SimpliciTI: Simple Modular RF Network Specification

Author: Larry Friedman

Texas Instruments, Inc.
San Diego, California USA

Version	Description	Date
0.99	Initial release to BDSNet Team	04/25/2007
1.00	Version for wider release with changes based on reviews and early implementation.	05/09/2007
1.01	• Modifications to some network application frame formats • Join API doesn't exist anymore but was included in pseudo-code example. • Add poll to Mgmt port messages.	05/21/2007
1.02	• Change references from SMRFNet to SimpliciTI	06/12/2007
1.03	• Update to reflect addition of callback support.	07/02/2007
1.04	• Add number-of-connections byte to Join frame to support AP data hub scenario • Enhance description of callback in support of AP data hub scenario	08/01/2007
1.05	• Fix error in frame content Figure for client side Join frame	01/02/2007
1.06	• Frame drawing more generic • Add transaction ID to NWK application frames • Poll frame needs SimpliciTI address in addition to port • Frequency application frames defined • Ping application payload now fixed length. • SMPL_LinkListen() now a timed wait.	04/04/2008
1.07	• Add Unlink support	08/08/2008
1.08	• Changes to frame format for auto acknowledgement • Extended API • Added frame objects for Link session (Security)	02/27/2009
1.09	• Section 4.11, update the frequencies that SimpliciTI supports.	03/24/2009

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

This document presents the specification for a simplified RF network solution. This solution is intended to support quick time-to-market for customers wanting a wireless solution of low power, low cost and low data rate networks without needing to know the details of the wireless network support.

This document specifies a simplified approach to small low data rate wireless network solution

2. Abbreviations and terminology

AP	Access Point: one of the 3 SimpliciTI device types.
API	Application Programming Interface
CCA	Clear Channel Assessment; part of listen-before-talk discipline
Customer	The target of this solution: the entity that will use SimpliciTI in their products.
ED	End Device: one of the 3 SimpliciTI device types.
End User	The Customer's customer
LBT	Listen-before-talk: the means used to arbitrate use of the radio spectrum in the network
LLC	Logical Link Control
MAC	Medium Access Control
MCU	Microcontroller Unit
RAM	Random Access Memory
RE	Range Extender: one of the 3 SimpliciTI device types.
SoC	System-0n-Chip
UI	User Interface

3. Sample Applications

The following are a few sample applications that exemplify the potential market for devices supported by SimpliciTI.

3.1 Alarms and security

These applications cover such devices as glass-break sensors, occupancy sensors, door locks, carbon monoxide and light sensors, etc.

3.2 Smoke detectors

Similar to alarm applications, the smoke detector application is prevalent enough so that it deserves its own category. This application can include cascaded smoke detectors in which the activation of a single device propagates to the activation of all devices co-located to efficiently spread an alarm.

3.3 Automatic Meter Reading (AMR)

Some AMR environments are appropriate for SimpliciTI implementations. In particular, this would include meters that are not powered such as gas or water meters.

3.4 Home automation

This can include device control such as garage doors, appliances, environmental devices, etc.

4. Requirements

4.1 Low Power

SimpliciTI is intended to support low powered devices when the devices are not mains-powered. These devices will typically be battery powered low data rate devices arranged in a simple, limited topology.

R1. Power management of the MCU shall be accomplished at the application level using native MCU resources.

R2. Power management of the radio shall be available to the application via function calls (see R7).

4.2 Low Cost

There are two configurations to be supported: the radio-MCU solution and a SoC solution that integrates MCU and radio.

R3. To keep the cost low the target MCU MSP430 core shall be 8K flash and 512 bytes RAM or 4K flash and 256 bytes of RAM depending on device type (see R10).

R4. The target SoC (8051 core) target shall be less than 16K flash and 1K RAM.

4.3 Simple Implementation and deployment

The SimpliciTI-based network should be easy to develop and easy to deploy. The simplified development environment should have a simple API that enforces details of the RF communication without complex configuration by developers.

The simplified RF environment should also make it easy for customers to develop applications that make the solution easily implemented in the field by end users.

R5. Radio and network configuration shall be encapsulated and hidden from the application layer and shall be driven by build-time configuration possibly assisted by a tool such as SmartRF Studio.

R6. The API set for messaging in this environment shall be of the open/read/write/close variety.

R7. There shall be access run-time configurability using an **`ioctl()`**-like method.

R8. Linking application peers shall be of a **`link()/listenLink()`** variety.

4.4 Target radio configurations

R9. The order for releases of code for target radios for this project shall be:

1. CC1100/CC2500 (Sub 1 GHz/2.4 GHz radio with MSP430)
2. CC1110/CC2510 (Sub 1 GHz/2.4 GHz radio in 8051 core SoC)
3. CC2430/CC2420 (DSSS radio with and without 8051 core SoC)

4.5 Limited network device types

The devices described are logical device types. They may or may not map to unique physical devices. A single physical device may function as more than one logical device.

R10. Only 3 device types shall be defined. There shall be an End Device type (mandatory), an Access Point device type (optional), and a Range extender device type (optional).

4.5.1 Access Point

R11. An Access Point (AP) shall be always on and presumed to be mains powered possibly with battery backup. Only 1 AP per network is permitted.

R12. APs shall have the following functions not all of which may apply on a given network:

1. Network address management
2. Store-and-forward messages on behalf of sleeping Rx devices

R13. An AP may contain sensor/actuator (End Device) functionality.

R14. An AP may contain Range Extender functionality.

4.5.2 Range Extender

R15. The Range Extender (RE) shall be always-on and presumed to be mains powered possibly with battery backup. The device retransmits each unique frame it receives.

R16. A Range Extender may contain sensor/actuator (End Device) functionality.

4.5.3 End Device

R17. The End Device (ED) realizes the application layer functionality.

R18. An End Device may or may not be always on.

R19. End Devices may be RxTx devices or they may be Tx-only devices.

R20. Unless configured externally, Tx-only devices:

1. Transmit on a preset single or sequence of frequencies or channels.
2. Use a preset encryption key to encrypt and decrypt messages.

Tx-only devices may have switches, buttons, or other UI opportunities for simple network configuration.

4.6 Topology

R21. There shall be two logical messaging configurations: star and direct peer-to-peer (device to device).

4.6.1 Star

When an AP provides store-and-forward support for sleeping Rx devices the AP acts like a hub in a Star network topology. If all Rx devices are always on the AP provides no store-and-forward support but can provide network management support and Range Extender functionality.

4.6.2 Peer-to-peer

Rx devices that are always on will receive frames directly from the source device, possibly through a Range Extender. These are considered peer-to-peer relationships. It is possible that the network has no AP in which case the topology is entirely peer-to-peer. Even if all messages are broadcast, from the application perspective messages are coming from a peer.

4.7 Routing

R22. Explicit routing shall not be supported.

R23. Rx devices that are not always on may receive data by polling the AP possibly through a Range Extender. The variety of low-power modes offered by the combination of MCU and radio may support interrupt-driven schemes as well.

R24. Rx devices that are always on shall receive data directly from the source possibly through a Range Extender.

4.8 Security

4.8.1 Encryption

R25. Where hardware encryption is supported it shall be available for use by the customer. If hardware encryption is not available a default software encryption solution shall be available to the customer.

R26. Encryption key distribution shall be part of the network management support optionally provided by the AP. If there is no AP or a network member is a Tx-only device key settings will be accomplished using a default build-time scheme or possibly be using external settings imposed by the end user.

R27. The encryption solution shall use symmetric keys.

4.8.2 Rogue devices

R28. Rogue devices, malicious or not, shall be prevented from disrupting a network. There shall be defense against such interference as replaying of frames.

4.9 Medium access

R29. Access by a device to transmit will be managed by listen-before-talk procedure. The procedure shall follow the European (ETSI) specification for this algorithm.

4.10 Frequency agility

Frequency agility is intended to support robustness by providing a means to change frequency when a specific frequency is noisy or otherwise ineffective. It is realized in the form of channel migration. Channel migration not intended to be used also to spread energy over the spectrum to comply with FCC Part 15 requirements. It is not intended that frequencies change in any rapid manner, for example, packet-to-packet.

R30. The network shall support migrating to alternate frequencies if an existing frequency offers too much radio interference.

4.11 Radio parameters

R31. Each network shall support up to 250 kbps.

R32. The following frequencies shall be supported: 480, 868, 915, 955 MHz (sub-1 GHz) and 2.4 GHz.

4.12 Range extension without transmit loops

R33. Frame content shall contain hints that prevent looping.

R34. Range extension shall be limited to 4 hops from the message source to the message destination.

R35. The number of Range Extenders shall be limited to 4 (four) in a network.

R36. A Range Extender will not resend frames that it receives from another Range Extender

4.13 Battery-only networks

R37. Battery-only networks shall be supported. This implies that there will be no Range Extenders and no AP in such a network.

4.14 Interoperability

R38. Interoperability among versions of SimpliciTI with different feature sets shall not be required. Interoperability may be supported as a side effect of released versions but it is not required and therefore not guaranteed.

5. Overview of SimpliciTI

SimpliciTI is intended to support customer development of wireless end user devices in environment in which the network support is simple and the customer desires a simple means to do messaging over air.

The protocol is oriented around application peer-to-peer messaging. In most cases the peers are linked together explicitly. However, SimpliciTI will support scenarios in which explicit linking between pairs of devices is neither needed nor desired. An example if this is the smoke alarm scenario in which there are multiple sensors/alarms and the intent is to propagate the alarm throughout the network.

In this case an activated alarm would send a broadcast message to any and all nearby alarms so that the alarm signal could be propagated throughout the network of alarms. Since explicit linking has not occurred in this scenario the Customer must take care to choose the default access token so that interference from non-network sources is not likely.

From the Customer's development perspective the API provides the means to initialize the network, link devices, and send messages.

5.1 Module components

The basic stack will support simple RF messaging with no additional features. As described in the figure below there are modules (features/functions) that can be added to the basic stack in various combinations.

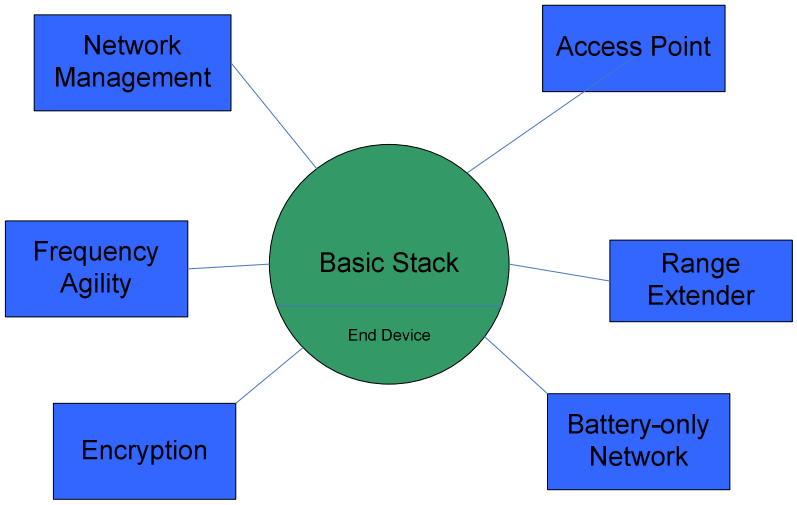


Figure 1: SimpliciTI Module components

A brief description of each component follows.

5.1.1 Basic stack

5.1.2 Encryption

The encryption choices are currently hardware or software. The CC12100/CC2500 radios do not have native support for hardware encryption so on platforms using these radios the encryption will be in software.

When encryption is enabled all fields except the address and encryption context fields are encrypted. Though this means that Range Extenders need to decrypt the frame to know whether to repeat it this will prevent rogue devices from storming the network with bogus frames leveraging off the Range Extenders.

5.1.3 Frequency agility

This capability allows the network to migrate to a different frequency if the existing frequency is noisy or otherwise compromised. It will be driven by a frequency table that is populated at build time.

Devices that can receive packets can detect that they are on the incorrect frequency by not receiving an acknowledgment after sending and resending a frame. The sender then steps through the frequency table until the acknowledgment is received. This is the scenario encountered by new devices trying to join a network and sleeping devices that awaken after a migration has occurred.

Changing frequency may also be actively imposed. In networks with an AP the AP will announce the change with a broadcast message. In networks without an AP there may be an external hardware announcement such as a switch or button to signal each device.

Tx-only devices must always send and resend on all frequencies in the table.

Detection of the need to change frequencies is not within the scope of this specification.

5.1.4 Network management

This is the domain of the AP and consists of functions such as store-and-forward functionality for sleeping devices, encryption key management, and frequency agility management.

5.1.5 Access Point

Access points can support End Devices.

Access Points will repeat frames but they set the Access Point Sender Type (see Section 6.5.5). When replaying a frame the hop count is decremented.

Access Points realize Network management functions.

5.1.6 Range Extender

Subject to anti-congestion restrictions enforced by the hop count Range Extenders replay all received packets unless either the RE itself is the destination of the frame (e.g., a Ping) or the Range Extender Sender Type is set in the received frame(see Section 6.5.5). APs will also replay frames but since the Access Point Sender Type is set in these frames Range Extenders are permitted to replay frames from APs. When replaying a frame the hop count is decremented.

5.1.7 Battery-only network

Because APs are always powered battery-only networks do not have APs. In battery-only networks the presumption is that all devices are sleeping devices. Since there is no store-and-forward capability the receipt of frames depends on retransmissions by the sender. The proper balance between the frequency of a receiving device awakening, the length of time the device remains listening, and the frequency with which the sending device retransmits must be in place.

5.2 Architecture overview

SimpliciTI software conceptually supports 3 layers as shown in Figure 2. The Application Layer is the only portion that the customer needs to develop. The communication support is provided by a simple set of API symbols used to initialize and configure the network, and read and write messages over air.

The architecture does not strictly follow the OSI Reference model.

a. 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.

b. Since the security support is here implemented as a peer of the Network layer there is no Presentation Layer where this is function is formally implemented in the OSI model.

c. If reliable transport is required that must be implemented by the application so there is no Transport layer.

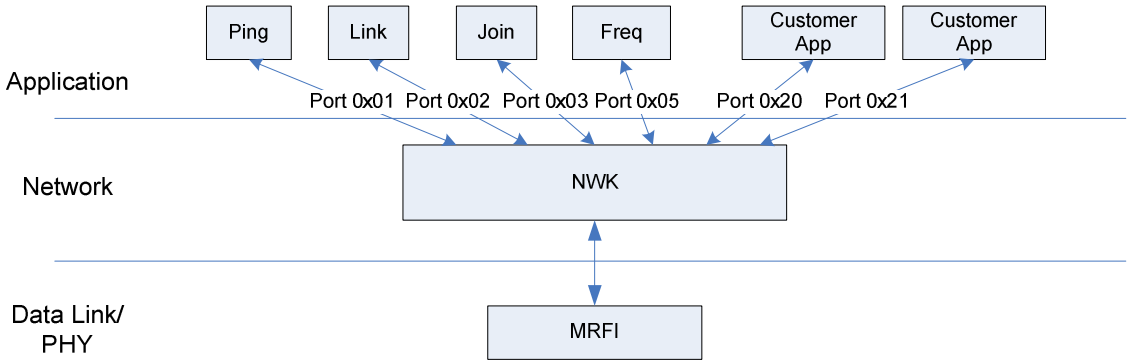


Figure 2: SimpliciTI Architecture

5.2.1 Application Layer

The customer develops the application as the implementation of sensor/actuator interactions with the environment. Using the SimpliciTI API the application can send/receive messages to/from an application peer on another device.

Management of the network itself is supported by SimpliciTI network "applications" as shown above. Each has its own application protocol as would any customer application. These applications can be easily extended and modified as needed. They are described in detail in Section 7.

5.2.2 Network Layer

The network layer actually spans the boundaries of the standard OSI model, as it collapses and hides functionality from the application.

5.2.2.1 Encapsulated Network parameters

Network setup is driven by build-time configuration schemes. These schemes may include code generation tools that generate static objects, header files, and manifest constants. Run time adjustment of some of these may be accessible from application via an **`ioctl()`**–like interface.

Network parameters can include:

1. Base frequency and frequency spacing
2. Number of frequencies supported (for frequency agility table)
3. Modulation method and data rate and other general radio parameters

4. Default and generated network encryption keys
5. Number of store-and-forward messages to hold
6. Device address
7. Repeat rates on Tx-only devices
8. Join and link tokens

5.2.3 Minimal RF Interface (MRFI)

This layer abstracts what is basically a frame read/write interface to the radio. Different radios supported by SimpliciTI require different implementations but the basic interface offered to the network layer is the same for all radios.

Different radios offer different levels of support for typical Data Link and PHY layer responsibilities. MRFI encapsulates these differences.

5.2.4 Board Support Package (BSP)

As a convenience there will be some minimal support for various target MCUs. In the form of a BSP (not shown in architecture drawing) the following will be abstracted:

1. SPI interface to the radio (when appropriate)
2. Interrupt management via GPIO connections to support asynchronous notifications from the radio (when appropriate)
3. LED and button support using GPIO lines.

5.3 Topology

Below are drawings of examples of the topologies supported by SimpliciTI. In each of the following End Device 1 (ED1) is sending data to End Device 2 (ED2). The following legend applies to the line formats:

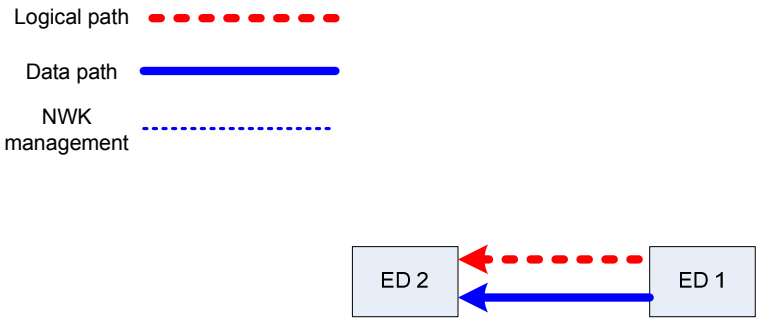


Figure 3: Direct peer-to-peer

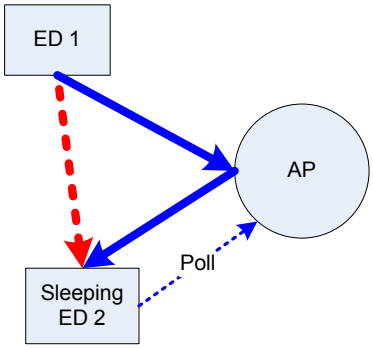


Figure 4: Store-and-forward peer-to-peer through Access Point

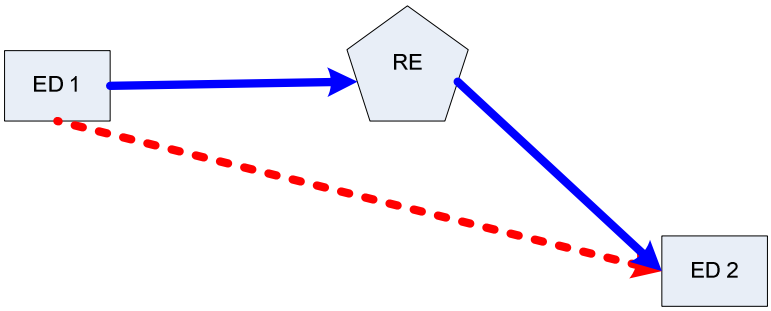


Figure 5: Direct peer-to-peer through Range Extender

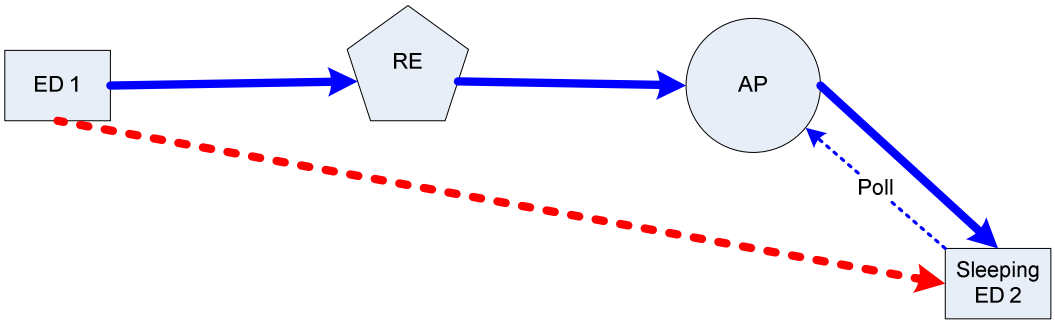


Figure 6: Store-and-forward peer-to-peer through Range Extender and Access Point

5.4 Network discipline

All frames are sent to and from applications on top of the **NWK** layer. There are both user applications and **NWK** applications. The user application frames are peer-to-peer user application messages. Other than fixed network information overhead such as destination address, frame length, etc., the packet's payload is application data.

The network management is accomplished by **NWK** applications whose frames are also peer-to-peer. Network management includes items such as device linking, radio frequency management, security key management, and network membership management. See Section 6.5.4 for more detail.

Frame acknowledgement is the responsibility of the application, be it a user application or a **NWK** application. The acknowledgment is part of the application peer-to-peer discipline. So, for example, the acknowledgment to a **Ping** frame is the echoed **Ping** frame in response.

User applications are connection based. Connections are established as bi-directional (but see Section 7.2.2.1).The connections are established when the devices are linked.

The **NWK** applications are connectionless. Some of these applications implement acknowledgements and some do not. See Section 7 for more detail.

5.5 Network formation

The following will be the procedure for an AP when the initialization call is made (forward references are made here to Section 6.5.4):

- Set the network frequency to the default frequency by broadcasting a **Freq** frame on all tabled frequencies.
- Set the encryption key by broadcasting a **Security** frame on the network frequency.
- Support **Join** frames sent by newly arriving devices.
- Support store-and-forward for joining devices that can receive frames but are not always on.

5.5.1 Address namespace usage

The address namespace consists of all 4-byte values except those that map to the radio's reserved broadcast address.

5.6 Power management

The SimpliciTI simple RF network is intended to support low power devices. In the dual chip solution both the radio stage and the MCU stage may have power managed independently.

It is the responsibility of the application to manage power on the MCU in the dual chip solution. The radio power management can be derived from the MCU power management by using the **ioctl()**-like interface provided in the API (See Section 8.8).

5.7 Transmit-only (Tx-only) devices

Transmit-only devices have the following behaviors:

- They use default network key. They may use another key if Customer provides some means for the stack to detect and implement the key such as an external switch.
- They use retransmission to ensure packet propagation since they cannot receive acknowledgments.
- There are two ways to handle frequency migration on Tx-only devices:
    - They transmit each frame on all tabled frequencies since they cannot receive acknowledgments.
    - Frequency changed by external intervention such as a switch or button.
- When a Tx-only devices links to another device the link token must be valid.
- Other network infrastructure will not acknowledge frames from Tx-only devices.

5.8 Network byte order

Multi-byte number objects are to be represented in little endian format. This convention is followed for number objects contained in network application payloads.

6. Frame structure

Frames consist of 3 logical portions:

- the portion processed by the **PHY/MAC** layers
- the portion supporting network management implemented in the **NWK** layer
- the portion representing the application payload supported in the application layer.

6.1 PHY/MAC frame portion

This portion contains the parts of the frame processed by the hardware. For the CC1100/CC2500 radios this consists of the preamble bits and the sync bits.

6.2 NWK frame portion

This portion of the frame is processed by firmware in the **NWK** layer. The fields in this portion are for network control and specify such parameters as frame type, encryption status, hop count, sequence number, etc.

6.3 Application frame portion

This is the encapsulated application data payload that is delivered to the application as a result of a receive API call for user applications. For **NWK** applications the application portion of the frame is processed as part of the receive thread.

6.4 Address fields

SimpliciTI does not have separate address fields for each layer as some protocols do. The 4-byte SimpliciTI address applies in general. This leads to an architectural ambiguity as to which layer actually "owns" the address.

The solution for SimpliciTI is that the network layer owns the address but control of the address fields themselves rests with MRFI. This is because the encapsulated radio knowledge determines how the fields are actually mapped to the radio frame and how address filtering will work. The frame structure drawing, then will show the address fields as part of the MRFI and not the network layer.

6.5 Frame details

The general frame layout is shown below. Detailed descriptions follow.

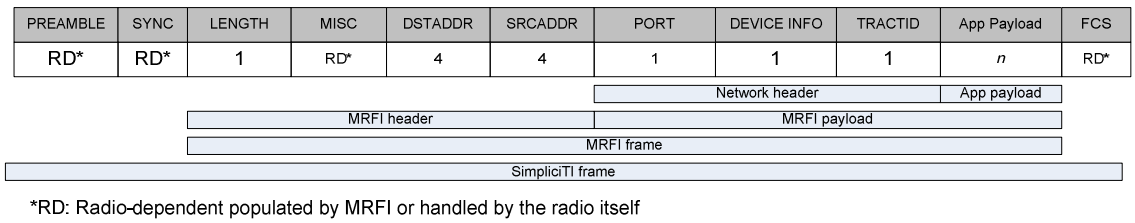


Figure 7: SimpliciTI frame structure (without Security enabled)

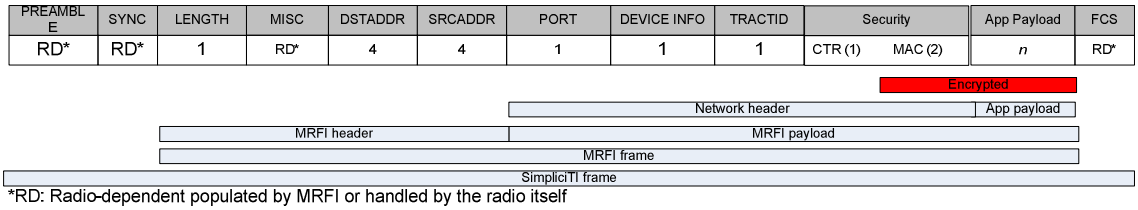


Figure 8: SimpliciTI frame structure (with Security enabled)

Field	Definition	Comments
PREAMBLE	Radio synchronization	Inserted by radio HW
SYNC	Radio synchronization	Inserted by radio HW.
LENGTH	Length of remaining packet in bytes	Inserted by FW on Tx. Partially filterable on Rx.
MISC	Miscellaneous frame fields	Differ for different radios. May be absent.
DSTADDR	Destination address	Inserted by FW. Filterable depending on radio.
SRCADDR	Source address	Inserted by FW.
PORT	Forwarded frame (7), Encryption context (6) Application port number (5-0).	Inserted by FW. Port namespace reserves 0x20-0x3F for customer applications and 0-1F for **NWK** management.
DEVICE INFO	Sender/receiver and platform capabilities	Inserted by FW. Details below.
TRACTID	Transaction id	Inserted by FW. Discipline depends on context. Need not be sequential.
APP PAYLOAD	Application data	$0 \leq n \leq 50$ for non-802.15.4 radios; $0 \leq n \leq 111$ for 802.15.4 radios
FCS	Frame Check Sequence	Usually a CRC appended by the radio hardware.

Table 1: SimpliciTI frame field summary

Some details are discussed below.

6.5.1 PREAMBLE/SYNC/LENGTH

The preamble and sync fields are inserted by the radio hardware.

If such an option exists the radio will be configured to send and receive variable length frames[1]. When so configured a length byte is expected in the frame. This is the byte after the last sync byte is expected to be the length of the frame (not including the length byte itself).

6.5.2 MISC

This field may be absent. Some radios, e.g., IEEE 802.15.4 radios, have information such as frame control bytes in this field.

6.5.3 DSTADDR/SRCADDR

The destination and source addresses are treated as byte arrays. MRFI handles mapping the address to the filtering capabilities of the radio depending on whether the network layer requests filtering . To the extent that hardware filtering is not available across the address array software filtering is implemented by MRFI when filtering is active.

6.5.4 Security Context/PORT

Bits 7-6 hold the security context as defined in the table below:

Bit	Description	Comments
7	Forwarded frame	This bit is set when an Access Point forwards a frame to a polling device on behalf of the sender.
6	0: No encryption 1: Encryption enabled	Default is no encryption.
5-0	Port number	Abstraction denoting the application for which the frame is destined.

Table 2: Security context and Port number

Ports are conceptual abstractions that specify the target application handling the frame. The Port numbers **`0x00–0x1F`** are reserved or assigned as "well known Ports:" with specific services. They are intended for use by network management.

Port numbers **`0x20–0x3F`** are mapped to user handlers. Port **`0x3F`** is reserved as a broadcast Port to be used when the sending device has not been explicitly linked[2] to the receiving device. Port numbers starting at **`0x3E`** and decreasing are reserved for static allocation by user configurations. This arrangement is intended to accommodate commissioning scenarios.

The Ports are mapped by the network layer to Link IDs which are handles provided to the customer application when a link or link-listen succeeds. The application then uses this handle when messages are sent or received.

A conceptual representation of the port abstraction is shown in the following figure. Not all the well known ports are represented.

[1] Frames must be ≤ 64 bytes so that no frame will be larger than the transmit or receive FIFO that is minimal for the radios supported.

[2] Explicit linking *could* be done at build time as well as run time depending on the Customer's needs.

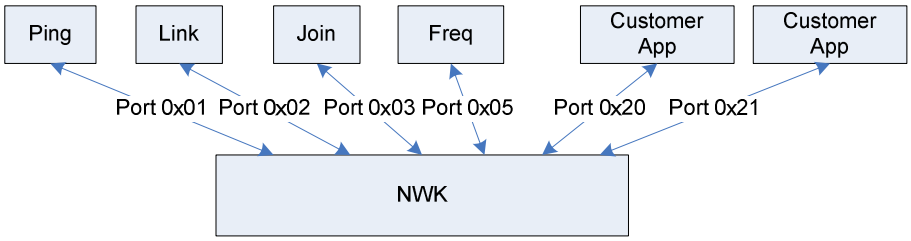


Figure 9: SimpliciTI port abstraction

6.5.5 DEVICE INFO

Device information byte provides information about the device issuing the frame.

The device information bits are defined as follows:

Bit	Description	Comments
7	Acknowledgement request	This bit is set by the **NWK** layer when a user application requests and acknowledgment from the peer.
6	00: Controlled listen 01: Sleeps/polls	Receiver type Sleeping devices may either poll or periodically listen. If they are specified as polling devices the Access Point will provide store-and-forward support. Otherwise the device listens at its own schedule under its own control.
5-4	00: End Device 01: Range Extender 10: Access Point 11: Reserved	Sender Type Most important for Range Extender to prevent a RE from forwarding a frame from another RE. This mitigates broadcast storms. Refers to device sending the frame and may not be the same as the device whose source address is specified in the frame.
3	Acknowledgement reply	This bit is set when the **NWK** layer sends a frame in response to an acknowledgment request.
2-0	Hop count	Decremented by each transmitting device until 0. Then discarded.

Table 3: DEVICE INFO bit values

6.5.6 TRACTID

The transaction ID is used by **NWK** to add robustness to the communications. The field can be used for example to match replies to outstanding messages or to help recognize a duplicate frame. Each sending side maintains its own transaction ID discipline.

7. NWK applications

The network layer applications on well-known ports are peer-to-peer objects used to manage the network. Some applications are mandatory. Some are conditional in that they must be present under certain conditions. A list follows. The conditions under which the conditional applications need be supported are noted.

Application	Port	Device Support
Ping	0x01	Mandatory except for Tx-only devices.
Link	0x02	Mandatory
Join	0x03	Only functional when the network has AP support.
Security	0x04	Only functional when the network has AP support.
Freq	0x05	Only functional when the network has AP support.
Mgmt	0x06	General use **NWK** management application. Used as the poll port.

Table 4: NWK applications

The first byte in the payload for each **NWK** application is an information byte. This byte conveys application-specific information. For example, there could be length information or option information such as whether to acknowledge the frame. In addition each application has its own payload discipline. Length information will be almost mandatory if block cipher encryption is used because in this case the payload length cannot be inferred from the frame length.

The payloads are described below.

7.1 Ping (Port 0x01)

7.1.1 Semantics

The purpose of this application is to detect the presence of a specific device.

This application sends a message from one device to another device. The receiving device echoes the payload contents back to the originator. The sequence is blocking in the sense that the originator will wait for the reply. A timeout parameter is applied.

Pings are unicast.

7.1.2 Payload (Client)

Address: unicast

Request	TID
1	1

Figure 10: Ping Client payload

The payload consists of just the transaction ID (TID). The TID (transaction ID) is used to detect missing or duplicate frames.

7.1.3 Payload (Server)

Address: unicast

REQ Reply	TID
1(0x8x)	1

Figure 11: Ping Server payload

The payload consists of echoing the transaction ID (TID). The TID (transaction ID) is used to detect missing or duplicate frames. The msb in the application information byte signifies that this is a reply.

7.2 Link (Port 0x02)

7.2.1 Semantics

The purpose of this application is to support the link management of two peers.

Currently supported requests on the Link port are as follows:

Request	Value	Comments
Link	1	Sender requests a link (connection) be established.
Unlink	2	Tear down link (connection).

Table 5: Link request values

7.2.2 Link request

The frame source device (Client) is linked to the destination device (Server).[3] Once the link is established each side can send messages to the other. The connection is bi-directional.

The Client's link message is broadcast. The Server's reply is unicast back to the client. The Server remembers the Client's address from the Client's broadcast message. The Client remembers the Server's address from the unicast reply.

The complete exchange is necessary for the link to be established. At the completion of this function a bi-directional connection may exist between the two devices.

From the Client side the sequence is blocking in the sense that the Client will wait for the reply. A timeout parameter is applied. From the Server side listen call is also blocking. also with a timeout parameter.

In this simple protocol only one link per listen will be permitted, first come first served. Presumably the Client and Server will be engaged in the Link exchange based on some user intervention such as a button press, though this isn't required.

Multiple devices can be linked by doing multiple Link sessions. Each Client will be provided with a unique Access Token. A Client can also talk to multiple Servers again by instigating separate Link sessions.

7.2.2.1 Client side

Address: broadcast

[3] The Client-Server peer locus is arbitrary. The link transaction simply requires one talker to send a link frame and one listener to hear it. These are called Client and Server respectively here to distinguish the roles during the transaction. The actual application may or may not maintain these roles.

Request	TID	Link Token	Local Port	Rx Type	Protocol Ver
1 (0x01)	1	4	1	1	1

Figure 12: Client side Link payload (no Security)

Request	TID	Link Token	Local Port	Rx Type	Protocol Ver	CTR Value
1 (0x01)	1	4	1	1	1	4

Figure 13: Client side Link payload (with Security)

The payload consists of a request byte, a transaction ID, the 4-byte link token, a local port number, the Rx type of the Client (see Section 6.5.5), and the protocol version of the source.

The transaction ID (TID) is used by the link listening application to filter out duplicates.

The token is used to uniquely identify the link context and should be used to prevent rogue devices (either intended or unintended) from linking to the network. The customer can set these tokens to be unique in any manner. It is the means by which a device is prevented from linking to a device on the incorrect network. The token is a network-wide parameter. See Section 10.1.1.

The local port number is the port number that the remote device should use to communicate back to the linking device. If the linking device is Tx-only or does not need to receive any messages over this connection then a value of 0 will be supplied. In this case the connection is unidirectional.

The Rx type is used by the listener as a hint about setting the hop count when sending a frame to the client. If the peer is a polling device then sending to the Client can use a hop count that is large enough to get only to the Access Point. Otherwise, the hop count can be set to the number of hops it took to get to the listener which can be deduced from the received frame[4].

The protocol version can be used to determine run-time context. This can include denying the link if the versions do not match or compatibility modes if older versions are to be respected.

If security is enabled a modified **CTR** mode security is used. In this case the frame contains a four-byte counter value with which the sending device will begin when messages are sent across the connection.

In topologies with an AP the AP will supply the link token to joining devices. In topologies without APs the default link token is used.

7.2.2.2 Server side

Address: unicast

REQ Reply	TID	Local Port	Rx Type
1 (0x81)	1	1	1

Figure 14: Server side Link response payload (no Security)

REQ Reply	TID	Local Port	Rx Type	CTR Value
1 (0x81)	1	1	1	4

Figure 15: Server side Link response payload (with Security)

[4] This may not be a safe practice if the peer such as a remote control device can move.

The response payload contains a transaction ID, a local port number and the Server side Rx type.

The Local Port object will be used to populate the PORT field in the frame when the Client's linked application sends a message to its peer on the Server. The association is maintained in the **NWK** layer.

The Rx type is used for the same purpose as the Client side, i.e. as a hint to how to set the hop count in frames sent to the Server.

As with the Client side, if security is enabled a modified **CTR** mode security is used. In this case the reply frame contains a four-byte counter value with which this sending device will begin when messages are sent across the connection. Note that messages may not be sent in both directions though bi-directional messaging is supported.

The most significant bit in the REQ Reply byte signifies that this is a reply.

7.2.3 Unlink request

Either peer may request that a connection be terminated. The requesting side waits for the results of the request from the peer. The peer will reply if possible. This request is supplied for completeness. A connection-based protocol like SimpliciTI could lead to awkward designs if the connection base is too dynamic.

The unlink request originator receives a result code indicating the status of the peer that is the target of the request. If the peer does not respond[5] the application is notified. Normally the originator will reclaim the connection resources on a successful result. Other corrective action is possible if the unlink fails with respect to the peer.

The peer also will normally reclaim connection resources. Since it is possible that the peer will miss the originator's request application designs should account for this.

7.2.3.1 Originator side

Address: Unicast

Request	TID	Remote port
1 (0x02)	1	1

Figure 16: Originator side Unlink payload

The originator supplies the port on which it receives messages from the peer. This is the remote port from the peer's perspective. The peer uses this value together with the source address in the frame to find the correct entry in its Connection table.

7.2.3.2 Peer side

Address: Unicast

REQ Reply	TID	Result code
1 (0x82)	1	1

Figure 17: Peer side Unlink reply payload

The result code indicates whether the peer successfully destroyed its side of the connection.

[5] For example the peer may not be listening when the request is sent.

7.3 Join (Port 0x03)

7.3.1 Semantics

This application guards entry to the network in topologies with APs.

When the network has an AP each device must join the network. Upon joining a device is optionally supplied with the following:

- Encryption context
- Link token
- Store-and-forward support if it is a sleeping Rx device

By supplying the first three items the network is more robust to intrusion by rogue devices.

The attempt to join should timeout in case the AP isn't up yet. It is up to the Customer application to keep trying to join. Since the presence of an AP is a build-time parameter all devices know how to behave.

Currently supported join request bytes conveys are as follows:

Request	Value	Comments
Join	1	Sender requests to Join.

Table 6: Join request values

7.3.1.1 Join request: Client side

Address: broadcast

Request	TID	Join Token	Number of connections	Protocol Ver
1 (0x01)	1	4	1	1

Figure 18: Join Client side payload

The payload consists of a request byte, a transaction ID, the 4-byte join token, the number of connections allowed on the source, and the protocol version of the source.

As in previous descriptions the transaction ID (TID) is used by the application to filter out duplicates.

The Join Token is used to verify that the device is valid on the network.

The number of connections is used by the Access Point (if present) as a hint to allocate resources. In the topology in which the Access Point functions as a data hub in a Star configuration the receipt of a Join frame can be used to stimulate a listening for the joining client to link. If the joining client does not intend to link (such as a Range Extender device) the number of connections field should be set to 0. The Access Point can then infer that there will be no peers on the joining device so it need not listen for a link frame or allocate connection table resources.

The protocol version can be used to determine run-time context. This can include denying the join if the versions do not match or compatibility modes if older versions are to be respected.

7.3.1.2 Join request: Server side

Address: unicast

REQ Reply	TID	Link Token	FUNC/LEN	KEY
1 (0x81)	1	4	1	n

Figure 19: Join Server side payload

The Server returns the source TID and the value that the Client should use as the link token for the network. Encryption context is also returned. The next bytes use the same format as those in the Security application (see Section 7.4.1.1).

The msb in the application information byte signifies that this is a reply.

7.4 Security (Port 0x04)

7.4.1 Semantics

This application is used to change security information such as encryption keys and encryption context. For this application the AP initiates the exchange so the application appears only in networks that have an AP and then only on non Tx-only End Devices and Range Extenders.

Having the AP initiate these exchanges also helps with sleeping Rx devices that are managed by the AP so that they do not get out of synchronization. The AP can maintain multiple encryption contexts and alert the sleeping devices when they awaken and poll.

The AP uses this application to enforce the encryption context and communicate a new key to all other devices.

These frames will always be sent encrypted using the default encryption key.

7.4.1.1 Initiator (AP) side

Address: broadcast

Request	TID	FUNC/LEN	KEY
1	1	1	n

Figure 20: Security initiator (AP) payload

As in previous descriptions the transaction ID (TID) is used by the application to filter out duplicates. The next bytes are used as follows:

Field	Definition	Description
FUNC/LEN	N/A	Reserved – currently unused.

Table 7: Security initiator payload details

7.5 Freq (Port 0x05)

7.5.1 Semantics

The protocol on this port supports the following operations. Details follow.

Request	Value	Comments
Change Channel	1	Only the AP can issue this command.

Echo request	2	This is functionally a frequency application ping that is used by a device to scan for the current channel. Only the AP will respond to this frame
Change channel request	3	A non-AP device can request a channel change. But the command must still be issued by the AP. Future support.

When the network has an AP this application provides frequency agility support. The AP initiates the change in frequency by sending a broadcast frame to the Freq Port announcing the frequency change context.

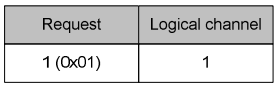
If the frequency of the network changes sleeping devices will find the AP by cycling through the frequency table upon awakening. They will know to do so because they will not receive an **ACK** when they poll. The frequency application echo request is used for this scan.

There is provision for a non-AP device to request a change of channels. The actual change is initiated by the AP at its discretion.

7.5.1.1 Change channel command

7.5.1.1.1 Initiator (AP) side

Address: broadcast



Request	Logical channel
1 (0x01)	1

Figure 21: FREQ application change channel command

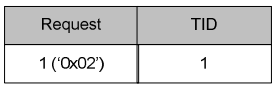
The payload consists of the request indicator '1' and the logical channel number of the new radio channel. The target channel is a logical channel number that is most directly implemented as an index into a table containing everything required for a frequency change context.

A transaction ID is not needed for this frame. It is broadcast so replies will not occur and therefore do not need to be matched with sent frames. On the receiver side it does no real harm if duplicates are received. This is unlikely anyway since the receipt of any frame will cause the channel to change.

7.5.1.2 Echo request (Frequency Application ping)

7.5.1.2.1 Client side

Address: broadcast/unicast to AP



Request	TID
1 ('0x02')	1

Figure 22: FREQ application echo request (ping)

The payload consists of the request indicator '2' and the transaction ID. The client side sends this frame to the Access Point if it has gleaned the AP address from the Join reply. Otherwise it broadcasts the frame. A broadcast is necessary if the scan is done at startup before the Join has succeeded.

7.5.1.2.2 Server side

Address: unicast

REQ Reply	TID
1 ('0x82')	1

Figure 23: FREQ application echo reply

The payload consists of the reply indicator '0x82' (original command value with msb turned on).and the TID.

The client side tries the ping on each channel until this reply is received. The client then can infer the current channel being supported.

7.5.1.3 Channel change request

Using this frame a non-AP device can request that the Access Point issue a change channel command. The AP need not comply.

There is no reply to this command. If the AP decides to comply it will issue a channel-change command (see Section 7.5.1.1.1).

7.5.1.3.1 Initiator side

Address: unicast to Access Point

Request
1 ('0x03')

Figure 24: FREQ application change channel request

The payload consists of the command indicator '3'. There will be no reply to this frame and no transaction ID is needed.

7.6 Mgmt (Port 0x06)

7.6.1 Semantics

This is a general management port to be used to manage the device. It will be used to access a device out-of-band and could be used, for example, to reset other Port state machines, Transaction IDs, etc. It will also be used by sleeping devices to poll the AP.

This is the port used by sleeping/polling end devices to poll the AP for frames.

This can be used as an emergency meet-me port. Messages can be sent in clear text to this port at any time.

7.6.1.1 End device polling

An End Device announces its intent to poll the Access Point for messages when it joins the network. This context is carried in the **DEVICE INFO** byte (see 6.5.5). The poll query is sent on the **Mgmt** port. The server-side reply is sent on the port about which the query was made. Note that this query is sent as a side effect of a receive call and is not part of the SimpliciTI API.

7.6.1.1.1 Client side

Address: unicast to Access Point

Request	TID	Port	Address
1	1	1	4

Figure 25: End Device Poll

The query of the Access Point is made for a specific port and address. The port is derived from the Link ID supplied in the receive call originating in the application. The address is the address of the client's peer, i.e., the SimpliciTI address of the peer from which the client is expecting a frame.

The transaction ID is used by the AP so that it can discern duplicate poll frames and not send a frame prematurely to the client.

7.6.1.1.2 Server side

The reply from the Server (Access Point) is not on the **Mgmt** port. The Server replies on the port queried.

If there is a frame waiting for the client on the port queried the Server sends the frame. If no frames are waiting on that port the Server constructs a frame with no payload and sends it to that port on the Client. Both these frames use the transaction ID sup[plied in the poll request.

8. API

The focus of the API is to encapsulate network functionality in a manner that allows a functional reliable network to form with little effort from the application. The main effect of this approach is that the resulting network sacrifices flexibility for simplicity.

The following functionality is supported by the API:

- Initialization
- Linking
- Application peer-to-peer messaging
- Device management

At the **NWK** layer most but not all exchanges have acknowledgments. At the application layer the acknowledgment status is controlled by the application peer-to-peer

8.1 smplStatus_t SMPL_Init(uint8 (*pCB)(linkID))

This call causes all **NWK** initialization to occur. A side effect of the initialization is an attempt to Join the network. If it is an Access Point supported network the Join request (issued silently) will negotiate network parameters in exchanges with the Access Point.

This call also causes the radio to be set up and initializes all **NWK** constructs based on system parameters defined in Section 10. If the device is an AP this may involve such other actions as generating an encryption key and a link token.

The argument is a pointer to a function that takes a Link ID argument and returns a uint8. This allows the application to provide callback function pointer to the frame handling logic for received frames. If the following conditions are met the callback will be invoked with the Link ID of the received frame as the argument

1. The supplied function pointer is not null
2. The destination port of the received frame is a user port
3. The destination address of the frame is that of the device
4. The connection is valid (i.e., the port is in the connection table).

If the function returns non-zero the frame is considered as having been handled and the frame resources are released for re-use. Otherwise the frame remains in the input frame queue for later handling by the application. In either case the application retrieves the received frame by executing a call to **`SMPL_Receive()`** with the link ID conveyed in the callback argument as the link ID argument. This call is guaranteed to succeed.

If the device is an Access Point and the AP is designated as a data hub the callback will also be invoked if the joining device supports End Device objects. In this case the Link ID will have the value 0x00. In this case the application cannot retrieve the received frame directly since it is handled in the **`NWK`** layer. The callback is used as a signal to the AP that it should execute a **`SMPL_LinkListen()`** to receive the subsequent link frame from the joining device.

Note that if the callback conditions are met the handler will be invoked in the receive ISR context. Designers should be careful about doing too much in this thread.

8.2 smplStatus_t SMPL _Link(linkID_t *linkID)

Link as a Client to another device that is listening (Server).

Listening devices will acknowledge. Once the acknowledgment as been received the call will return with a link ID that should be used for all subsequent messaging to and from the device to which the link has been made. This is not a blocking call and will return status that indicates whether the link succeeded. If it did not succeed the application can re-try.

8.3 smplStatus_t SMPL _Unlink(linkID_t linkID)

Tear down the connection bound to the supplied Link ID. Part of Extended API.

This call will unconditionally disable the local connection table entry bound to the supplied Link ID. In addition a message will be sent to the Link Port of the peer requesting that the peer tear down its side of the connection (see Section 7.2.3). The disabling of the connection on the peer may or may not succeed when initiated from the local device.

8.4 smplStatus_t SMPL _LinkListen(linkID_t *linkID)

This is the companion to the Link call and waits for a Client Link message.

Only the first valid Link message is accepted as a result of this call. Only one outstanding listen can be in force at any given time. Link messages that arrive before the listen is executed are discarded.

There is some defense against receiving duplicate link frames during a listen. The defense is designed to detect retransmissions or late frames. But it will also detect link frames coming from a device that was reset provided that the reset device generates the same link frame each time. For example, a device that randomly generates a new device address each time will not be detected as a duplicate linking device and will consume resources.

This is a timed blocking call and will return when either a valid link frame is received or a (configurable) fixed amount of time has elapsed. The application can discriminate between these two by the return code. The application is then free to implement a recovery policy including another **SMPL_LinkListen()**.

8.5 smplStatus_t SMPL _SendOpt(linkID_t lid, uint8 *msg, uint8 len, uint8 opt)

This call sends the message **msg** of length **len** to the device with link id **lid** applying the send options present in the **opt** bit map. The **lid** is the one returned by the **SMPL_Link()** call.

It returns after the message is sent. This is so that discipline can be maintained over the transmit context and the power context of the MCU and the radio.

The message size is limited to the transmit FIFO size which depends on the radio. Segmentation and reassembly if needed is the responsibility of the application

8.6 smplStatus_t SMPL _Send(linkID_t lid, uint8 *msg, uint8 len)

This API is maintained for legacy reasons. Initial version of SimpliciTI did not support the transmit option argument.

8.7 smplStatus_t SMPL _Receive(linkID_t lid, uint8 *msg, uint8 *len)

This call checks the frame buffers for a message from link id **lid** and returns the oldest one and its length. The frame buffers are populated as a result of handling a radio Rx interrupt. The **SMPL_Receive()** call does **not** affect the radio state, i.e., it does **not** turn the radio on or place it into the Rx state. It simply checks to see if a relevant frame has been received in an ISR thread and is being held in the frame buffer.

The Link ID supplied in the call is one originally defined by having completed a successful link operation.

8.8 smplStatus_t SMPL_Ping(linkID_t lid)

This API is part of the Extended API set. It provides application level access to the **NWK** ping application. It is a simple means by which an application can determine the presence of the device on which a peer resides.

This call does *not* ping the peer itself. The Link ID supplied in the formal parameter is sued to glean the address of the device hosting the peer. The API causes the NWK to send a ping frame to the Pi9ng NWK application on the device hosting the peer. The result of this trasnction is returned to the caller.

There is a useful side effect of this call if Frequency Agility is enabled. The call will result in a channel scan if a reply is not received on the current channel. A return code indicating success implies both that the peer device is there and that the channel setting on the local device is current.

8.9 smplStatus_t SMPL_Commission (addr_t *addr, uint8_t lPort, uint8_t rPort, linkID_t *lid);

This API is part of the Extended API set. It is used to do static configuration of a peer connection. It can be used to eliminate the need for a link session between peers. It is a hard-wired connection configuration that requires knowledge of the peer device address (**addr**) and both the local (**lPort**) and remote (**rPort**) port assignments. On successful return from this call a Link ID (**lid**) is provided. This value is assigned by the **NWK**.

The local port namespace reserves a defined area for static assignment. The size of this area is a build time setting. The default size is 1. The automatic port assignment mechanism will not use this part of the namespace when used for explicit link sessions.

8.10 void SMPL_Ioctl(ioctlObject_t object, ioctlAction_t action, void *val)

This is the means by which the application can configure the **NWK** at run time. The currently available values are described in Section 10.2.

8.11 Pseudo-code example

The following pseudo-code snippet shows the API sequence for how a device would join a SimpliciTI network with an AP and then link to two other devices and send different messages.

```
void main()
{
```



```
  linkID_t linkIDLow, linkIDHigh;
  uint32   temp;

  // Initialize the board's HW
  BSP_InitBoard();

  // Initialize SimpliciTI. The initialization will cause the
  // device to Join as a side effect. The Join will return
  // security key and a token to be used in linking
  // sequences for this network. This is all hidden from
  // the application.

  SMPL_Init(0);  // no callback supplied

  // Establish links to two different (logical) devices.
  // One will get a message if the sampled temperature is
  // to low. The other gets a message if it is too high.
  SMPL _Link(&linkIDLow);
  SMPL _Link(&linkIDHigh);
  while (TRUE)
  {
    // put board to sleep until timer wakes it up
    // to read the temperature sensor
    MCU_Sleep();
    HW_ReadTempSensor(&temp);
    if (temp > TOO_HIGH)
    {
      SMPL _Send(linkIDHigh, "Hot!", 4);
    }
    if (temp < TOO_LOW)
    {
      SMPL _Send(linkIDLow, "Cold!", 5);
    }
  }
}
```

When this device is linking the target device must have executed a **`SMPL_LinkListen()`** first. This is how the linked pairs are identified. The two links need not be on the same physical device.

There would be a **SMPL_Receive(linkID…)** after each of the **SMPL_Send()** statements if the application has implemented acknowledgments. The receive activity might have to be conditioned with a timeout discipline to know whether to re-transmit the message if that were the intent. More likely, though, this example would be simple unacknowledged messages unless missing one was critical.

9. Sequence diagrams and state machines

10. Customer configurable objects

10.1 Build time

10.1.1 Non-radio items

The following are the build-time configurable items. Each has a default value so none actually need to be modified by the customer. Only those currently supported are listed. The macros defining the values are found in the files **smpl_config.dat** and **smpl_nwk_config.dat**, a pair for each of the 3 device types.

Item	Default value	Description
MAX_HOPS	3	Maximum number of times a frame is resent before frame dropped. Each RE and the AP decrement the hop count and resend the frame.
MAX_HOPS_FROM_AP	1	Maximum distance and ED can be from the AP. Using this hop count significantly reduces the broadcast storm that can result from an ED poll if **MAX_HOPS** is used. Only for AP networks.
NUM_CONNECTIONS	4	Number of links supported as a result of both **SMPL_Link()** and **SMPL_LinkListen()** calls. Should be 0 if the device supports no ED objects (APs or REs).
MAX_APP_PAYLOAD	10	Maximum number of bytes in the application payload.
SIZE_INRAME_Q	2	Number of frames held in the Rx frame queue. Can be 0 for Tx-only devices, or for devices that never receive frames.
SIZE_OUTFRAME_Q	2	Number of frames held in the Tx frame queue. Some **NWK** applications keep Tx frames around to match replies.
DEFAULT_JOIN_TOKEN	**0x01020304**	Joining a network requires this value to match on all devices. It is sent in the Join message and matched in the receiving Access Point.
DEFAULT_LINK_TOKEN	**0x05060708**	Obtaining link access to a network device requires this value to match on all devices. It is sent in the Link message and matched in the receiving device.
THIS_DEVICE_ADDRESS	**0x12345678**	Each device address must be unique. The address assignment should lock out **0xnnnnnn00** and **0xnnnnnnFF** which are broadcast address for the CC1100/CC2500-class radios.
FREQUENCY_AGILITY	**Not defined**	When defined enabled support for Frequency Agility. Otherwise only the first entry in the channel table is used.
NVOBJECT_SUPPORT	**Not defined**	Support for saving and restoring connection context.
SMPL_SECURE	**Not defined**	Enables SimpliciTI security support.
APP_AUTO_ACK	**Not defined**	Support for application layer acknowledgment support.
EXTENDED_API	**Not defined**	Support for `SMPL_Ping()`, `SMPL_Unlink()`, and `SMPL_Commission()`.
SW_TIMER	**Not defined**	If enabled uses software to implement delays.
Access Point Devices		
ACCESS_POINT	**Defined**	
NUM_STORE_AND_FWD_CLIENTS	**10**	Number polling End Devices supported by this Access Point
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.
Range Extender Devices		
RANGE_EXTENDER	**Defined**	
End Devices		
END_DEVICE	**Defined**	Defined unless it is a application hosted on a device tht is

		also either a Range Extender or Access Point.
RX_POLLS	**Not defined**	Define if the device polls for frames.

Table 8: Build-time Customer configurable non-radio parameters

10.1.2 Radio configuration

A large number of radio parameters are available to be read and modified if necessary. The SimpliciTI environment will be shipped with default settings for these parameters.

Basically they are the parameters set by SmartRF. They will not be listed individually here. Some may not have the default SmartRF values and some may be subsequently modified by startup code either by **NWK** or application either directly using **SMPL_Ioctl()** or indirectly during startup exchanges.

With few exceptions these parameters will not be directly accessible to customers through the SimpliciTI API. If a customer needs to modify any of these parameters they will have access to them directly in the code but they will not be supported with an API.

10.2 Run time

Run time configuration of **NWK** objects is accomplished using the **SMPL_Ioctl(*object, request, value*)** API. The following Table summarizes the run-time objects.

Object	Description	Comments
IOCTL_OBJ_FREQ	Get/Set radio frequency	Frequency agility. May be used by application or **NWK**.
IOCTL_OBJ_CRYPTKEY	Set encryption key	Customer may provide external means for user to set a non-default key. Requires reset to take effect.
IOCTL_OBJ_RAW_IO	Application layer access to the frame header to directly send or receive a frame.	This object is used for example to ping another device where the network address of the target device is supplied directly and not done through the connection table.
IOCTL_OBJ_RADIO	Application layer access to some radio controls.	Limited access to radio directly. For example, sleeping and awakening the radio and getting signal strength information.
IOCTL_OBJ_AP_JOIN	Access Point join-allow context	Interface to control whether Access Point will allow devices to join or not.
IOCTL_OBJ_ADDR	Get/set device address	Permits run-time address generation for a device. Set function *must* be done before the **SMPL_Init()** call.
IOCTL_OBJ_CONNOBJ	Connection object	Delete a connection entry. Accessed with Link ID. Affects this device only – does not gracefully tear down connection.
IOCTL_OBJ_FWVER **IOCTL_OBJ_PROTOVER**	Firmware and protocol versions.	Get only. FW version a byte array of length **SMPL_FWVERSION_SIZE.** Protocol version a `uint8_t`.
IOCTL_OBJ_NVOBJ	Non-volatile memory object	Get and set the NV object. This permits a device to save connection context across power cycles so that communications can be restored in event of a reset.
IOCTL_OBJ_TOKEN	Link or Join token object	Allows application to get/set a network access token.

Table 9: Customer configurable run-time objects

