Table of Contents

List of Acronyms

API	Application Programming Interface	**RARP**	Reverse Address Resolution Protocol
ARP	Address Resolution Protocol	**SDK**	Software Development Kit
ASIC	Application-specific Integrated Circuit	**SLIP**	Serial Line Internet Protocol
IP	Internet Protocol	**TCP**	Transmission Control Protocol
NIC	Network Interface Card	**UART**	Universal Asynchronous Receiver-Transmitter
OSI	Open Systems Interconnection	**UDP**	User Datagram Protocol
Palm OS	Palm Operating System	**USB**	Universal Serial Bus
PPP	Point-to-Point Protocol		

1. Overview

The purpose of this document is to describe the communications interface between applications running on Handspring™ handhelds and hardware on Springboard™ communication modules. The document is divided into several components, as follows:

TCP/IP Protocol Stack: TCP/IP is the typical architecture used for Internet-based communications. Many LAN systems are also configured with this protocol. This section provides a quick definition of TCP/IP to help you visualize how network communications are implemented in Handspring handhelds, along with how application software can take advantage of built-in network functionality.

Network Protocols in the Palm OS® Environment: This section shows how the TCP/IP protocol stack is implemented in the Palm OS.

Preference Panels: This section introduces you to several additional functions provided by the Palm OS to manage modem and network connection details. The Preference Panels are analogous to the functions in the Windows Control Panel.

Communications Architecture in the Handspring Visor™: The implementation of TCP/IP is defined within the Palm OS environment. In this section, you will take a look at the specific implementation details with a Handspring handheld to support the Springboard expansion slot, USB support, and external keyboards.

The Springboard Philosophy: This section briefly describes the architectural goals of the Springboard interface. It is helpful to review this section in order to understand the reference design.

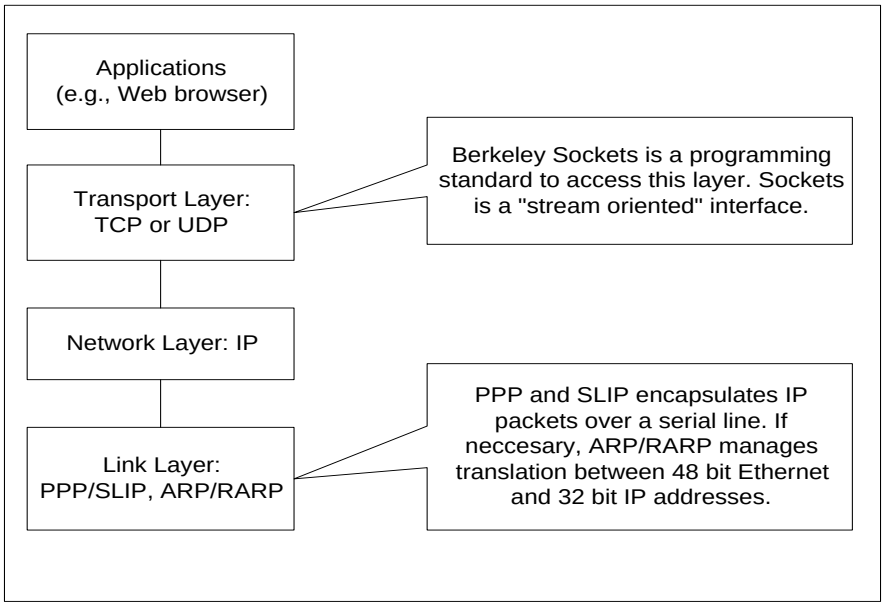
Hardware Reference Design: This section examines the Diagnostic Module as a starting point for designing a communications module. The Diagnostic Module interfaces the Springboard parallel bus to a serial interface with buffering in a UART. Software is resident on the module Flash memory to implement the plug-and-play functionality of Springboard.

Software Reference Design: This section reviews the software that runs on the Diagnostic Module. This software drives the UART on the Diagnostic Module and is a transparent replacement for the serial port. As such, the existing TCP/IP protocol stack -- and application software that uses it -- works without modifications.

We'll also take a look at an example of a communications configuration using a Handspring handheld and the reference design. Finally, we'll document some of the function calls referenced earlier in the document.

2. TCP/IP Protocol Stack

The architecture for Internet communications in a desktop/notebook environment is well known. A typical communications stack is illustrated below.



TCP/IP is described with a four-layer protocol stack: Application, Transport, Network, and Link. This is what we reference in this document, as opposed to the 7-layer OSI model with which many developers may be familiar.

The Link Layer is responsible for managing hardware details. This layer is where the drivers for a serial port or Ethernet NIC reside. In a mobile or remote environment where no local area network exists, PPP and SLIP are often used to connect to the Internet through modem and serial connections, respectively. PPP and SLIP encapsulate IP packets. In a network environment, implementation of TCP/IP over Ethernet requires an additional component of address translation. ARP and RARP provide this component.

The Network Layer is responsible for the basic movement of packets through the network. IP is the protocol that provides a “best attempt” at delivering the information to the intended destination. IP is connectionless (i.e., there is no logical link between destination and source packets) and unreliable (i.e., there is no guaranteed delivery).

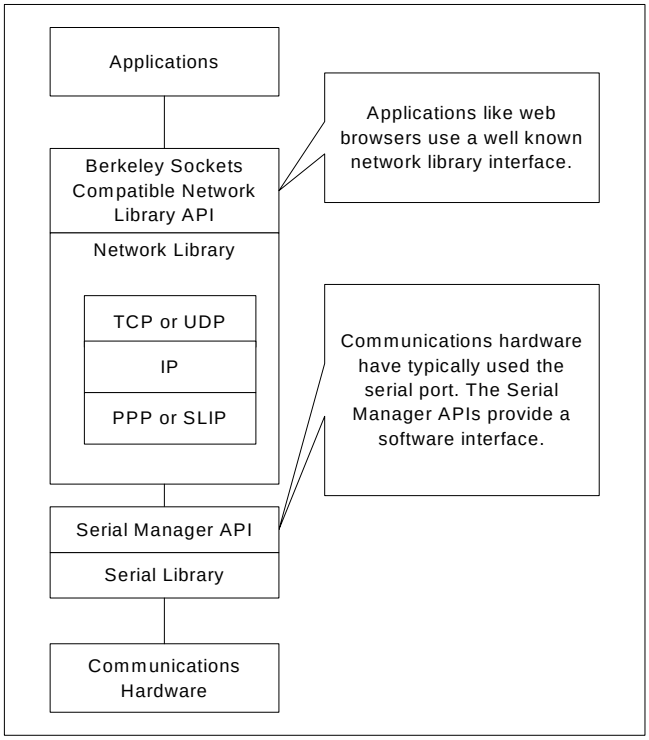
The Transport Layer provides a flow of data between two systems. TCP and UDP are both supported protocols. TCP provides a reliable, connection-oriented protocol that manages timeouts, acknowledgments, and other transport management functions. UDP is a much simpler protocol that is connectionless and unreliable. UDP produces one datagram for each write operation in contrast to TCP, which breaks up application data as appropriate, then manages the reconstruction on the remote end.

Berkeley Sockets is a stream-oriented programming interface for the Transport Layer. This interface introduces the concept of a socket or pipeline between two systems. The data that an application sends into the pipe reappears in the remote system. In a Windows system, this interface is implemented as WinSock. In the Palm OS, the Network API is compatible with Berkeley Sockets.

3. Network Protocols In The Palm OS Environment

Historically, the serial port on the cradle connector has been the only way to connect a Palm OS based device with the outside world. The Serial Manager (Serial Library) is used to communicate with the actual serial hardware. A Network Library (NetLib) incorporates functionality at the Transport (TCP and UDP), Network (IP), and Link (PPP and SLIP) Layers. These layers are integrated to optimize performance (speed and space) in a handheld environment. In fact, the Network Library exists as a shared library, and must be expressly opened by an application to conserve dynamic memory space when not in use. Applications use a Berkeley Sockets-compatible API to take advantage of the network libraries within the handheld.

The overall model looks like this:



The Serial Manager API provides basic byte-oriented communications functionality. These function calls include:

- SerClose()
- SerOpen()
- SerReceive()
- SerSend()
- SerReceiveWait()
- SerGetStatus()

The Berkeley Sockets Compatible Network API provides basic stream-oriented communications functionality. A complete listing is available within the Palm OS® SDK Reference. These function calls include:

- NetLibSocketConnect()
- NetLibSocketClose()

- NetLibReceive()
- NetLibSend()
- NetLibSocketOpen()

Note that Palm OS 3.5 provides additional communications functionality by supporting multiple serial drivers, providing packet-oriented communications (compared to byte-oriented), and supporting web-oriented protocols (http and https). These additional functions are not included in Palm OS 3.1 and will be described in another document.

4. Preference Panels

There are several features that work in combination with the Palm OS communications architecture. The Prefs application manages the details of the modem and network settings. A user can set these parameters by launching the Prefs application, then selecting the Modem or Network menu items from the top right corner of the display.

For the user: The Prefs application contains a Modem panel to define a dialing string, baud rates, and any required setup strings. The Prefs application also contains a Network panel to define network parameters regarding PPP or SLIP connections. User names, passwords, phone numbers (or 00 for a direct connection), login scripts, and IP addresses are all defined here.

For the developer: The Modem Manager provides high-level functions to establish a modem connection using the definitions established in the Prefs modem panel. The Network Library API uses network information stored in the Prefs when executing function calls. For example, the NetLibOpen() function call might use the configuration information in the Prefs network panel to establish a PPP connection through a modem.

The Modem and Network features are fully detailed in the Palm OS SDK materials referenced at the end of this document.

5. Communications Architecture In The Handspring Visor

The Handspring Visor is based on Palm OS 3.1. Note that even when new versions of the Palm OS are released, retaining compatibility with Palm OS 3.1 will give you access to a wider base of installed devices. Within this environment, serial communication functions are provided through shared libraries. These libraries are registered in a table using `SysLibLoad()`. Application software uses the `SysLibFind()` function to select a library by name. `SysLibFind()` then returns a reference number that is used in subsequent serial library function calls.

"`Serial Library`" is the default name used by Palm OS applications to use the serial port. With Visor, the functionality of this library has been modified to allow for re-direction. At the time of the Palm OS 3.1 release, there was no way to replace the existing "`Serial Library`" through the Palm OS APIs. The advantage of changing only the serial driver layer to support new hardware (while retaining backward compatibility with existing software) was compelling from both customer experience and development platform perspectives.

To accommodate this need, Handspring implemented API extensions called `HsGetPref()` and `HsSetPref()`. Note that Palm OS 3.5 also provides a mechanism for managing multiple serial libraries, but these calls will not work with the growing installed base of Visor handhelds with Palm OS 3.1. To maintain backward compatibility, software should determine whether it is running on a Visor and execute the appropriate function calls. `HsGetPref()` and `HsSetPref()` are documented in the "Springboard Development Guide for Handspring Handheld Computers" available on-line at:
http://www.handspring.com/developers/documentation.jhtml.

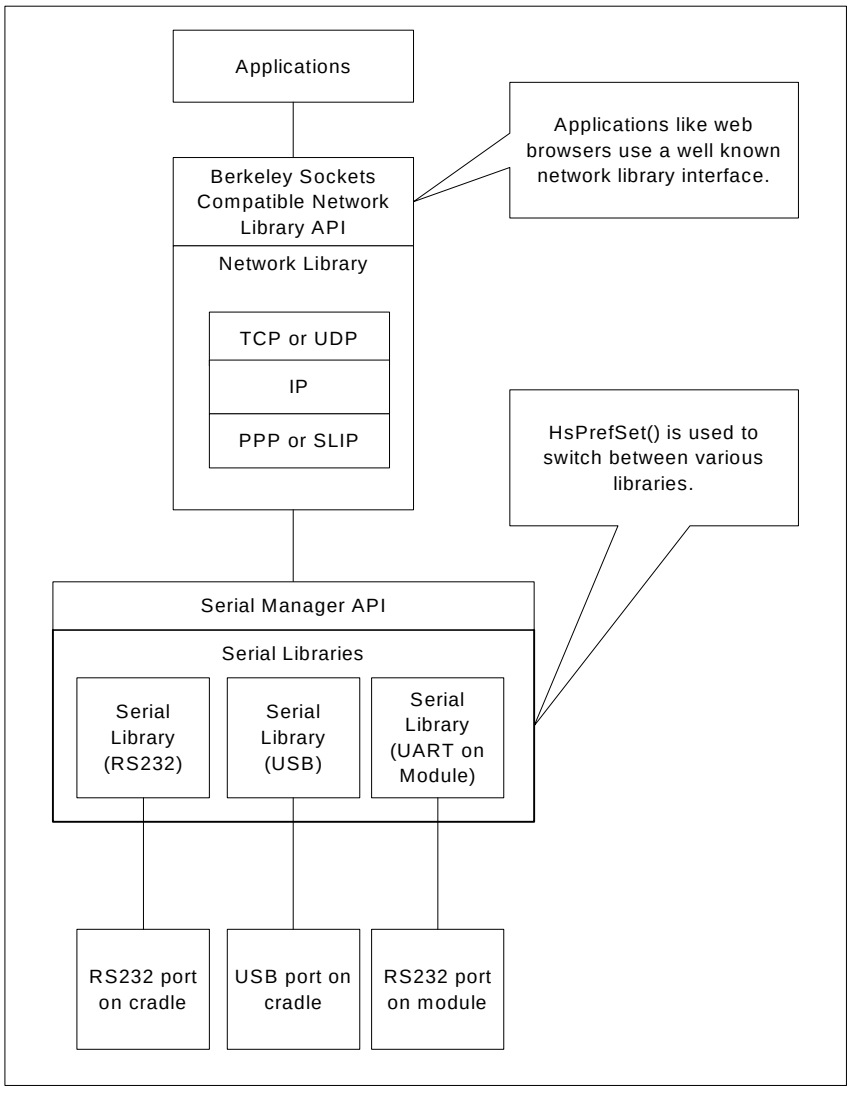
In a Handspring handheld application calls `SysLibFind()` to register the "`Serial Library`" the library that is actually loaded is defined by `HsSetPref()`. `SysLibFind()` retrieves these preferences and loads the

appropriate library. The change is transparent to the application software since the re-direction can be done by a module Setup Application (discussed in more detail later).

Should an application want to specifically eliminate the possibility of re-direction, the original library has been renamed "`BuiltIn SerLib`." An application using this library will always use the built in UART that communicates with the serial port on the cradle connector.

Another library has been defined under the name "`USB Library.`" As the name suggests, this library supports USB as the communications mechanism. Applications that specifically want to use USB should request this library.

The architecture underlying a typical communications program on Visor is illustrated below:



There are a few other nuances of which developers should be aware:

5.1. Hotsync® and Keyboards

Hotsync is a special case. The library that Hotsync uses has been modified so that both USB and serial communications are supported automatically. The serial cradle for the Handspring Visor grounds a pin labeled KBD* on the cradle port (the asterisk denotes an active low signal). This signal indicates to the operating system that it is in a serial cradle. If KBD* is asserted, Hotsync will use the "`Serial Library`", which can be re-directed via `HsPrefSet()` to a Springboard module, if necessary. If KBD* is not asserted, HotSync will use the "`USB Library`."

Additionally, when Visor is placed into a serial cradle a daemon is automatically launched to support keyboard input. This process opens the "`BuiltIn SerLib`" and waits for specially formatted packet data on the Rx pin (representing keyboard input). At this point, if an application were to open a serial port library again, an error would return indicating that the port is already in use.

The workaround for this error is to disable the keyboard process via software:
Example:

```
#include <HsExt.h>

// If on Handspring device, disable the keyboard
// thread before opening the serial library.
if (!FtrGet ('hsEx', 0, &ftrHSVersion))
HsExtKeyboardEnable (false);

(continue with SysLibFind(), SerOpen(), etc)

// Note that there is no need to re-enable the keyboard support.  // It is
automatically re-enabled each time it is placed into the // cradle.
```

For development purposes, soft-resetting the Visor in the serial cradle will also result in disabling the keyboard daemon. That is because it is launched when the device is inserted into the cradle, but is not re-started until Visor is removed and re-inserted into the serial cradle.

5.2. Console Mode

A full description of Console Mode and the Palm Debugger can be found in the Palm OS® Programming Development Tools Guide.

Console Mode is used with Palm Debugger (and other tools) to provide information flow between the desktop and handheld environments. Console Mode supports both USB and RS232 communications. It does this by looking at the state of the Up arrow on Visor and loads a specific library. If the Up arrow is pressed when launching Console Mode via the [shortcut .. 2] command, Console Mode will use "`Serial Library`." If the Up arrow is not pressed, Console Mode will use "`USB Library`." Finally, if `HsPrefSet`() has redirected the serial library, the Up arrow is ignored.

Note that Debug mode within Palm Debugger can also be configured to use USB or serial. However, this happens at a lower level, not through `HsPrefSet`().

5.3. IRDA

There is only one UART shared by both RS232 and IR ports. Beaming can also be enabled such that the unit will receive a "beam" from another device automatically. This works if two requirements are met:

(a) "Beam Receive" is turned on in Preferences>General.
(b) No other application or function is currently using the serial library. Beaming does not function when an application is actively using the serial port (Hotsync, background keyboard process, or other communications application).

5.4. UART Details

At its lowest level, the UART used in Visor is integrated onto the CPU. As such, there is no physical UART chip outside the CPU. The detailed specifications for the UART are part of the CPU specifications. For Visor, this is the Motorola Dragonball EZ.

This family of processors supports memory-mapped I/O. This is in contrast to other processors that have separate instructions for I/O and memory operations. Registers for integrated functions, like the UART, are mapped at the top of memory.

Please note that writing directly to the hardware is **highly discouraged** as it often leads to incompatibilities with future devices.

6. The Springboard Philosophy

The Springboard architecture provides a mechanism for true plug-and-play capabilities. To support this feature, all software required to operate the module resides in the module memory itself. When a module is inserted, memory on the module is mapped into the Handspring handhelds memory space. The system will automatically load and execute a Setup Application on the module that will perform all the appropriate initialization functions. Conversely, upon removal the Setup Application will be called again to clean up the handheld environment, as appropriate.

Specifically for Springboard communications modules, the Setup Application can be used to re-direct Serial Library functions to a library written specifically for the hardware on module. Since the Serial Library is re-directed, the Network Manager data (a Serial Library client) will likewise be directed to the module. Software installed by the customer that makes use of Serial Manager or Network Manager API functions will work transparently with the module.

A parallel interface is needed to support I/O between the Handspring handheld and a Springboard module. A memory-mapped UART is often used in communication module designs. The next section outlines a specific module implementation given the Springboard design philosophy.

In addition to the communications aspects of Springboard modules, developers should review other general design considerations. For example, handling unexpected module extractions and other advanced software topics are covered in application notes on the Handspring website:

http://www.handspring.com/developers/tech_notes.jhtml

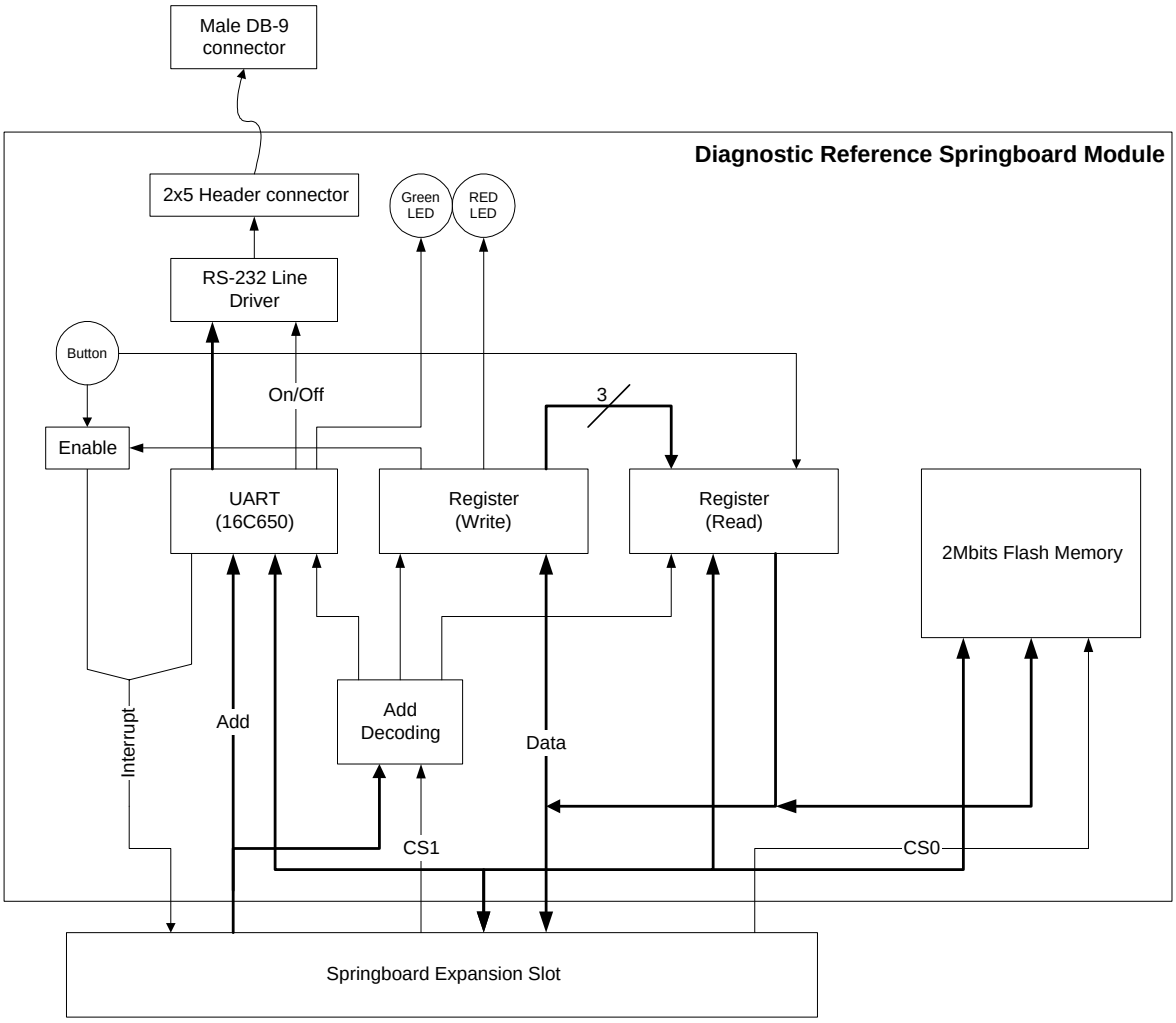
In addition, general Springboard module development is covered in several documents on our website: "Handspring Development Kit" and "Springboard Module Development in 7 Steps".

http://www.handspring.com/developers/getting_started.jhtml

7. Hardware Reference Design

The full details of the Diagnostic Module are available on the website in the form of an application note. It can be a reference design for a Springboard communications module. This design is what is used as an example in this paper. An application note (AN03: "Diagnostic Reference Module") covers this module in more detail at:

http://www.handspring.com/developers/tech_notes.jhtml



The Diagnostic Module's electrical hardware consists of a Flash memory device, a serial UART device, registers, and latches. These components can be accessed via the Springboard slot to demonstrate the basics of the Springboard interface.

- Address decoding logic to demonstrate how to map multiple devices and functions.
- Flash memory device to demonstrate how to interface memory to the Springboard expansion bus to store module-resident software applications.
- 16650 UART device to demonstrate how to interface and communicate with memory-mapped I/O devices on the Springboard expansion bus. The UART provides a demonstration of basic serial communication protocol with external devices such as a standard communication port on a PC or most RS-232 compatible communication ports.

- Registers to control the red LED and the interrupt enable of the push-button. These demonstrate glue logic interfaces such as address decoding and register read-write controls.
- Latches to demonstrate how to implement control status such as microphone signal detect and software-controlled interrupt readings

The address decoding is listed in the table below. All of the I/O devices listed below are mapped to chip select 1, which starts at address location 0x29000000. Only Address lines A23, A22, WE* and OE* are used to decode the devices' enable/select.

Springboard signals					Device accessed
A23	**A22**	**WE***	**OE***	**Address location**	**Device**
0	0	X	X	0x290x xxxx	16650 UART
0	1	0	1	0x294x xxxx	Address Latch Write
0	1	1	0	0x294x xxxx	Address Latch Read
1	0	1	0	0x298x xxxx	Status Latch (read only)
1	1	X	X	0x29Cx xxxx	GPIO Register

The GPIO Register implements controls for the Red LED and the push-button interrupt circuitry. It also serves as a register read-back for data lines D7-D5. The address and data controls are listed in the table below.

GPIO Register (Address 0x29CX XXXX)	
Data Bit #	Definition
0	Red LED (0-off, 1-on)
1-2	Interrupt button enable/clear (01- clear; 10 – set)
3	Unused
4-6	Set GPIO (4-6) (Readable from Status Register)
7	Unused

From a communications viewpoint, a UART is useful for several reasons. First, it provides a level of buffering to manage interrupt latency. A 16 or 32 byte buffer is common in many UART designs. Second, a memory-mapped UART provides an easy initial parallel interface to the Springboard module. The next design step may be to incorporate the UART -- or replace its function -- in an ASIC. Finally, many communications systems are already using a serial interface. The Diagnostic Module can be used to build software in parallel with hardware development, and to test "big board" designs prior to creating a form factor prototype.

8. Software Reference Design

As with the hardware details, the full details of the Diagnostic Module software is available on the website in the form of an application note. An application note (AN03: "Diagnostic Reference Module") covers this module in more detail at:

http://www.handspring.com/developers/tech_notes.jhtml

The Diagnostic Module's software consists of a set of applications that demonstrate various features of the Springboard application. The software classification types and descriptions are listed below.

- The optional Card Welcome application that provides a welcome greeting and preference configurations.
- The required Card Setup application that facilitates the plug-and-play paradigm by installing the necessary software hooks, registering interrupt handlers and other event handlers, and installing necessary libraries and databases.
- The Hand Terminal "main" application sample provides the application's user interface and the functional processing. It is the main application of the Diagnostic module. The Hand Terminal application implements the basic features of terminal communication software.
- The 16550 UART serial library provides all the standard serial library API replacements. This library maps the serial functions into the hardware controls of the UART device.
- The Serial Diagnostic utilities to provide statistical performance monitoring and data viewing. (This feature is enabled through conditional compilation directives.)
- The Card Updater application to build all the above applications and combine them into resource data to be programmed onto the card's Flash memory.

The serial library that comes with the Diagnostic Module is compatible with Palm OS applications. Hotsync re-direction through the module serial port is an example of this level of compatibility. Since the source code for the library is available, it can be the base from which you develop a specific serial library to manage the nuances of your communications hardware.

Various wired and wireless communications modules will be designed around a similar philosophy. For example, CDPD and cellular radio modules are planned that use the UART interface to communicate with software on the handheld. A serial library will provide the hardware-specific interface to the module. With this approach, NetLib and other applications written for Palm OS can operate over a new network infrastructure without modification.

It is important to note that the characteristics of the network may vary significantly. In particular, wireless networks vary significantly in terms of latency and throughput, which will impact the user experience. The developer should consider how this might impact the design of the application software.

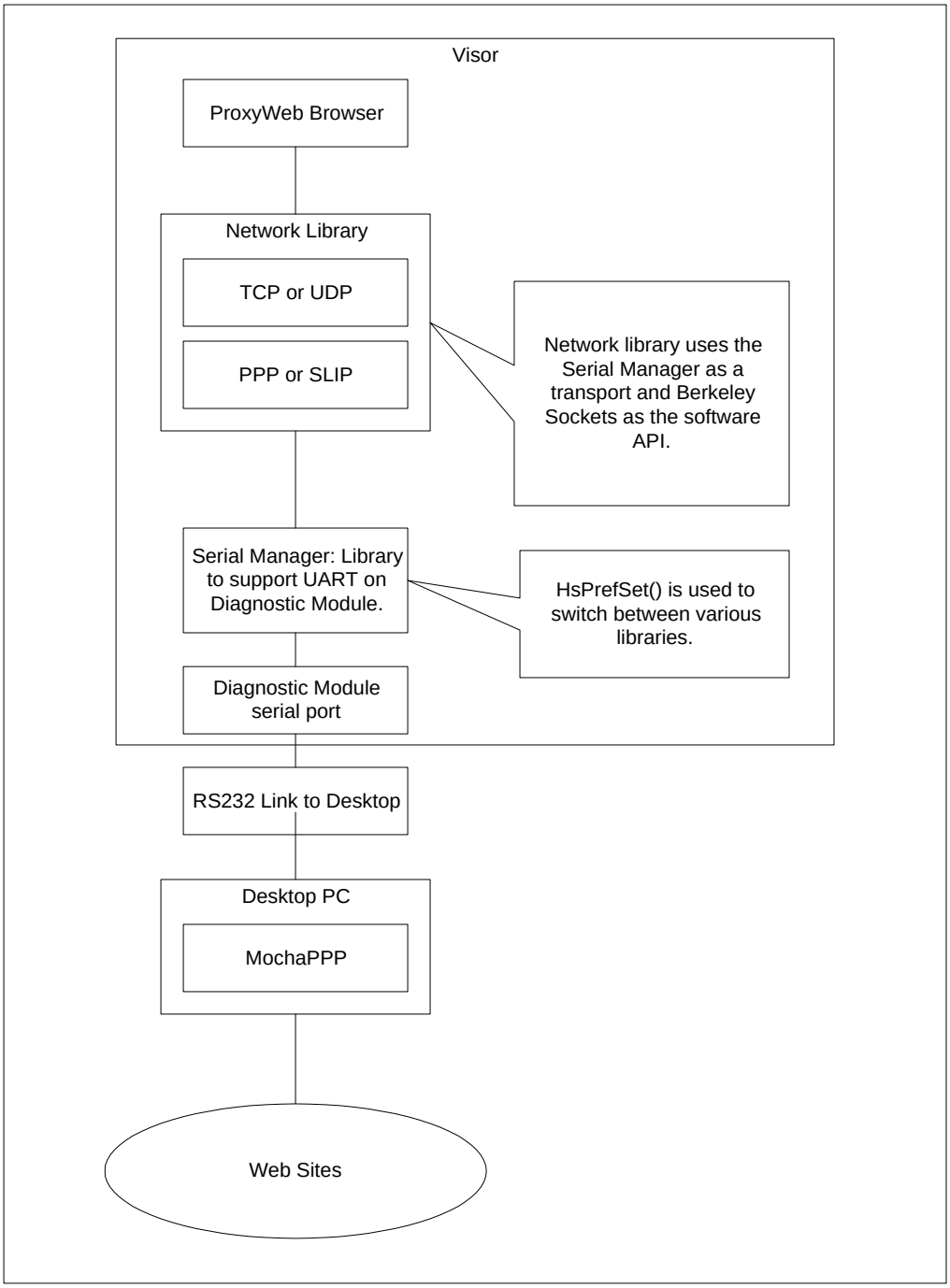
8.1. Other Approaches

For some applications, a UART may not be an appropriate interface for your module. For example, certain designs may require faster data transfer speeds. In this case, shared memory on the Springboard module may be the best way to buffer data.

Also, certain implementations would not be appropriate for a serial interface. For example, suppose you want to design an Ethernet module. Translation of an Ethernet address to an IP address is not included in the Diagnostic Module reference design or serial library software. Additionally, function calls that are expected of a serial library, such as setting the baud rate, don't apply to Ethernet implementations.

8.2. Example Application

The following example illustrates how a communications module can work with standard Palm OS communications software. ProxyWeb is a Palm OS web browser that uses NetLib. MochaPPP is an application that establishes a PPP connection on the handheld through a desktop PC. Using the Diagnostic Module, a Handspring handheld can send data through the module instead of the serial cradle, with no changes to existing software.



9. Source Documentation

"TCP/IP Illustrated Volume I: The Protocols" by W. Richard Stevens provides a very thorough review of how components of the protocol stack work together. ISBN 0-201-63346-9.

Handspring Development Kit Manual
(Handspring documentation: http://www.handspring.com/developers/documentation.jhtml)

AN03: Diagnostic Reference Module
(Handspring application notes: http://www.handspring.com/developers/tech_notes.jhtml)

Diagnostic Module schematics can be found in AN01: "*Testing your Springboard Module with your Visor*."

Palm OS® Programmer's Companion
Palm OS® SDK Reference
Palm OS® Programming Development Tools Guide
(Palm documentation can be found at http://www.Palm OS.com/dev/tech/docs/)

10. History

Date	Revision #	Description of changes
13 Dec 00	1.01	Reformat.
18 Sep 00	1.00	Initial release.
22 Jan 02	1.02	Change Title of document and some reference… more update to come in next rev… including the communicator product line