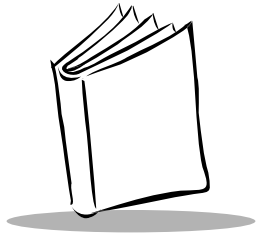
SPT 1740 Spectrum24® Driver Extensions Library Developer's Guide

72-38524-01
Revision A
July 1999

symbol®

Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
http://www.symbol.com



Contents

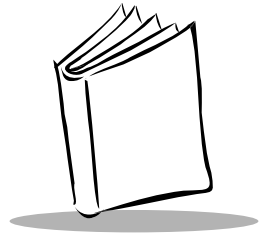
About This Guide

Chapter 1. Spectrum24® Radio Overview

Chapter 2. Spectrum24® Network Overview

Chapter 3. SPT 1740 Spectrum24 Introduction

Chapter 4. Spectrum24 Radio Library Function Calls (API)



About This Guide

Introduction

The *SPT 1740 Spectrum24® Driver Extension Library Developer's Guide* provides an overview of Spectrum24® wireless operation, as well as information about the Spectrum24 library functions that can be used by developers for the SPT 1740.

Notational Conventions

This document uses these conventions:

- "terminal" or "mobile unit (MU)" refers to the SPT 1740.
- "User" refers to anyone using an application on the SPT 1740.
- *Italics* are used to highlight specific items in the general text, and to identify chapters and sections in this and related documents.
- Bullets (•) indicate:
  - lists of alternatives or action items.
  - lists of required steps that are not necessarily sequential.
- Numbered lists indicate a set of sequential steps, i.e., those that describe step-by-step procedures.

Related Documents

- *SPT 1700 Series Quick Reference Guide*, p/n 70-37543-xx
- *SPT 1700 Series Product Reference Guide*, p/n 70-37544-xx
- *Spectrum24 Access Point AP-3020 Product Reference Guide*, p/n 70-20504-xx.

Service Information

If you have a problem with the SPT 1700 equipment, contact the Symbol Support Center. If your problem cannot be resolved over the phone, you may need to return your equipment for servicing. If that is necessary, you will be given special directions.

Note: *Symbol Technologies is not responsible for any damages incurred during shipment if the approved shipping container is not used. Shipping the units improperly can possibly void the warranty. If the original shipping container has not been kept, contact Symbol to have another sent to you.*

Symbol Support Centers

For service information, warranty information or technical assistance contact or call the Symbol Support Center in:

United States

Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
1-800-659-2240

Canada

Symbol Technologies Canada, Inc.
2540 Matheson Boulevard East
Mississauga, Ontario, Canada L4W 4Z2
905-629-7226

United Kingdom

Symbol Technologies
Symbol Place
Winnersh Triangle, Berkshire RG41 5TP
United Kingdom
0800 328 2424 (Inside UK)
+44 118 945 7529 (Outside UK)

Asia/Pacific

Symbol Technologies Asia, Inc.
230 Victoria Street #04-05
Bugis Junction Office Tower
Singapore 188024
337-6588 (Inside Singapore)
+65-337-6588 (Outside Singapore)

Australia
Symbol Technologies Pty. Ltd.
432 St. Kilda Road
Melbourne, Victoria 3004
1-800-672-906 (Inside Australia)
+61-3-9866-6044 (Outside Australia)

Austria
Symbol Technologies Austria GmbH
Prinz-Eugen Strasse 70
Suite 3
2.Haus, 5.Stock
1040 Vienna, Austria
1-505-5794 (Inside Austria)
+43-1-505-5794 (Outside Austria)

Denmark
Symbol Technologies AS
Gydevang 2,
DK-3450 Allerod, Denmark
7020-1718 (Inside Denmark)
+45-7020-1718 (Outside Denmark)

Europe/Mid-East Distributor Operations
Contact your local distributor or call
+44 118 945 7360

Finland
Oy Symbol Technologies
Kaupintie 8 A 6
FIN-00440 Helsinki, Finland
9 5407 580 (Inside Finland)
+358 9 5407 580 (Outside Finland)

France
Symbol Technologies France
Centre d'Affaire d'Antony
3 Rue de la Renaissance
92184 Antony Cedex, France
01-40-96-52-21 (Inside France)
+33-1-40-96-52-50 (Outside France)

Germany
Symbol Technologies GmbH
Waldstrasse 68
D-63128 Dietzenbach, Germany
6074-49020 (Inside Germany)
+49-6074-49020 (Outside Germany)

Italy
Symbol Technologies Italia S.R.L.
Via Cristoforo Columbo, 49
20090 Trezzano S/N Navigilo
Milano, Italy
2-484441 (Inside Italy)
+39-02-484441 (Outside Italy)

Latin America Sales Support
7900 Glades Road
Suite 340
Boca Raton, Florida 33434 USA
1-800-347-0178 (Inside United States)
+1-561-483-1275 (Outside United States)

Mexico
Symbol Technologies Mexico Ltd.
Torre Picasso
Boulevard Manuel Avila Camacho No 88
Lomas de Chapultepec CP 11000
Mexico City, DF, Mexico
5-520-1835 (Inside Mexico)
+52-5-520-1835 (Outside Mexico)

Netherlands
Symbol Technologies
Kerkplein 2, 7051 CX
Postbus 24 7050 AA
Varsseveld, Netherlands
315-271700 (Inside Netherlands)
+31-315-271700 (Outside Netherlands)

Norway
Symbol Technologies
Trollasveien 36
Postboks 72
1414 Trollasen, Norway
66810600 (Inside Norway)
+47-66810600 (Outside Norway)

South Africa
Symbol Technologies Africa Inc.
Block B2
Rutherford Estate
1 Scott Street
Waverly 2090 Johannesburg
Republic of South Africa
11-4405668 (Inside South Africa)
+27-11-4405668 (Outside South Africa)

Spain
Symbol Technologies S.A.
Edificioi la Piovera Azul
C. Peonias, No. 2 - Sexta Planta
28042 Madrid, Spain
9-1-320-39-09 (Inside Spain)
+34-9-1-320-39-09 (Outside Spain)

Sweden
Symbol Technologies AB
Albygatan 109D
Solna
Sweden
84452900 (Inside Sweden)
+46 84452900 (Outside Sweden)

If you purchased your Symbol product from a Symbol Business Partner, contact that Business Partner for service.

Warranty

Symbol Technologies, Inc (“Symbol”) manufactures its hardware products in accordance with industry-standard practices. Symbol warrants that for a period of twelve (12) months from date of shipment, products will be free from defects in materials and workmanship.

This warranty is provided to the original owner only and is not transferable to any third party. It shall not apply to any product (i) which has been repaired or altered unless done or approved by Symbol, (ii) which has not been maintained in accordance with any operating or handling instructions supplied by Symbol, (iii) which has been subjected to unusual physical or electrical stress, misuse, abuse, power shortage, negligence or accident or (iv) which has been used other than in accordance with the product operating and handling instructions. Preventive maintenance is the responsibility of customer and is not covered under this warranty.

Wear items and accessories having a Symbol serial number, will carry a 90-day limited warranty. Non-serialized items will carry a 30-day limited warranty.

Warranty Coverage and Procedure

During the warranty period, Symbol will repair or replace defective products returned to Symbol’s manufacturing plan in the US. For warranty service in North America, call the Symbol Support Center at 1-800-659-2240. International customers should contact the local Symbol office or support center. If warranty service is required, Symbol will issue a Return Material Authorization Number. Products must be shipped in the original or comparable packaging, shipping and insurance charges prepaid. Symbol will ship the repaired or replacement product freight and insurance prepaid in North America. Shipments from the US or other locations will be made F.O.B. Symbol’s manufacturing plant.

Symbol will use new or refurbished parts at its discretion and will own all parts removed from repaired products. Customer will pay for the replacement product in case it does not return the replaced product to Symbol within 3 days of receipt of the replacement product. The process for return and customer’s charges will be in accordance with Symbol’s Exchange Policy in effect at the time of the exchange.

Customer accepts full responsibility for its software and data including the appropriate backup thereof.

Repair or replacement of a product during warranty will not extend the original warranty term.

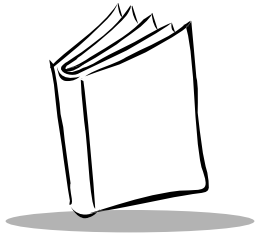
Symbol’s Customer Service organization offers an array of service plans, such as on-site, depot, or phone support, that can be implemented to meet customer’s special operational requirements and are available at a substantial discount during warranty period.

General

Except for the warranties stated above, Symbol disclaims all warranties, express or implied, on products furnished hereunder, including without limitation implied warranties of merchantability and fitness for a particular purpose. The stated express warranties are in lieu of all obligations or liabilities on part of Symbol for damages, including without limitation, special, indirect, or consequential damages arising out of or in connection with the use or performance of the product.

Seller's liability for damages to buyer or others resulting from the use of any product, shall in no way exceed the purchase price of said product, except in instances of injury to persons or property.

Some states (or jurisdictions) do not allow the exclusion or limitation of incidental or consequential damages, so the proceeding exclusion or limitation may not apply to you.



Chapter 1
Spectrum24® Radio Overview

Introduction

This chapter provides an overview of the Spectrum24® radio network's operation, including the network's components and its use of radio to transmit data. The SPT 1740 is just one component of a Spectrum24 network, and is referred to as a Mobile Unit (MU). The SPT 1740 is one of many Spectrum24 802.11 type mobile devices that may belong to a Spectrum24 network. In this manual, the term "SPT 1740" is used instead of the general term "mobile unit."

Spectrum24 is a network of radio devices (i.e., SPT 1740 terminals) and radio access points (APs) that use radio transmissions to communicate with one another. The Spectrum24 radio network is physically connected to a wired Ethernet LAN through the access points. The AP acts as the bridge between the wired and wireless networks. APs receive messages sent on Ethernet and either forward them over the air to the SPT 1740s or ignore the messages (i.e., messages not destined for the SPT 1740s). APs also receive radio messages from SPT 1740s and transfer them onto Ethernet.

SPT 1740 terminals and other Spectrum24 devices use radios to communicate with APs. Applications running on an SPT 1740 send data through software interfaces to the SPT 1740 radio and out to an AP. The AP transfers the data onto Ethernet, then to a host computer running an application that is exchanging messages with the application on the SPT 1740.

See *Access Point and SPT 1740 Functions* on page 2-6 for more information about APs and SPT 1740s.

Radio Communication

SPT 1740s and APs use electromagnetic radio waves to send and receive Spectrum24 data transmissions. Spectrum24 uses the unlicensed band between 2.4 and 2.5 GHz, and uses frequency modulation and frequency hopping to spread its energy over the band.

Spectrum24 uses *frequency modulation* (*FM*) to transmit digital data between the SPT 1740s and the APs. In FM, the digital data signal is superimposed on a carrier signal by a process known as *modulation*. Modulation causes the carrier signal frequency to vary slightly above and below a *center frequency* (this adds the ones and zeros of digital data; see Note below). The modulated signal is transmitted as electromagnetic radio waves. The receiving antenna absorbs the waves as electrical signals. The receiving device attached to the antenna *demodulates* the signal by subtracting the carrier's center frequency from the received signal, leaving the original ones and zeros digital data. This process is shown in Figure 1-1.

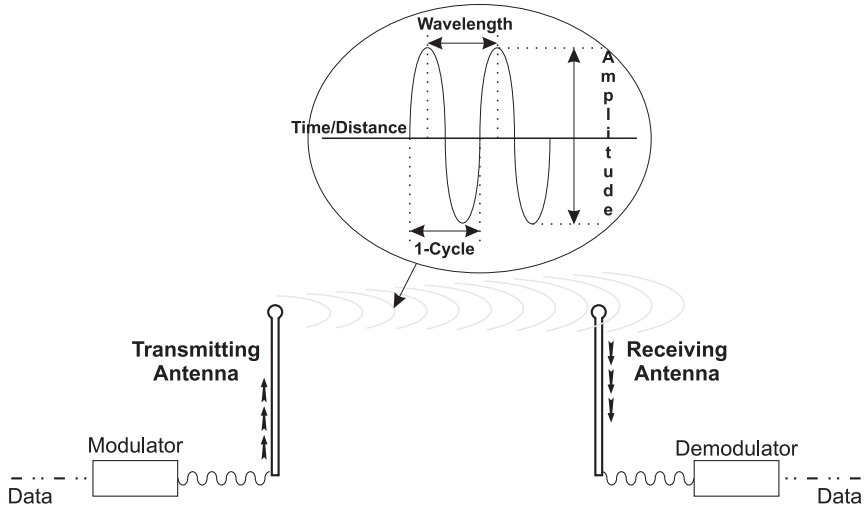


Figure 1-1. Radio Transmission Overview

Note: *The shorter the wavelength, the higher the frequency. FM can briefly shorten the wavelength (increasing the frequency) to modulate a digital one, and briefly lengthen the wavelength (lowering the frequency) to modulate a digital zero. By rapidly shortening and lengthening the wavelength about a center frequency, digital ones and zeros are modulated into the FM signal.*

Spread Spectrum and Frequency Hopping

The Spectrum24 network maximizes data transmission rates and maintains data integrity using frequency hopping *spread spectrum* technology for its radio transmissions. Over time, spread spectrum (also known as *broadband*) communication continually changes (*hops*) the transmitted signal's center frequency. Hopping is analogous to a picket fence, where each picket represents a different center frequency within a specific frequency range. Each Spectrum24 signal has a bandwidth of 1 MHz. The radio transmits and receives for a short period of time on each center frequency. Eventually, every center frequency is used, spreading the transmission over the entire broadband range. Figure 1-2 illustrates a 10-hop sequence over a 1-second period.

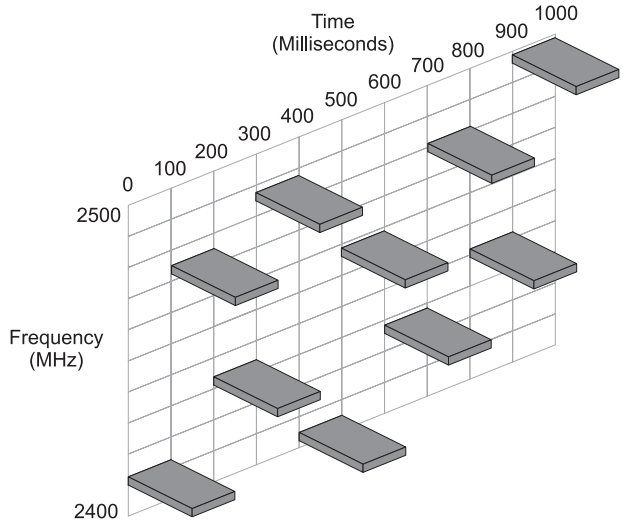


Figure 1-2. Frequency Hopping

When more than one SPT 1740 (or other device) transmits a signal on the same center frequency, the amount of data that is successfully transmitted may be reduced. This multiple transmission is known as *interference*. Frequency hopping reduces the impact of interference. The Spectrum24's carrier signal hops ten times a second, following a predetermined pattern known as the *hop sequence*. The hop sequence is defined in the time synchronization field in the system message packets transmitted by APs as the first transmission of each hop. This is also known as a beacon. Each beacon contains the current hop frequency and the amount of time remaining in the current hop sequence (as well as other data like ESS_ID, see page 2-3). The SPT 1740 radio firmware uses these fields to fine tune its hopping synchronization to the

AP. To communicate, an AP and SPT 1740 in the LAN must synchronize to the same hop sequence.

Hops are always made to a frequency at least six MHz away from the current frequency, which helps avoid interference. Although different devices occasionally hop simultaneously to the same frequency, they will eventually hop to different frequencies. At LAN startup, each AP negotiates with the other APs in the LAN and selects a different hopping sequence. This allows APs to simultaneously transmit (they are on different frequencies) by providing adequate frequency separation and even distribution of the spectrum among all APs. The exceptions are:

- APs operating in a *wireless* AP to AP mode
- APs out of range of each other, so they don't interfere with each other.

Spectrum24 operates in the 2.4-2.5 GHz range; the specific frequency end points, number of hops, and hop sequences depend on the country in which the network is used. These parameters are hard coded into the units' firmware, and are not defined by the user. Spectrum24 does not need to be licensed by the FCC or other regulatory agencies.

Data Rates

Spectrum24 provides a raw throughput rate of 125 KB per second (1 Mbit raw bit rate). With burst protocols like TCP/IP or FTP— with their associated headers and trailers—and the added overhead of the system messages, a single Spectrum24 device like the SPT 1740 can achieve a data payload throughput rate of 50 to 60 KB per second (on a network where that device is the only communicating device). The channel capacity for APs is approximately 80 KB per second for full-sized (1514-byte length) messages. Because of the overhead (headers, trailers) required, small messages are much less efficient at transferring the actual data. The data throughput for small single-byte messages is approximately 9 KB per second.

Cellular Structure

The communication range of an AP is called a *cell*. The AP provides communication and services to all associated SPT 1740s located in that cell. An SPT 1740 identifies a particular AP by its *Basic Service Set Identifier(BSS_ID)* (the AP's MAC address). A single or group of APs with the same user-configurable *Extended Service Set Identifier* (*ESS_ID*) defines the unique Spectrum24 network and coverage area. Before a radio link can be established for the first time, the LAN's ESS_ID must be programmed into the LAN's SPT 1740s and APs. The SPT 1740s and APs can then identify and connect with each other.

Note: *More than one wireless LAN can exist in a single environment by having different ESS_IDs assigned to the devices.*

Figure 1-3 illustrates how adding APs with the same ESS_ID creates more cells and increases the network's coverage area.

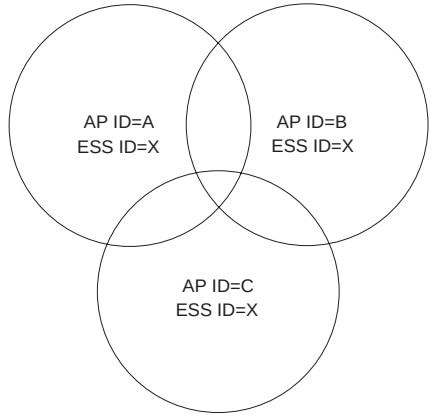


Figure 1-3. Expanding the Coverage Area

CSMA/CA

Spectrum24 APs and SPT 1740s use the *CSMA/CA* (*Carrier Sense Multiple Access/Collision Avoidance*) protocol (Slotted Aloha type) to access and share the wireless medium. Spectrum24 CSMA/CA is similar to Ethernet in that they both distribute available transmit and receive times. Each Spectrum24 device tests (*Carrier Sense*) the airwaves for traffic before transmitting. Because each device communicates in the same medium (*Multiple Access*), collisions may occur when more than one unit detects a clear medium and attempts to transmit at the same time. To reduce collisions in a busy environment, each device has a randomization function (*back off*) to determine when it can transmit data into the wireless medium (*Collision Avoidance*).

Note: *Back-off time (usually in multiples of 50 µs) is based on the system's slot size. Slot size is the network's shortest idle time; it is the time a device needs to determine whether another device has begun transmitting. The device attempting to transmit must listen at least the slot size before proceeding.*

Transmission Process

When a message is ready to be transmitted, the device performs a *Clear Channel Assessment* (*CCA*) to determine if the network, in this case airwaves, is busy. This is an attempt at Collision Avoidance. If the network is not busy (no other transmission is heard), the device transmits its message.

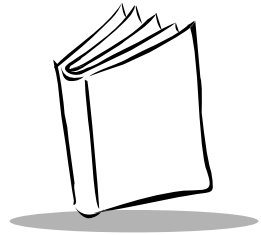
If the network is busy (another transmission is heard) or if a collision occurs, the device backs off for the duration of the other message (when clear air might become available), then repeats the CCA. If the network is still busy, the device increases the back-off time; the device repeats this process until the message can be transmitted.

CCA Function

CCA is a digital hardware function integrated into Spectrum24 radios that detects preamble and data bits over the full dynamic range of the receiver. It discriminates between preamble and data bits and detects the start of a message. The radio encode/decode circuit also provides a digital CCA function used for CSMA/CA processing.

Message Acknowledgements

Spectrum24 uses a positive acknowledgment mechanism (ACK) (shown in Figure 2-1) for data messages and key system messages, such as those that associate an SPT 1740 radio with an AP. If a device doesn't receive an ACK message within a specified time, the device retransmits the message. In Spectrum24, the message is retransmitted after several hops to prevent the message from being lost because of interference on a particular frequency. Therefore, low-level transport functions still operate in congested environments where one or more channels may be blocked by interfering signals.



Chapter 2
Spectrum24® Network Overview

This chapter reviews some of the Spectrum24-specific terminologies, such as associating, roaming, and PSP, and describes some operational characteristics.

Associating

The SPT 1740's radio firmware performs an association to open communications between a specific SPT 1740 and an AP. The SPT 1740s are responsible for starting this network connection. The SPT 1740 must first power on and program its radio with enough information, at a minimum the ESS_ID, to connect to the Spectrum24 radio network (APs). The SPT 1740's radio firmware sends out probe packets with the ESS_ID and listens for an AP with that ESS_ID, then uses the message transfer quality (*received signal strength indication* RSSI, retry attempts, bad message rejection) between the AP and SPT 1740 to evaluate the signal quality of each AP found, and picks the AP with the best quality signal. The SPT 1740 then sends a low-level "connect" request to the AP. If the AP is associated with fewer than 128 radio devices (its maximum *load*), it sends an "accept" message to the SPT 1740. This process is known as *associating.* Once it is associated with an AP, the SPT 1740 can establish a standard network connection over Ethernet. The SPT 1740 can use its own IP address or get one from a DHCP server. It can locate a host, log in, and begin transferring data.

The radio firmware in Spectrum24 SPT 1740s tracks and can pass the association status to the applications (see *S24GetAssociationStatus* on page 4-15). The SPT 1740 makes its association status available so that applications can be warned when the SPT 1740 may have been taken out of range of all APs or when the AP network may be non-functional. However, a brief time period exists between when an application can check the association status and the time a message is sent through the transport layer. The association may be lost during this time period. If this occurs, connection problems may result, depending on how well the

application and the different transport layers treat the inability to communicate. This small time period should not prevent the application from protecting itself by checking the association status.

Roaming

The SPT 1740 radio associates with an AP until the SPT 1740 radio needs to switch cells. The process of switching cells is called *roaming* and is initiated automatically by the SPT 1740 radio firmware. Roaming occurs when:

- An unassociated SPT 1740 tries to associate or reassociate with an available AP.
- The RSSI of the AP currently associated with the SPT 1740 is much lower than that of another available AP.
- The ratio of transmitted bytes to attempted-transmitted bytes sent by an SPT 1740 falls below a specified threshold.
- The SPT 1740 detects an imbalance in the number of SPT 1740s associated with the APs in the coverage area, and roams to a less-loaded AP.

Roaming is transparent and virtually instantaneous to high-level applications.

Searching

SPT 1740 radios perform preemptive roaming by intermittently *searching* for APs and then associating with the best available AP.

Searching is a process in which the SPT 1740 radio periodically sends out probe messages containing its ESS_ID—and, if used, the specific APs BSS_ID—on all frequencies defined by the country code in the firmware. If the SPT 1740 radio does not receive a response from an AP, it tries the next frequency.

APs respond to the probe messages with:

- their hopping sequences
- the current hopping frequency set
- the length of time until the end of the current hop (*hop interval*).

From these responses, each SPT 1740 radio maintains information in an *AP Table* on all currently known APs in the network.

Note: *Searches are performed every 30 seconds during normal operation. The SPT 1740 radio can probe all U.S. frequencies (up to 79) in one hop cycle (100 ms) or less.*

An SPT 1740 radio continues searching and associating for as long as it is active. This allows the SPT 1740 radio to communicate with new APs and stop communicating with any that are out of range or deactivated.

ESSID and BSSID

ESSID and BSSID are names used by SPT 1740 (mobile Spectrum24® units) radios to connect with 802.11 protocol enabled access points using the same names (the SPT 1740 is 802.11 enabled). ESSID is the local radio network's name and is used by both access points and SPT 1740s to identify members of the network. An ESSID name is mandatory for 802.11 operation. Only units that have the proper ESSID are allowed on the network. ESSID is valid throughout the RF network, so SPT 1740's that roam can stay connected to the network throughout an installation.

Note: *ESSID is a character string of up to 32 bytes in length. Do not programmatically set a string to non-printable characters, because it will be hard to decode that string later if it is forgotten.*

BSSID is the 6-byte MAC address of a particular access point. BSSID can limit network accessibility to specific access points in the attempt to load balance a LAN, or to keep specific operations in one area. Note that each SPT 1740 or other Spectrum24 MUs also has its own BSSID.

Two mutually exclusive attributes, the mandatory AP BSSID and the preferred AP BSSID, can be set in an SPT 1740 to constrain the possible AP BSSIDs to which the SPT 1740 can associate. If a mandatory BSSID is set, the SPT 1740 only associates (RF connect) with that specific access point. If the SPT 1740 goes out-of-range of the mandatory access point, but is still in range of other access points, the SPT 1740 does not associate to the network, even if ESSIDs match, until it is brought back in range of the mandatory access point.

Setting the preferred BSSID forces a SPT 1740 to associate to a specific access point as long as that access point can be heard by the SPT 1740. Normally, a SPT 1740 automatically roams between access points based on the signal quality of the access points it can hear. The preferred BSSID stops the automated roaming and as long as the access point can be heard,

the SPT 1740 maintains contact, no matter how bad the connection. However, if the SPT 1740 goes out-of-range of that specific access point, it associates with other access points with the proper ESSID.

Message Formats

The Spectrum24 network uses 6-byte, hexadecimal IEEE addresses (known as *MAC addresses*) to identify each SPT 1740 radio and AP. A MAC address is a unique number for every device in the network that comes programmed in each network component.

SPT 1740s and APs use data and system messages to communicate. Data messages convey user packets. System messages include:

- packets used as probes for identifying APs that are within range
- beacon and traffic indicators
- polls to obtain user data
- *ACKs* (*acknowledgments*).

Each system and data message transmitted between an SPT 1740 and an AP includes:

- preamble
- start frame delimiter
- protected length field
- message header
- data
- 32-bit *cyclical redundancy check (CRC)* field.

Message Fragmentation and Reassembly

The Spectrum24 driver supports Ethernet messages up to 1514 bytes in length. The firmware in Spectrum24 APs and SPT 1740 radios divides each Ethernet frame and attaches a 12-byte prefix containing the address header to each fragment, as shown in Figure 2-1. The address header includes the destination and source MAC addresses. Each fragment is a frame that is up to 548 bytes in length (560 bytes with the address header).

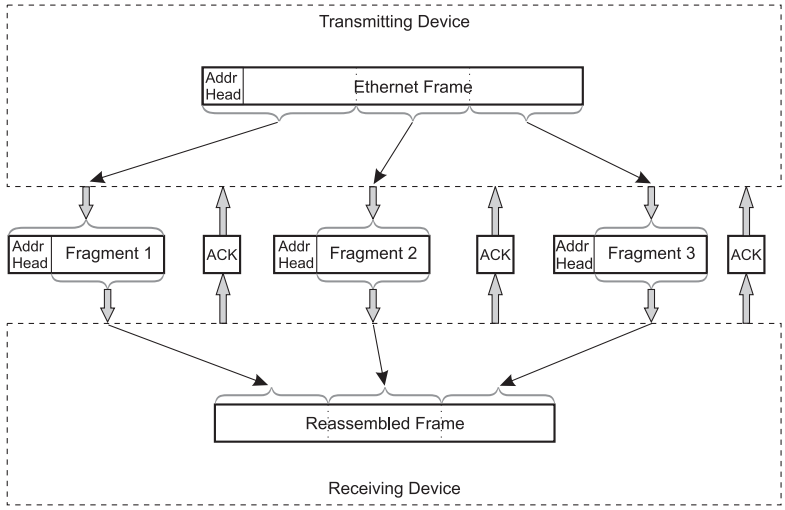


Figure 2-1. Information Packet Fragmentation and Reassembly

The transmitting device (AP or SPT 1740) sends the first fragment to the receiving device (SPT 1740 or AP). If the receiving device responds with an *acknowledgement* (*ACK*) message, the transmitting device sends the next fragment. This process is repeated until all the fragments have been sent. The receiving device removes the address headers from each fragment and reassembles the message frame.

In Spectrum24, transmitting a fragment and receiving an ACK message takes approximately 6 ms.

Message Processing

In addition to receiving SPT 1740 messages, the AP firmware maintains a queue of messages to be transmitted to the SPT 1740s. Messages are transmitted in the order in which they were added to the queue. Broadcast and normal messages have separate queues.

When not transmitting, the AP operates in receive mode. It verifies all validated (CRC'd) packets received and accepts only those packets that have a destination MAC address matching its own.

On the SPT 1740, the application may configure the radio firmware to accept broadcast packets in addition to the normal directed message packets.

Note: *The SPT 1740 stack does not support multicast messaging.*

Access Point and SPT 1740 Functions

At a lower level of radio communications, APs and SPT 1740s must follow a set of data exchange rules. They must:

- negotiate which radios communicate with one another
- configure themselves to avoid interfering with each other
- exchange messages in a timely manner.

In Spectrum24 at one level, the AP can be considered a controlling unit and SPT 1740s considered independent units.

The AP controls the Spectrum24 environment. As the controlling unit, it:

- determines the hopping sequence
- transmits control packets
- maintains message queues for all of its associated SPT 1740s
- schedules and performs network broadcasts
- informs SPT 1740s when they have queued data in the AP and should request data transfer.

As long as an SPT 1740 is associated with an AP, it must obey the rules of that AP cell. Acting as independent units, SPT 1740s must:

- listen to AP control messages and mimic that AP's hopping sequence
- avoid transmission collisions
- listen for network broadcasts from their associated AP.

SPT 1740s are responsible for maintaining a quality connection with APs. SPT 1740s monitor their environment and track the signal quality of APs in their local area. When the quality of the association with the current AP falls below that of another AP, the SPT 1740 begins communicating with the other AP (*reassociate*).

AP Control Messages (System Messages)

APs periodically transmit control information to the SPT 1740s, including the frequency hopping sequence for the AP. SPT 1740s program their radios to follow the hops taken by the

AP. This allows the SPT 1740s to continually hear the AP as it moves from one frequency to another.

Another part of AP control information sent specifies which SPT 1740s have a message queued in the AP. The AP transmits this control information at the beginning of each frequency hop. This allows the SPT 1740s to wake up and turn their radios on (*receive*) for only a very short time during each frequency hop. If no data is queued in the AP for the SPT 1740, the SPT 1740 can turn off its radio and revert to a low-power mode until the next frequency hop. For battery-operated devices, a great deal of power is saved. This method of awakening at predetermined times is called *power save polling* (*PSP*). More information about PSP can be found in *Wired and Wireless Network Connections* on page 2-8.

A special control message known as a *delivery traffic indicator message (DTIM)* is transmitted by the AP every second. DTIMs synchronize all SPT 1740s connected to an AP. Therefore, the AP can send the broadcast messages to all SPT 1740s simultaneously. As the AP receives broadcast messages over Ethernet, it saves them (up to the latest 10) in a queue of their own. When the next DTIM occurs, the AP transmits all broadcast messages to the SPT 1740s.

Spectrum24 Radio Power Management

Radio Suspend Mode

This mode is equivalent to turning off the SPT 1740. In suspend mode, the SPT 1740's CPU is running but the analog radio is powered off, so the SPT 1740 receives no messages. The SPT 1740 maintains state information (associated AP, hop sequence, power mode, IP address, etc.) so that upon waking it resumes its previous state, and the radio resumes where it left off. This mode occurs as a part of each PSP mode wake up or by manual power up of the SPT 1740 radio.

Beacon Algorithms-Power Save Polling (PSP)

Access points transmit a signal every 100 milliseconds. This is called a beacon, which tells SPT 1740s about RF message traffic that the access point has pending for SPT 1740s. This lets SPT 1740s know when they have data to be read from the access point.

SPT 1740s can be programmed to listen at every beacon, or to skip a number of beacons. This Beacon Algorithm is another name for PSP. Spectrum24® SPT 1740 Beacon Algorithms are numbered from 1 to 10 with a special Beacon Algorithm that is numbered 11.

Algorithm 1 listens to every beacon, and consumes the most power. Beacon Algorithm 2 listens every 2 beacons. In idle mode, no data to transfer, Beacon Algorithm 2 consumes half the radio power of Beacon Algorithm 1. Beacon Algorithm 3 listens to every third beacon, and so on, through Beacon Algorithm 10. The more beacons that are skipped, the less the idle time power consumption. Beacon Algorithm 10 is the maximum number, and it listens to every 10th beacon, or once per second. This setting allows the least idle time power consumption. However, when the Beacon Algorithm is fixed, data is only transferred when the SPT 1740 listens for the beacon. This provides a built-in delay in network communication. The selection of beacon algorithms generates a trade-off between battery consumption, response time, and throughput.

The special Beacon Algorithm 11 is dynamic; in idle mode (no data transfer) it can be set to operate with a maximum skipped beacon count between 1 and 10. It remains at this maximum setting until a beacon indicates the SPT 1740 has data in the access point. The SPT 1740 radio then transitions to a minimum skipped beacon count between 1 and 10 and remains at that rate until all data is transferred from the access point. After data transfer, the SPT 1740 radio returns to the maximum count and stays there until the access point indicates more data awaits the SPT 1740. Beacon Algorithm 11 is usually set to minimum 1 and maximum 10. However, as long as minimum is less than maximum, the settings can be any number between 1 and 10. The selection of Beacon Algorithm 11 with Min=1 and Max=10 provides the least power consumption with the best throughput and is the default SPT 1740 setting.

Data Transfer in PSP Mode

While in *power save polling* (*PSP*) mode, SPT 1740 radios sync up and receive beacon messages from their associated APs. Beacon messages are sent at the beginning of every frequency hop (every 100 ms) and contain information about which SPT 1740s have data queued in the AP. When a beacon message includes an SPT 1740's MAC address, that SPT 1740 radio generates a poll message. The AP responds with the message fragments, each of which the SPT 1740 radio acknowledges (see Figure 2-1). To prevent poll message congestion, all units randomly back off from the end of the beacon message. This allows each SPT 1740 to be taken care of one at a time without all SPT 1740 radios simultaneously requesting messages.

Wired and Wireless Network Connections

At the highest level of network communications, applications exchange data messages. These messages are often sent from one computer to another, along with system control messages. The transfer medium can be a physical connection (wire, fiber), open-air (radio, IR), or a combination of the two.

Every device in a totally wired network can always communicate with all other devices in the network, as long as the devices are turned on and the wiring connecting them is intact. Most standard communication protocol stacks assume that the messages always get through because they are written to depend on the intact wired connection. When a message is sent, the sending device starts a time-out counter and waits for a reply from the receiving device. If the time-out expires, the original message can be retransmitted and the time-out restarted. This retransmit process could continue indefinitely, but it is usually performed fewer than five times. If the last retransmit times out, an error condition is returned to the application indicating that the connection was lost. At that point, an error message is usually displayed and the application is aborted because the wire is assumed to be broken.

In a wireless network, where devices can be moved around, the device may not be able to maintain the connection to the network. This could be a normal occurrence, as when a device is taken from one building to another and is temporarily out of the network's range. Ideally, the connection between the application in the SPT 1740 and the application in the network's host computer remains intact. When the SPT 1740 gets back in range, the operation should resume. However, if the protocol stack (written for a wired network) times out because it does not get an ACK message from the receiving device, the connection is usually assumed to be broken; an error code is returned to the application, and the application is aborted. In a wireless environment, this should not happen. Software in a wireless environment should at least detect an out-of-range condition and not push messages onto the transport layer until the SPT 1740 is back in range. If detection is not possible, applications should be designed to either recover from the out-of-range condition (rather than abort) or use transport layers that are tolerant of a protracted inability to communicate. If tolerant transport layers are not available, set the time-out or retransmit parameters to their maximum so that the occasional out-of-range condition does not abort the application as soon as this condition starts.

AP To SPT 1740 To AP Communication

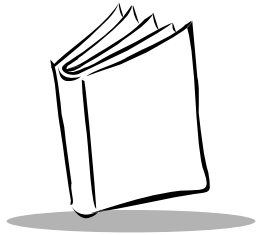
One of the differences between wired and wireless connections is how different message classes are handled. Normal (directed) data message exchanges are initiated by the SPT 1740s, and broadcast message transmits are initiated by the APs.

For AP-to-SPT 1740 transfers, normal messages are only transferred from the AP when the SPT 1740 requests the AP to send the data. Since each AP frequency hop sends a notification of queued messages, SPT 1740s that need to retrieve messages must share the available time and retrieve their messages in an orderly fashion. Because the AP does not know when to transmit to each SPT 1740, the SPT 1740s initiate the transfer. The SPT 1740s use a semi-random back-off to determine which unit goes first. The SPT 1740 that selects the least amount of back-off time goes first. That SPT 1740 listens for clear air and sends a request to the AP to send its data. The AP immediately (within 50 microseconds) complies and sends

the first message to the target SPT 1740. When more than one message is to be sent to the same SPT 1740, a continuation bit is set in each message header. The SPT 1740 receives messages until the continuation bit is no longer set.

The remaining SPT 1740s repeat the process until all the SPT 1740s have requested all their data and the AP is emptied. If all the data cannot be emptied before the next frequency hop, the SPT 1740s that still need data wait for the next hop to begin, then start the process again.

In Spectrum24, the AP controls the transmission of broadcast messages. The AP sends DTIMs at predetermined times. SPT 1740s sync with the AP's DTIM and listen for the broadcasts. This is different from the wired network, where all devices are listening all the time and all broadcasts can be sent as soon as they are received. Broadcast messages are *UDP* (*User Datagram Protocol*) class messages, and are not guaranteed to be delivered. The AP has limited memory and can not store an unlimited number of broadcast messages. The AP keeps only the last 10 broadcast messages, so some broadcast messages may be discarded. In most networks, lost broadcasts are expected, so the broadcasts are retransmitted by the sender.



Chapter 3
SPT 1740 Spectrum24 Introduction

The Spectrum24® Driver Extensions shared library contains all of the public functions implemented in the low-level driver software. Using these functions, an application can control some of the SPT 1740's radio behavior and gather information about the SPT 1740's radio settings. However, applications are not required to call any of the functions in this library. By default, the SPT 1740 initializes and connects to the Spectrum24 network based on the preferences set in the SPT 1740 Network (Spectrum24) Preference Panel. These extension functions let the developer configure such items in the SPT 1740 radio as preferred access points. They also allow the developer to read items like the SPT 1740's driver version number, access point association table, and the SPT 1740's MAC layer access point connection association status (which is the SPT 1740's current radio connection status with an access point).

Figure 1-1 shows how an application interfaces mainly with the SPT 1740's PalmOS NetLib API functions and partially, depending on the complexity of the application, with the SPT 1740 Spectrum24® Driver Extensions library. The Spectrum24 Driver Extensions Library is not required at a Ratio application. The functions below the Spectrum24® Driver Extensions layer are hidden from the application.

Note: *NetLib is a wrapper around the IP stack and is used by SPT 1740 applications to send and receive data. For a detailed description of NetLib, see the standard Palm Programming documentation.*

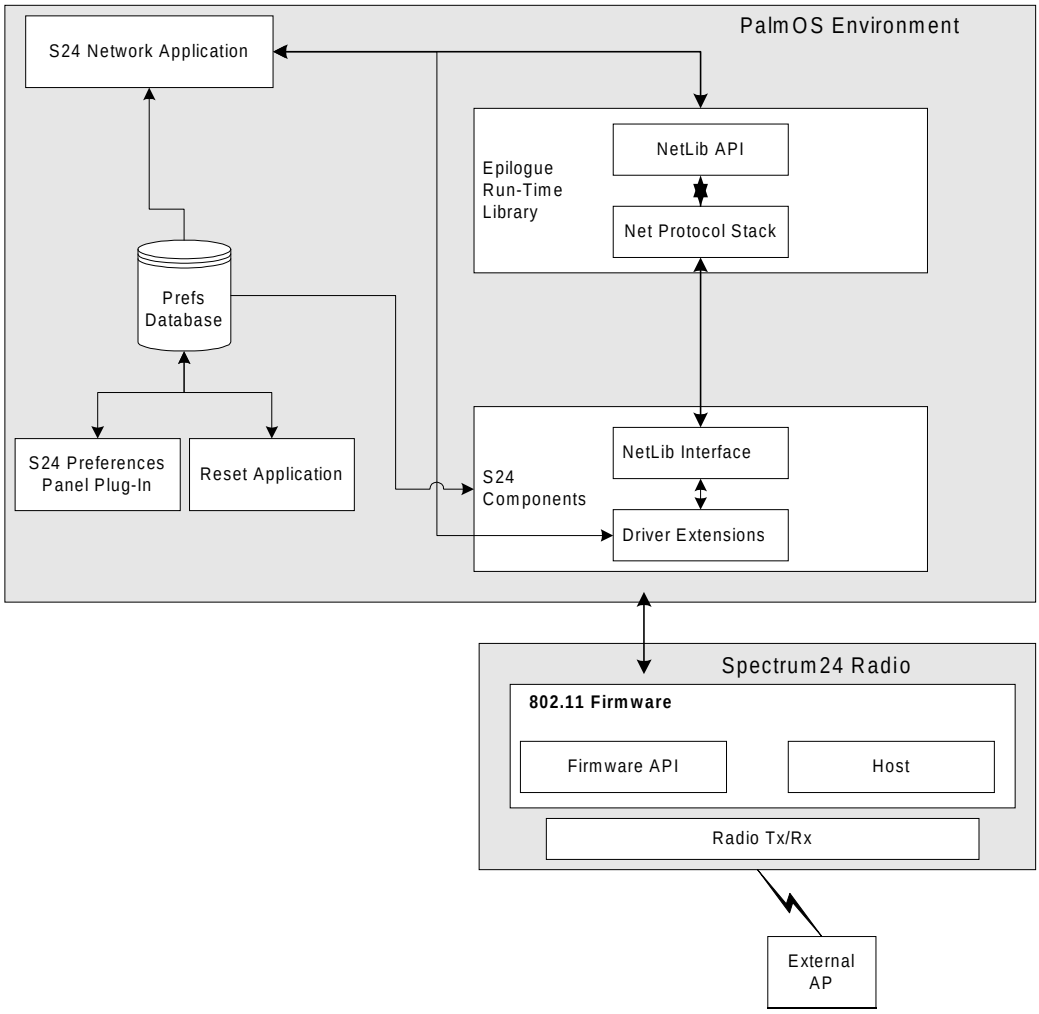


Figure 3-1. Spectrum24 Library Interfaces

Spectrum24/NetLib Design and Implementation Considerations

Feature Limitations for this Model

The following list describes features not supported:

- The SPT 1740 supports only the IEEE 802.11 protocol.
- 802.1 header message formats are not supported.
- Mobile IP is not supported.
- Multicast messages are not supported.
- Embedded Spectrum24 encryption is not supported.
- The maximum Ethernet packet size is 1514 bytes (1472 data bytes).
- Read/write of Spectrum24 radio flash is not supported.

General Application Design Considerations

Applications running on a local radio network (LRN) must be able to handle the following adverse conditions:

- Long delays caused by congestion (other Spectrum24 communications) of the LRN and by the SPT 1740 roaming in and out of the LRN's range
- Long delays in the wired network responses
- High TCP retry timeouts, which might cause the socket to close and lose the network connection
- Power failure caused by the SPT 1740's battery being completely drained or removed
- Operation in SPT 1740 power mode switching environments (auto-power-off system timeout) and SPT 1740 radio (PSP, sleep, and VCC off) conditions
- Slow connection or re-connection to the host application server
- RF transmissions being locked out by aggressive user scanning operations.

Turning the SPT 1740 Radio Interface On and Off

When an application calls the NetLibOpen command (see the standard Palm Programming documentation), NetLib interfaces to the Spectrum24 Radio Driver which powers up the radio, and initializes and configures the radio for RF transmissions. The Spectrum24 driver powers off the radio when an application calls the NetLibClose command.

NetLib Interface UI

When initializing, the network interface on the SPT 1740 displays three windows that contain the following status information about the RF network connection:

- associating with ESSID <essid>
- binding (getting an IP address from the DHCP server)
- connected.

One key status indicator is association with an access point. Your application can determine through an API (*S24GetAssociationStatus* on page 4-15) whether or not the SPT 1740 is associated to an access point. Ideally, your application should display a dialog or other indication when the SPT 1740 radio loses association with an AP for a period of time.

Note: *Situations where the UI does not go beyond the associating stage usually imply a failure to connect to an access point. Stopping at the binding stage indicates a problem negotiating an IP address through the DHCP process.*

Network Connections

Applications use the SPT 1740's TCP socket connection, accessed through standard Palm NetLib calls, to communicate over the LRN. Applications can also use file transfer protocol (FTP) for file transfers over the network; however, trivial file transfer protocol (TFTP) is not recommended, since delivery is not guaranteed.

TCP Retry Timer

Applications that use a TCP connection to communicate over the LRN can experience a high TCP retry rate. This may result in long network delays, which may cause a TCP socket to time out and lose the network connection. To accommodate, and in some cases solve this problem, use *S24SetTcpResendTimeout* on page 4-44 to adjust the SPT 1740's TCP socket time out timer.

Roaming/Unassociation Condition

A SPT 1740 unassociates with an access point when the SPT 1740 radio is out of range or is in the process of associating with another access point. If an application tries to transfer data while the radio is unassociated with an access point, the data and perhaps the network connection may be lost. The application should not attempt to transmit data until the association is reacquired.

Applications call *S24GetAssociationStatus* on page 4-15 to get the current AP association status before sending data through NetLib calls.

Note: *A user interface should display if the SPT 1740 is unassociated for more than a few seconds to notify the operator to return to the network and display the reason the application is on hold. Do not display the out-of-range UI on the first unassociation; give the SPT 1740 a chance to re-associate.*

Scanner Priority Over RF

In scan-enabled applications, pressing the SPT 1740's bar code scanner button immediately activates the unit's laser and causes the target data to be decoded. Immediately prior to enabling the scanner, the SPT 1740's radio is put into a no transmit state (Tx Inhibit), in which all packet transmissions are stored, received data is not ACKed, and further transmission requests are ignored until the scan is completed. No radio transmissions are allowed. After the scan completes, the radio is taken out of the Tx Inhibit state.

Scan-enabled applications should not allow the user to perform rapid scans that inhibit RF transmissions for prolonged periods. This may cause the SPT 1740 radio's association with the AP to be lost, the connection to the network to be dropped, and data to be lost. A well-designed scan/RF application should pace the application's network activity to prevent such frequent scanning that the radio is locked from transmitting for an extended period.

Power Management

Power management functionality is integrated into the radio driver and network interface. The power management implementation integrates features provided in the SPT 1740 Palm OS (auto-off timer, power key on/off, dead battery level, and low battery level), SPT 1740 Spectrum24 radio firmware (PSP, sleep, host power-down), and the SPT 1740 Spectrum24 hardware interface controller application-specific integrated circuit (ASIC) (VCC on/off).

To maximize battery life on the SPT 1740, the Spectrum24 driver automatically tries to keep the radio operating in the most energy efficient manner by operating in one of the following:

- Normal operation (Stage 0): Setting the SPT 1740 radio power mode to PSP-11 to handle network traffic.
- Napping (Stage 1): Setting the SPT 1740 radio power mode to the lowest power mode while in stand-by waiting for network traffic. The SPT 1740 transitions to this mode only when the Palm OS autooff timer goes off and puts the unit into sleep mode.

- VCC off (Stage 2): Removing power to the SPT 1740 radio. The SPT 1740 transitions into this mode when the unit is not used for an extended period, when the unit is in normal operation and the user pushes the power key, or when various low or no battery conditions are detected.

When the system recovers from a power-down, the Spectrum24 driver resets the SPT 1740 radio to its normal operating mode, PSP-11.

Napping (Stage 1)

The Spectrum24 driver puts the SPT 1740 radio in napping mode when the system auto-off timer expires. While in napping (host power-down) mode, the SPT 1740 radio operates in PSP-10 power mode until the Stage 1 time-out timer expires. While in this mode, the SPT 1740 radio remains associated with an AP and can still receive network traffic. The SPT 1740 radio interrupts the SPT 1740 main processor if it receives a directed or broadcast (if enabled) packet and thus awakens the SPT 1740 to resume operation.

When the system resumes normal operation, the Spectrum24 driver clears the Stage 1 time-out timer and restores the SPT 1740 radio's power mode to PSP-11 or to the beacon algorithm preferences set by the application.

Use the S24SetPreference function call to configure the Stage 1 time-out timer. The default time for Stage 1 is 10 minutes.

VCC Off (Stage 2)

If the user performs a reset of the portable unit, removes the batteries, or the Stage 1 timeout expires, VCC power to the SPT 1740 radio is removed. This helps stop system memory from being corrupted by preventing the SPT 1740 radio from draining the backup battery.

Note: *It is the application's responsibility to re-establish the connections to the host. Socket connections must be re-established through NetLib using the NetLibConnectionRefresh function call.*

Palm OS Power Mode

Auto-Off Timer

When the SPT 1740's auto-off timer expires, the device enters a hardware napping state. While in this state, the SPT 1740's manual operation is suspended, but VCC power to external devices such as the Spectrum24 SPT 1740 radio is still operational. The auto-off time

can be set through the General Preferences panel or through a Palm OS API call (**SysSetAutoOffTime()**).

When the SPT 1740 initially goes to sleep (normal Palm timeout), NetLib remains open and the Spectrum24 radio remains on to keep the network connections alive. When future network communications for the SPT 1740 are received, the SPT 1740 will immediately awaken and resume operations; applications don't have to restart NetLib and re-establish TCP connections upon wakeup.

A programmable, full-device shutdown timeout is available to conserve batteries when the SPT 1740 is not used for an extended period. For more information see *S24SetPreference* on page 4-40.

Power Key Off

When the power off key is pressed during normal operation, the Spectrum24 driver enters Stage 2 VCC off power mode and power is removed from the radio.

Power Key On

The Spectrum24 driver resumes the SPT 1740 radio and restores the SPT 1740 radio to PSP-11 power mode (or the PSP mode programmed by the application). Remember, the application must re-establish its host connection through the NetLib call *NetLibConnectionRefresh*.

Low Battery Level

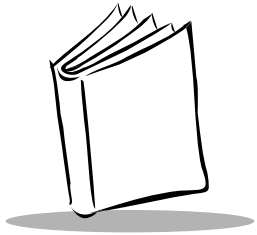
The Spectrum24 driver receives an event from the SPT 1740 Palm OS and enters Stage 2 VCC off power mode.

Dead Battery Level

The Spectrum24 driver receives an event from the SPT 1740 Palm OS and enters Stage 2 VCC off power mode.

NetLibConnectionRefresh

Power-off conditions cause the Mac-layer (radio) to go down while NetLib stays open. Calls to NetLibConnectionRefresh ensure that the Mac-layer is up. If an application has not been closed and NetLib is still open, the application should call NetLibConnectionRefresh before each socket operation and Spectrum24 API call. This causes the Spectrum24 driver to either restart or resume the SPT 1740 radio (depending on the current power-down mode) and set the SPT 1740 radio to power mode PSP-11. It also causes reassociation to an AP.



Chapter 4
Spectrum24 Radio Library Function Calls (API)

The Spectrum24 Radio library programming interface contains a set of public driver extension functions. Applications are not required to call any of the functions in this library. By default, the SPT 1740 radio initializes and connects to the Spectrum24 network based on the entries made in Spectrum24 Preferences. Applications call these functions by linking with the library.

Driver Extensions Shared Library

To link the Spectrum24 driver extensions into your application, include the SPT1740.lib library in your application's project files. This library is a collection of interface functions that insulates the application from some details of using the SPT 1740's shared libraries. For example, your application will not have to maintain the shared library reference number typically used to call a function contained in a shared library. The Spectrum24 driver interface functions take care of this for the application. This approach adds a calling layer, but makes porting existing code to the SPT 1740 easier and reduces the chance of bugs introduced by porting the application.

Returned Status Definitions

The function calls in this chapter may return one of the status codes described in Table 4-1.

Table 4-1. Returned Status Codes

Status Code	Definition
s24ErrNone	No errors were detected.
s24ErrParam	Invalid input parameter was detected.
s24ErrNotOpen	Driver was not able to open a database or library.
s24ErrStillOpen	Driver failed to close the shared library.
s24ErrMemory	Driver detected a memory allocation problem.
s24ErrCardNotReady	The SPT 1740 radio is not responding properly.
s24ErrIOModeFailed	The SPT 1740 radio initialization failed.
s24ErrBufferLen	Caller supplied buffer is too short for the buffer to be returned.
s24ErrTimeout	The SPT 1740 radio is not responding properly.
s24ErrInvFWVersion	The version of the firmware loaded into the radio adapter is incompatible with the SPT1740. If you do not have the proper firmware, contact Symbol Customer Support.

Spectrum24 Commands

Table 4-2 lists the Spectrum24 commands described in this chapter.

Table 4-2. Spectrum24 Driver Extension Commands

PalmIsSPT1740

Purpose Allows application to ensure that it is running on an SPT1740.

Prototype

```
Boolean PalmIsSPT1740 ( void);
```

Parameters None.

Returned Status True = Unit is a SPT1740.

False = Unit is not a SPT1740.

S24GetAdapterBSSID

Purpose Retrieves the SPT 1740 radio's BSS_ID setting (MAC address of the SPT 1740).

Prototype

```
s24Err S24GetAdapterBSSID (
                    BytePtr pszBssID, Word wBuffLen);
```

Parameters

-> **pszBssID**	A pointer to a string array where the BSS_ID is stored.
-> **wBuffLen**	ASCII byte string length of the BSS_ID.

Returned Status Zero = No errors retrieving the BSS_ID.

Non-zero = Error getting the BSS_ID. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments This function returns a pointer to the BSS_ID string.

See Also

S24GetMandatoryBSSID
S24GetPreferredBSSID
S24SetMandatoryBSSID
S24SetPreferredBSSID

Code Sample for S24GetAdapterBSSID

```
s24Err  s24err;
Byte    byBSSID[S24_BSS_ID_LENGTH + 1];   // BssID bytestring
Char    szReadableBSSID[S24_BSS_ID_LENGTH*2+6]; // BssID in human-readable form

printf("\nAbout to S24GetAdapterBSSID...");

s24err = S24GetAdapterBSSID( byBSSID, S24_BSS_ID_LENGTH + 1 );

if (s24err == s24ErrNone)
{
    printf(" success -");
    FormatIEEEAddress( szReadableBSSID, byBSSID, S24_BSS_ID_LENGTH*2+6 );
    printf(" BSSID: %s", szReadableBSSID);
}
else
    printf(" Failure, S24Err #%d", s24err);
```

Note: *Listing for FormatIEEEAddress() included in this document.*

Code Samples for Manipulating BSSID

```
//
//  FUNCTION: UnformatIEEEAddress( CharPtr pszAddr, BytePtr pbMacAddr, Word
wBuffLen)
//
//  DESCRIPTION: Convert a BSSID string like "00:A0:F8:00:22:F8" into its
//              byte equivalent,
//              0x00A0F80022F8.
//
//  PARAMETERS: pszAddrThe string representation, i.e. "00:A0:F8:00:22:F8"
//              pbMacAddr Buffer into which actal bytes are stored
//              wBuffLenLength of pbMacAddr, must be 6 bytes or more
//
void UnformatIEEEAddress( CharPtr pszAddr, BytePtr pbMacAddr, Word wBuffLen)
{
    int i;

    if (wBuffLen < S24_BSS_ID_LENGTH)
        return;
```

```
    // We want pszAddr to be read from left-to-right, and pbMacAddr to be filled in
    // from right-to-left
    // For each of the 6 bytes in a BSSID, convert from ASCII to hex:

    for (i = 0; i < S24_BSS_ID_LENGTH; i++)
    {
        // We fill this in from left to right:
        pbMacAddr[i] = Ascii2Hex(pszAddr);
        pszAddr += 3;// Moves from "00:AF" to "AF"
    }
}

Int FormatIEEEAddress( CharPtr pszAddr, BytePtr pbMacAddr, Word wBuffLen )
{
    int inx;
    char ch;

    if( wBuffLen < S24_BSS_ID_LENGTH*2 + 6 )
        return 0;

    for( inx=0; inx<S24_BSS_ID_LENGTH; inx++ )
    {
        ch = Hex2Ascii( (Byte)(pbMacAddr[inx]>>4) );
        *pszAddr++ = ch;
        ch = Hex2Ascii( pbMacAddr[inx] );
        *pszAddr++ = ch;

        if( inx != S24_BSS_ID_LENGTH-1 )
            *pszAddr++ = ':';
        else
            *pszAddr = '\0';
    }

    return inx-1;
}
```

S24GetAdapterESSID

Purpose Retrieves the SPT 1740 radio's ESS_ID setting.

Prototype

```
s24Err S24GetAdapterESSID (
                    CharPtr pszEssID, Word wBuffLen);
```

Parameters

->	**pszEssID**	A pointer to a string array where the ESS_ID is stored.
->	**wBuffLen**	ASCII byte string length of the ESS_ID (variable length of up to 32 bytes).

Returned Status Zero = No errors getting the ESS_ID.

Non-zero = Error getting the ESS_ID. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments The ESS_ID is a name of up to 32 bytes which identifies the network that the SPT 1740 radio is a member of. The ESS_ID can be entered in the preferences panel and saved in the network (Spectrum24) preferences database. It can also be set by the **S24SetAdapterESSID** function call.

This function returns a pointer to the ESS_ID string.

See Also **S24SetAdapterESSID**

Code Sample for S24GetAdapterESSID

```
s24Err s24err;
Char    szESSID [S24_ESS_ID_LENGTH];

printf("\nAbout to S24GetAdapterESSID...");

s24err = S24GetAdapterESSID( szESSID, S24_ESS_ID_LENGTH + 1 );

if (s24err == s24ErrNone)
{
    printf(" success -");
    printf(" ESSID: %s", szESSID );
}
else
    printf(" Failure, S24Err #%d", s24err);
```

S24GetAPBSSID

Purpose Retrieves the BSS_ID (MAC address) of the access point to which the SPT1740 is associated.

Prototype

```
s24Err S24GetAPBSSID (BytePtr pszBssID, Word
                      wBuffLen);
```

Parameters

-> **pszBssID**	A pointer to a string array where the BSS_ID is to be stored.
-> **wBuffLen**	Size of the BSS_ID buffer. Must be at least as long as **S24_MAC_ADDR_LENGTH**.

Returned Status Zero = No errors retrieving the BSS_ID.

Non-zero = Error getting the BSS_ID. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments User should check association status before making this call.

See Also

S24GetMandatoryBSSID
S24GetPreferredBSSID
S24SetMandatoryBSSID
S24SetPreferredBSSID
S24GetAssociationStatus

Code Sample to Get BSSID of AP to which SPT 1740 Is Associated

```
s24Err s24err;
Byte    byBSSID [S24_BSS_ID_LENGTH ];// BssID bytestring
Char    szReadableBSSID[S24_BSS_ID_LENGTH * 2 + 6];

printf("\nAbout to S24GetAdapterBSSID...");

s24err = S24GetAPBSSID( byBSSID, S24_BSS_ID_LENGTH);

if (s24err == s24ErrNone)
{
    printf(" success -");
    FormatIEEEAddress( szReadableBSSID, byBSSID, S24_BSS_ID_LENGTH*2+6 );
    printf(" AP's BSSID: %s", szReadableBSSID);
}
else
    printf(" Failure, S24Err #%d", s24err);
```

Note: *FormatIEEEAddress() is illustrated elsewhere in this section.*

S24GetAPTable

Purpose Retrieves a table that contains the BSSIDs of all current APs that the SPT 1740 can contact.

Prototype

```
s24Err S24GetAPTable (
                S24_AP_TABLE4 **ppApTable,
                Word wNumEntries, Word *
                wNumReturned);
```

Parameters

-> **ppApTable**	Returned buffer containing AP table entries. Structure defined in s24statsdef.h.
-> **wNumEntries**	Maximum number of entries that **ppApTable** can contain.
-> **wNumReturned**	Number of entries actually returned.

Returned Status Zero = No errors.

Non-zero = Error getting the AP table. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

NetLibOpen must be called before this function can provide useful information.

Code Sample for S24GetAPTable()

```
#include "S24Api.h"

{
    //////////////////////////
    // S24GetAPTable()
    //////////////////////////
#define MAX_AP 21
int             i;
S24_AP_TABLE4**apTables;           // List of AP Table entries
s24Err      s24err = s24ErrNone;   // Error container
Word        actualEntries,         // Actual # of entries from API
                numEntries,
                apTableSize;

numEntries = MAX_AP;               // This should always be MAX_AP

// Set up the apTables structure:
apTableSize = numEntries * sizeof ( S24_AP_TABLE4 *);
apTables = (S24_AP_TABLE4 **) MemPtrNew( apTableSize );
ErrFatalDisplayIf(!apTables, "Unable to allocate memory for AP table list.");
MemSet( apTables, apTableSize, 0 );

for (i = 0; i < numEntries; i++)
{
    apTables[i] = (S24_AP_TABLE4*) MemPtrNew (sizeof(S24_AP_TABLE4));
    ErrFatalDisplayIf(!apTables[i],
        "Unable to allocate memory for an AP table.");
    // Clear this out!
    MemSet( apTables[i], sizeof ( S24_AP_TABLE4 ), 0 );
}

// Poll the radio for the AP table now:
s24err = S24GetAPTable( apTables, numEntries, &actualEntries);
ErrFatalDisplayIf(s24err != s24ErrNone, "Unable to get AP Tables.");

// do whatever with the table

// now free up memory
for (i = 0; i < numEntries; i++)
```

```
    if (apTables[i])
    {
        MemPtrFree(apTables[i]);
    }
MemPtrFree(apTables);
}
```

S24GetAssociationStatus

Purpose Retrieves the SPT 1740 radio's current AP association status.

Prototype

```
s24Err S24GetAssociationStatus (
                    BooleanPtr pbAssociated);
```

Parameters

-> **pbAssociated** 0=Adapter is associated.

1=Adapter is not associated.

Returned Status Zero = No errors getting the association status.

Non-zero = Error getting the association status. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments This function determines whether the SPT 1740 radio is able to communicate (*associate*) with an AP. Reasons why a SPT 1740 radio cannot associate with an AP include:

- The SPT 1740 radio is out of range of any APs in the local radio network (LRN) defined by the ESSID and/or BSSID.
- The LRN is down.
- The SPT 1740 radio is in the process of switching to another AP (*roaming*) and will reassociate quickly.

Even if an SPT 1740 radio becomes associated with an AP, do not assume the SPT 1740 radio will continue communicating with the LRN because one of the above conditions may occur. Therefore, if the SPT 1740 radio becomes unassociated with an AP, it is recommended that the application stop putting any further data messages on the stack (**NetLib**), which may increase the possibility of an error condition due to a transport process timing out a message. While the SPT 1740 radio is waiting for reassociation, the application could sit in a loop before sending any more data messages out through **NetLib**. Also, the application could put up a user interface message after being unassociated for about five seconds. Do not put up a UI message the first time the unit is unassociated. Give it some time to reassociate.

Code Sample for S24GetAssociationStatus()

```
#include "S24Api.h"

{
    ////////////////////////////
    // S24GetAssociationStatus()
    ////////////////////////////
    s24Err s24err;
    Boolean bAssociated;

    s24err = S24GetAssociationStatus( &bAssociated);
    ErrFatalDisplayIf(s24err != s24ErrNone,
        "Unable to get association status");
    if(bAssociated)
        FrmCustomAlert (GeneralAlert, "Associated", NULL, NULL );
    else
        FrmCustomAlert (GeneralAlert, "Not Associated", NULL, NULL );
}
```

S24GetBeaconParams

Purpose Retrieves the PSP power-saving mode parameters set for the SPT 1740 radio. These parameters define how frequently the radio is on to listen for access point beacons.

Prototype

```
s24Err S24GetBeaconParams (
                    WordPtr wpBeaconAlgorithm,
                    WordPtr wpBeaconMin,
                    WordPtr wpBeaconMax);
```

Parameters

-> **wpBeaconAlgorithm** The number of Access Point RF beacons that the SPT 1740 radio waits for before it listens for a signal. The beacon numbers range between 1 and 10 and a special beacon number 11.

-> **wpBeaconMin** For Beacon Algorithm 11 only. The minimum number of AP beacons to wait before listening for another AP beacon.

-> **wpBeaconMax** For Beacon Algorithm 11 only. The maximum number of AP beacons to wait before listening for another AP beacon.

Returned Status Zero = No errors getting the SPT 1740 radio beacon parameters.

Non-zero = Error getting the SPT 1740 radio beacon parameters. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments The access points transmit a signal every 100 milliseconds. This is known as a beacon.

The SPT 1740 radio can be set to listen at every beacon or to wait a set number of beacons before listening. The fewer times the radio is used, the longer the battery life, although the response time is less (see *Power Management* on page 3-5).

It is not recommended that an application adjust this algorithm haphazardly.

See Also **S24SetBeaconParams**

Code Sample for S24GetBeaconParams()

```
#include "S24Api.h"

{
    ////////////////////////////
    // S24GetBeaconParams()
    ////////////////////////////
    s24Errs24err;
    WordwBeaconAlgorithm,
        wBeaconMin,
        wBeaconMax;

    s24err = S24GetBeaconParams(&wBeaconAlgorithm,&wBeaconMin,&wBeaconMax );
    ErrFatalDisplayIf(s24err != s24ErrNone,
        "Unable to get beacon parameters");
}
```

S24GetCountryCode

Purpose Retrieves the SPT 1740 radio's country setting.

Prototype

```
s24err S24GetCountryCode (
                BytePtr pbCountryID,
                Word wIDLen,
                BytePtr pbCountryText,
                Word wTextLen);
```

Parameters

-> **pbCountryID**	A pointer to the country's ID code.
-> **wIDLen**	The length of the country's ID code.
-> **pbCountryText**	A pointer to the text description of the country.
-> **wTextLen**	The length of the country's name.

Returned Status Zero = No errors getting the SPT 1740 radio's country code.

Non-zero = Error getting the SPT 1740 radio's country code. See *Returned Status Definitions* on page 4-2 for a list of possible error codes.

Comments For informational purposes only, the returned data is a copy of the information stored in the SPT 1740 radio. Country code is a factory setting, and it is used by the SPT 1740 radio's firmware to program the radio hardware to operate within a particular country's regulations.

Code Sample for S24GetCountryCodes()

```
#include "S24Api.h"

{
        ////////////////////////////
        // S24GetCountryCode()
        ////////////////////////////
    s24Err  s24err;
    Word    wSize = S24_COUNTRY_TEXT_LEN;
    char    szText[S24_COUNTRY_TEXT_LEN];

    // first param for country code, this param is not used.
    s24err = S24GetCountryCode(NULL, wSize, (unsigned char *)szText, wSize );
    ErrFatalDisplayIf(s24err != s24ErrNone, "Unable to get country text");
    if(StrLen(szText) == 0)
    {
        // Standard configuration (Same as "United States")
        StrCopy(szText, "Standard");
    }
}
```

S24GetDriverVersion

Purpose Retrieves the version number of the SPT 1740's Spectrum24 RF driver.

Prototype

```
S24GetDriverVersion (
                    CharPtr pszVer,
                    Word nBuffLen);
```

Parameters

->	**pszVer**	A pointer to the driver version string.
->	**nBuffLen**	Length of the driver version number.

Returned Status Zero = No errors getting the driver's version number.

Non-zero = Error getting the driver's version number. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments This function is for informational purposes only. Future releases of the Spectrum24 RF driver may have additional functionality. The version number could then be used by an application to gain the added functionality.

Code Sample for S24GetDriverVersion()

```
#include "S24Api.h"

{
    ////////////////////////////
    // S24GetDriverVersion()
    ////////////////////////////

    s24Err s24err;
    WordwSize = 10;
    charszText[10+1];

    s24err = S24GetDriverVersion(szText, wSize );
    ErrFatalDisplayIf(s24err != s24ErrNone, "Unable to get country text");
}
```

S24GetInfo

Purpose Extracts various settings from the Spectrum24 Radio.

Prototype
```
S24Err S24GetInfo (
                    S24InfoPtr InfoP);
```

Parameters

->	**InfoP**	A pointer to a structure of type **S24Info**.

Returned Status Non-zero = Error in getting the Spectrum24 information. On an error condition, do not assume any data in the S24Info structure is valid. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments The caller must create the **S24Info** structure prior to calling. **S24Info** structure is defined in the **S24APIStruct.h** file.

Code Sample for S24GetInfo()

```
#include "S24Api.h"

{
    ////////////////////
    // S24GetInfo()
    ////////////////////
    s24Err      s24err;
    S24Info     Info;

    s24err = S24GetInfo( &Info);
    ErrFatalDisplayIf(s24err != s24ErrNone, "Unable to get Spectrum24 Info.");
}
```

S24GetMandatoryBSSID

Purpose Retrieves the SPT 1740 radio's mandatory BSS_ID setting.

Prototype

```
s24Err S24GetMandatoryBSSID (
                    BytePtr pszBssID, Word wBuffLen);
```

Parameters

-> **pszBssID**	A pointer to a string array where the BSS_ID is stored.
-> **wBuffLen**	ASCII byte string length of the BSS_ID (fixed length of 6 bytes).

Returned Status Zero = No errors getting the mandatory BSS_ID.

Non-zero = Error getting the mandatory BSS_ID. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments The mandatory BSS_ID forces the SPT 1740 radio to communicate only with an access point that has that BSS_ID. If a mandatory BSS_ID is set, the SPT 1740 cannot associate with any other APs in the ESS_ID network. This function returns a pointer to the BSS_ID string. The SPT 1740 radio's mandatory BSS_ID is set by the **S24SetMandatoryBSSID** call.

See Also **S24GetAdapterBSSID**
S24SetPreferredBSSID
S24SetMandatoryBSSID
S24GetPreferredBSSID

Code Sample for S24GetMandatoryBSSID()

```
#include "S24Api.h"

{
    ////////////////////////
    // S24GetMandatoryBSSID()
    ////////////////////////
    // if no mandatory BSSID, then
    s24Err      s24err;
    unsigned charszBuf[S24_MAC_ADDR_LENGTH];
    Word        wSize = S24_MAC_ADDR_LENGTH;

    s24err = S24GetMandatoryBSSID( szBuf, wSize);
    ErrFatalDisplayIf(s24err != s24ErrNone,
        "Unable to get BSSID.");
    }
```

S24GetMKKCallSign

Purpose For SPT 1740 radios programmed for Japanese configuration, this function retrieves the MKK call sign bytes.

Prototype

```
s24Err S24GetMKKCallSign (
                BooleanPtr pbValidCallsign,
                BytePtr pMKKCallSignBuff,
                Word wBuffLen);
```

Parameters

->	**pbValidCallsign**	The flag byte.
->	**pMKKCallSignBuff**	A pointer to a buffer containing the MKK call sign.
->	**wTextLen**	ASCII byte string length of the MKK call sign (fixed length of 16 bytes).

Returned Status Zero = No errors getting the MKK call sign.

Non-zero = Error getting the MKK call sign. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments This function call returns meaningful information only for SPT 1740 radios programmed for Japanese configuration. Other country configurations return zeros.

The MKK call sign buffer is structured as follows: The first byte is a flag byte. The following 15 bytes are the preamble, the frame delimiter, and the encoded MKK serial number.

S24GetMyIPAddress

Purpose Retrieves the SPT 1740's IP address after the network connection is established.

Prototype

```
Word S24GetMyIPAddress (
                Word refnum, CharPtr ipAddr);
```

Parameters

-> **refnum**	The **NetLib** refnum determined by a **SysLibFind (Net.lib", &refnum)** call.
-> **IPAddr**	An ASCII string pointer, set by the user to a buffer to be filled in by the function. The user must ensure that adequate space is allocated for a dotted IP address string (111.111.111.111).

Returned Status Zero = No errors.

Non-zero = Error number assigned by a call to **NetLibMaster()**. Refer to the Palm documentation for a list of error codes.

Comments This call can be used to acquire the IP address that is being used by a **NetLib** session. The user must call **NetLibOpen** prior to this call. This call provides a simplified interface to the **NetLib NetLibMaster** call and is especially useful for DHCP mode, where the IP address is assigned by the host. For direct IP addresses setup, the IP address returned by this function should be the same as the IP address set up in the Network Preferences.

Code Sample for S24GetMyIPAddress()

```
#include "S24Api.h"

{
    //////////////////////////
    // S24GetMyIPAddress()
    //////////////////////////
    char    szBuf[20];
    Word    wRefNum;
    Word    status;  // refer to Palm NetLibMaster() documentation

    SysLibFind("Net.lib", &wRefNum);

    status = S24GetMyIPAddress( wRefNum, szBuf);
    ErrFatalDisplayIf(status != 0, "Unable to get IP Address String.");
}
```

S24GetPreference

Purpose Retrieves the SPT 1740 radio's network preferences.

Prototype

```
s24Err S24GetPreference (
                S24PreferencesType pref,
                VoidPtr pBuff,
                Word wBuffLen);
```

Parameters

->	**pref**	An index of the network preferences.
->	**pBuff**	A pointer to the preferences buffer.
->	**wBuffLen**	Length of the preferences.

Returned Status Zero = No errors getting the network preferences.

Non-zero = Error getting the network preferences. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments **S24PreferenceType** is defined in **S24Defines.h**. S24Defines.h contains a preferences structure, S24UserNetworkPrefs, which can be used as a template for calling S24GetPreference().

See Also **S24SetPreference**
S24PreferencesType is defined in **S24Defines.h**.

Code Sample for S24GetPreference()

```
#include "S24Api.h"

{
    s24Err      s24err;
    ULong       ulBufSize;
    S24UserNetworkPrefs Prefs;

    ulBufSize = sizeof(Prefs.szESSID);

    // Pass buffer to S24GetPreferences() to fill it out:
    s24err = S24GetPreference( s24PrefESSID, Prefs.szESSID, ulBufSize);
    ErrFatalDisplayIf(s24err != s24ErrNone,
        "Unable to get ESSID string from S24GetPreferences.");
}
```

S24GetPreferredBSSID

Purpose Retrieves the SPT 1740 radio's preferred BSS_ID setting.

Prototype

```
s24Err S24GetPreferredBSSID (
                    BytePtr pbyBssID, Word wBuffLen);
```

Parameters

->	**pbyBssID**	A pointer to the SPT 1740 radio's preferred BSS_ID.
->	**wBuffLen**	Fixed buffer length (6 bytes minimum).

Returned Status Zero = No errors getting the preferred BSS_ID.

Non-zero = Error getting the preferred BSS_ID. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments A preferred BSS_ID forces the SPT 1740 radio to try to associate with an AP that has that BSS_ID. If it cannot associate with that AP, the SPT 1740 radio tries to associate with another AP in the ESS_ID network. The SPT 1740 radio's preferred BSSID is set by the **S24SetPreferredBSSID**.

See Also

S24GetAdapterBSSID
S24GetMandatoryBSSID
S24SetMandatoryBSSID
S24SetPreferredBSSID

Code Sample for S24GetPreferredBSSID()

```
#include "S24Api.h"

{
    //////////////////////////
    // S24GetPreferredBSSID()
    //////////////////////////
    s24Err  s24err;
    char    szBuf[S24_MAC_ADDR_LENGTH];
    Word    wSize = S24_MAC_ADDR_LENGTH;

    s24err = S24GetPreferredBSSID( szBuf, wSize);
    ErrFatalDisplayIf(s24err != s24ErrNone,
        "Unable to get preferred BSSID.");
}
```

S24SetAdapterESSID

Purpose Sets the SPT 1740 radio's ESS_ID setting.

Prototype

```
s24Err S24SetAdapterESSID (
                    CharPtr szEssID);
```

Parameters

->	**szEssID**	A NULL-terminated string representing the ESS_ID. Maximum string length is 32 characters. Avoid special characters, especially nonprintable characters.

Returned Status Zero = No errors setting the ESS_ID.

Non-zero = Error setting the ESS_ID. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments In the 802.11 protocol, the ESS_ID is the identification name for the local radio network (LRN). All devices (access points and SPT 1740 units) in the LRN that communicate with each other must have the same ESS_ID. **S24SetAdapterESSID** assigns the network's ESS_ID to the SPT 1740 radio. The SPT 1740 radio firmware uses the ESS_ID to probe for access points with which it can associate.

The SPT 1740 radio's initial ESS_ID is set from the network (Spectrum24) preference panel.

Spectrum24 library must be open and connected to a valid AP. Change the ESS_ID to another valid AP ESS_ID and check for association. These changes are not permanent, and are only valid with the library is open.

See Also **S24GetAdapterESSID**

Code Sample for S24SetAdapterESSID()

```
#include "S24Api.h"

{
    ////////////////////////////////////////////
    // S24SetAdapterESSID()
    ////////////////////////////////////////////
    //
    // Generally, the ESSID is set through the pref
    // panel or an S24SetPreference() call.  Dynamic
    // modification of the ESSID is not encouraged.
    // This call is not persistant after a NetLibClose.
    //
    // If you are going to proceed anyway, here is
    // some sample code.
    ////////////////////////////////////////////
    s24Err  s24err;
    char    szNewESSID[S24_ESS_ID_LENGTH+1];
    char    szOldESSID[S24_ESS_ID_LENGTH+1];
    Boolean bAssociated;

    s24err = S24GetAdapterESSID( szOldESSID, S24_ESS_ID_LENGTH);
    ErrFatalDisplayIf(s24err != s24ErrNone, "Unable to get Adapter ESSID");

    StrCopy(szNewESSID, "Test_ESSID");
    s24err = S24SetAdapterESSID( szNewESSID );
    ErrFatalDisplayIf(s24err != s24ErrNone, "Unable to set adapter ESSID.");

    ////////////////////////////
    // S24GetAssociationStatus()
    ////////////////////////////

    //give time to associate to new ESSID
    SysTaskDelay(sysTicksPerSecond*3);

    s24err = S24GetAssociationStatus( &bAssociated);
    ErrFatalDisplayIf(s24err != s24ErrNone,
        "Unable to get association status");
    if(bAssociated)
        FrmCustomAlert (GeneralAlert, "Associated", NULL, NULL );
```

```
    else
        FrmCustomAlert (GeneralAlert, "Not Associated", NULL, NULL );

    // restore old ESSID
    s24err = S24SetAdapterESSID( szOldESSID );
    ErrFatalDisplayIf(s24err != s24ErrNone, "Unable to set adapter ESSID.");

    ////////////////////////////
    // S24GetAssociationStatus()
    ////////////////////////////

    //give time to re-associate to old ESSID
    SysTaskDelay(sysTicksPerSecond*3);

    s24err = S24GetAssociationStatus( &bAssociated);
    ErrFatalDisplayIf(s24err != s24ErrNone,
        "Unable to get association status");
    if(bAssociated)
        FrmCustomAlert (GeneralAlert, "Associated", NULL, NULL );
    else
        FrmCustomAlert (GeneralAlert, "Not Associated", NULL, NULL );
}
```

S24SetBeaconParams

Purpose Configures the power save polling (PSP) mode parameters by specifying the number of beacons that the SPT 1740 radio waits for before it begins listening for a beacon signal.

Prototype

```
s24Err S24SetBeaconParams (
                Word wBeaconAlgorithm,
                Word wBeaconMin,
                Word wBeaconMax);
```

Parameters

-> **wBeaconAlgorithm**	The number of access point RF beacons that the SPT 1740 radio waits for before it listens for a beacon signal.
-> **wBeaconMin**	For Beacon Algorithm 11 only. The minimum number of AP beacons to wait for before listening for another AP beacon.
-> **wBeaconMax**	For Beacon Algorithm 11 only. The maximum number of AP beacons to wait for before listening for another AP beacon.

Returned Status Zero=No errors setting the PSP beacon parameters.

Non-zero=Error setting the PSP beacon parameters. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments Access points transmit a broadcast signal every 100 milliseconds. This is know as a beacon. The SPT 1740 radio can be set to listen at every beacon, or to wait a set number of beacons before listening. The fewer times the radio is used, the longer the battery life, although the response time is less (see *Beacon Algorithms-Power Save Polling (PSP)* on page 2-7). The default PSP mode is PSPll, which operates on every beacon when there is message traffic for the SPT 1740, and backs off to every 10th beacon (once per second) when there is no traffic. It is not recommended that an application adjust this algorithm haphazardly.

See Also **S24GetBeaconParameters**

Code Sample for S24SetBeaconParams

```
s24Err s24err;
Word wBAlgorithm,
    wBMin,
    wBMax;

if(<user wants to set to most flexible>)
{
    wBAlgorithm = 11;
    wBMin       = 1;
    wBMax       = 10;
}
else if (<user wants to select slowest beacon rate>)
{
    wBAlgorithm = 10;
    wBMin       = n/a;
    wBMax       = n/a;
}

printf("\nAbout to S24SetBeaconParams...");

s24err = S24SetBeaconParams( wBAlgorithm, wBMin, wBMax );

if (s24err == s24ErrNone)
{
    printf(" success.");
}
else
    printf(" Failure, S24Err #%d", s24err);
```

S24SetMandatoryBSSID

Purpose Assigns the mandatory access point BSS_ID to the SPT 1740 radio.

Prototype

```
S24 SetMandatoryBSSID (
                    BytePtr pbyBssID);
```

Parameters

->	**pbyBssID**	The BSS_ID of the associated access point.

Returned Status Zero = No errors setting the mandatory access point's BSS_ID.

Non-zero = Error setting the mandatory access point's BSS_ID. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments Assigning a mandatory access point BSS_ID to an SPT 1740 radio causes the SPT 1740 radio to communicate only with the access point that has that BSS_ID. If the SPT 1740 moves out of the mandatory access point's range, the SPT 1740 will not reconnect to the network until the SPT 1740 is brought back in range of the mandatory BSSID.

Use this call for specific load-leveling configurations.

See Also

S24GetAdapterBSSID
S24GetMandatoryBSSID
S24GetPreferredBSSID
S24SetPreferredBSSID

Code Sample for S24SetMandatoryBSSID

```
s24Err s24err;
Byte byMac[6];

// Parse the argument:
if  (   (argc > 1) && // Two args
        (argv[1]) && // second arg is valid ptr
        (StrLen(argv[1]) >= 17)// second arg s/b like: "00:A0:F8:00:22:F8"
    )
    UnformatIEEEAddress( argv[1], byMac, 6 );// Convert into bytes
else
{
    printf("\nInvalid argument");
    goto FullHelp;
}

printf("\nAbout to perform S24SetMandatoryBSSID");

s24err = S24SetMandatoryBSSID( byMac );

if (s24err == s24ErrNone)
    printf(" success!");
else
    printf(" Failure, S24Err #%d", s24err);
```

S24SetPreference

Purpose Sets the SPT 1740 radio's network preferences.

Prototype

```
s24Err S24SetPreference (
                S24Preferences pref,
                VoidPtr pBuff,
                Word wBuffLen);
```

Parameters

->	**pref**	An index of the network preferences.
->	**pBuff**	A pointer to the preferences buffer.
->	**wBuffLen**	Length of the preferences.

Returned Status Zero = No errors setting the network preferences.

Non-zero = Error setting the network preferences. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments **S24PreferenceType** is defined in **S24Defines.h**. S24Defines.h contains a preferences structure, S24UserNetworkPrefs, which can be used as a template for calling S24GetPreference().

See Also **S24GetPreference**

Code Sample for S24SetPreference

```
s24Err s24err;
Word wNewVCCTimeout = 600;  // 10 minutes

printf("\nAbout to perform _S24SetPreference() on s24PrefVCCOffTimeout with %d
seconds.\n", wNewVCCTimeout);

s24err = S24SetPreference(
s24PrefVCCOffTimeout,&wNewVCCTimeout,sizeof(wNewVCCTimeout));

if (s24err == s24ErrNone)
    printf(" success!");
else
    printf(" Failure, S24Err #%d", s24err);
```

S24SetPreferredBSSID

Purpose Assigns the preferred access point BSS_ID to the SPT 1740 radio.

Prototype

```
s24Err S24SetPreferredBSSID (
                    BytePtr pbyBssID, Word wBuffLen);
```

Parameters

->	**pbyBssID**	The BSS_ID of the associated access point.

Returned Status Zero = No errors setting the preferred access point's BSS_ID.

Non-zero = Error setting the preferred access point's BSS_ID. See *Returned Status Definitions* on page 4-2 for a list of possible codes.

Comments Assigning a preferred access point BSS_ID to an SPT 1740 radio causes the SPT 1740 radio to attempt to communicate with the preferred access point. As long as the SPT 1740 can hear the BSSID access point, the SPT 1740 maintains association to that access point. If however, the SPT 1740 goes out-of-range of the preferred BSSID access point, it is allowed to find another access point as long as the ESSIDs match.

Use this function call to share the load among access points that can communicate with a number of SPT 1740s.

See Also

S24GetAdapterBSSID
S24GetMandatoryBSSID
S24GetPreferredBSSID
S24SetMandatoryBSSID

Code Sample for S24SetPreferredBSSID()

```
#include "S24Api.h"

{
    //////////////////////////
    // S24SetPreferredBSSID()
    //////////////////////////
    s24Err  s24err;
    char    szBuf[S24_MAC_ADDR_LENGTH];
    Word    wSize = S24_MAC_ADDR_LENGTH;

    s24err = S24SetPreferredBSSID( szBuf);
    ErrFatalDisplayIf(s24err != s24ErrNone,
        "Unable to set Preferred BSSID.");
}
```

S24SetTcpResendTimeout

Purpose Sets the initial timeout value for **NetLib** to reattempt a transmission.

Prototype
```
void S24SetTcpResendTimeout (
                    Word libRefNum,
                    DWord tcpResendTimeout);
```

Parameters

-> **refnum** The **NetLib refnum** is determined by a **SysLibFind("Net.lib", &refnum)** call.

Returned Status None.

Comments The **NetLib** resend logic dynamically adjusts the resend timeout value. This value is the initial value only.

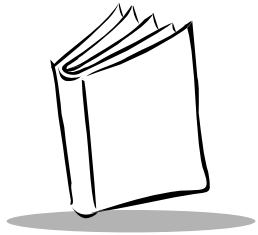
Code Sample for S24SetTcpResentTimeout()

```
#include "S24Api.h"

{
    //////////////////////////
    // S24SetTcpResendTimeout()
    //////////////////////////
    Word    wRefNum;

    //  Time in milliseconds before TCP resends a packet.
    //  This is just the initial value, the timeout is adjusted
    //  from this initial value depending on history of ACK times.
    //  This is sometimes referred to as the RTO (Roundtrip Time Out)
    //  See RFC-1122 for additional information.
    DWord   dwTimeout;

    SysLibFind("Net.lib", &wRefNum);

    dwTimeout = 3000;// 3 seconds (this is an example, not a recommendation)

    S24SetTcpResendTimeout( wRefNum, dwTimeout);
}
```

Glossary

Term	Definition
Access point (AP)	A Spectrum24 component that is physically attached to an Ethernet LAN and is equipped with an Spectrum24 radio. Access points communicate with other Spectrum24 radio-enabled equipment, such as *SPT 1740s*.
BSS_ID	Basic Service Set ID (IEEE MAC address).
ESS_ID	Extended Service Set ID. 802.11 network name. Can be up to 32 ASCII characters in length.
Mandatory access point BSS_ID	If an SPT 1740 is configured with a mandatory access point's BSS_ID, the SPT 1740 can communicate only with that mandatory access point. If the SPT 1740 moves out of range, it cannot communicate with another access point.
Preferred access point BSS_ID	If an SPT 1740 is configured with a preferred access point's BSS_ID, the SPT 1740 attempts to communicate with that access point, even if the SPT 1740 is within range of access points with better quality signals.
PSP mode	Power save polling mode. PSP modes determine the SPT 1740 radio wake-up cycle. PSP modes are defined as PSP-1, PSP-2, and so on to PSP-10. There is also a self-adapting mode, PSP-11. PSP-1 wakes up the SPT 1740 radio every 100 milliseconds; PSP-2 wakes up the radio every 200 milliseconds, and so on. PSP-10 wakes up the radio SPT 1740 every 1 second (1000 ms). The default is PSP-11, in which the SPT 1740 radio starts in PSP-1 mode, then degrades to PSP-10 if no network traffic is encountered. If network traffic is encountered, PSP-1 starts immediately and continues until traffic stops; then the SPT 1740 radio continues to degrade to PSP-10. The SPT 1740 radio is best optimized for battery power consumption under normal operation in PSP-11.

Term	Definition
Roaming	A condition in which an SPT 1740 radio associates from one AP to another.



Index

H

I

L

M

N

P

R

S

T

V

W