Devices
RX_POLLS	Not defined Define RX_POLLS of the device is a polling End Device.
ACCESS_POINT	Must be defined for devices of this type Device type declaration for Access Points
NUM_STORE_AND_FWD_CLIENTS	3 Number of store-and-forward clients supported.
AP_IS_DATA_HUB	Not defined If this macro is defined the AP will be notified through the callback each time a device joins. The AP should be running an application that listens for a link message on receipt of this notification. The ED joining must link immediately after it receives the Join reply.

Table 4: Device-specific configuration macros

10. General information and caveats

The following are in no particular order.

1. **CC25xx/CC11xx radios only**: SimpliciTI uses exported register settings from the SmartRF Studio configuration tool. This tool exports register values for both over-the-air RF settings, as well values for control register settings. SimpliciTI uses the values for over-the-air RF settings but must ignore most of the control register settings. The control settings are affect functionality so they must not be overridden. The following table lists registers that are directly controlled by software:

Register	Comments
`IOCFG0`	GPIO configuration. Controlled exclusively by software.
`IOCFG2`	GPIO configuration. Controlled exclusively by software.
`MCSM0`	State machine configuration. The value of PO_TIMEOUT is extracted from SmartRF Studio output, all other fields are controlled by software.
`MCSM1`	State machine configuration. Controlled exclusively by software.
`PKTLEN`	Packet length. Controlled exclusively by software.
`PKTCTRL0`	Packet control register. The value of WHITE_DATA is extracted from SmartRF Studio output, all other fields are controlled by software.
`PATABLE0`	SmartRF Studio does not currently export this value. If a future revision does export this value, it will be used instead of the built-in software default.
`CHANNR`	Channel number. The value exported by SmartRF Studio is used. However, this can be overridden by defining MRFI_CHAN at the project level as equal to the desired channel number.

Table 5: CC1100/CC2500 register setting exceptions

2. See [2] for details on the Frequency Agility feature. See Section 3.1 in that document for details about how to modify the channels in use for this feature.
3. The SMPL_LinkListen() call blocks for a relatively long period of time on the assumption that it is executed in temporal contiguity with the SMPL_Link() call on the peer. The blocking time may be modified by changing the appropriate commented macro definitions in the file nwk_api.c.
4. Arrival order for received frames is maintained. If the frame queue is filled and a new frame arrives the oldest frame is discarded to make room.
5. When an application does a SMPL_Receive() it will receive the payload from the oldest frame available for the specified link ID.
6. When frames are forwarded to a store-and-forward client they will be forwarded in the order received.
7. Broadcast frames and NWK application frames are not saved on behalf of store-and-forward clients.

8. If the Access Point itself hosts an application that sends to a polling device, these frames will be sent after all frames forwarded from other devices instead of in the order in which they were made available.
9. Access Points do not remember the addresses of joined devices. This means that Access Points will replay frames it sees regardless of source. (The same is true for Range Extenders.) It is up to the peer application to ignore rogue messages. The use of the Join and Link tokens can mitigate interference from non-member devices by helping to prevent connections to unauthorized devices but it is no guarantee.
10. Frames to user applications are partially validated at the NWK layer by matching the destination port and source address to connection table entries. The received frame must pass this assessment or it is discarded. Otherwise there is no way to remove them from the input frame queue since there will be no application trying to retrieve these frames.