



Table of Contents

1.	Keyboard Support via Remote UI.....	2
1.1.	Remote UI Packet Description.....	2
1.2.	Remote UI Packet Header	3
1.3.	Remote UI Packet Body	4
1.4.	CRC Computation	5
1.5.	Remote UI Packet Example	6

Change History		
Revision	Date	Description
1.0	12 Mar 2001	Initial Release

List of Acronyms	
CRC	Cyclic Redundancy Check
RXD	Receive Data
TXD	Transmit Data
UI	User Interface

Handspring™, Visor™, Springboard™, and the Handspring and Springboard logos are trademarks or registered trademarks of Handspring, Inc. © 2001 Handspring, Inc.

1. Keyboard Support via Remote UI

The handheld platform supports character input from an external keyboard or pen-based device through the Cradle Connector located at the bottom of the handheld device. Grounding the KBD* pin on the Cradle Connector indicates to the handheld processor that:

- A keyboard or other remote UI device is present on the Cradle Connector, and,
- Incoming serial data packets on the RXD pin on the Cradle Connector should be interpreted as described by this document.

Note that the Cradle Connector does not include hardware signaling for buffer overflow conditions within the handheld device.

Remote UI is supported in all existing versions of Palm OS; for more information please see the *Palm OS Programmer's Companion* at <http://www.palmos.com/dev/tech/docs/>.

1.1. Remote UI Packet Description

As long the KBD* pin on the Cradle Connector is held low, the handheld will receive incoming data packets on RXD and interpret them as Remote UI Packets. Remote UI Packets have three sections: a header, a body, and a CRC, as shown in Table 1.

Table 1: Remote UI Packet

Header	Body	CRC
10 (bytes)	16 (bytes)	2 (bytes)

The Remote UI Packet structure is flexible enough to support remote input from a variety of devices, but for software simplification, most of these fields can be hardcoded for keyboard-specific input.

Table 2 describes in detail the fields in the header, body, and CRC sections.

Table 2: Remote UI Packet Fields

Field	Data Length (in Bytes)	Parameter Name	Value	Comment
Header Fields				
1	2	signature1	0xBEEF	Indicates serial link packet.
2	1	signature2	0xED	Indicates serial link packet.
3	1	dest	0x02	Indicates remote UI serial link packet.
4	1	src	0x02	Indicates remote UI serial link packet. Typically used as "return address" for response messages. Not required for keyboard input.
5	1	type	0x00	Indicates system packet type to the handheld.
6	2	bodySize	calculated	Size of body in bytes.
7	1	transactionID	calculated	Increment by one for each new message. Not required for keyboard input (will not be checked).

Field	Data Length (in Bytes)	Parameter Name	Value	Comment
8	1	checksum	calculated	8-bit sum of header fields NOT including this field.
Data Fields				
9	1	command	0x0D	Indicates that input is a remote event.
10	1	filler	don't care	For word alignment.
11	1	penDown	0 or 1	Indicates pen event. Reset to 0 for keyboard input.
12	1	filler	don't care	For word alignment.
13	2	penX	0	Pen X coordinate. Reset to 0 for keyboard input.
14	2	penY	0	Pen Y coordinate. Reset to 0 for keyboard input.
15	1	filler	don't care	For word alignment.
16	1	keyPress	0x01	Indicates a key has been pressed.
17	2	keyModifier	lookup	Modifier bits (shift, control, etc.) Bitmapped to this 16-bit field.
18	2	keyAscii	lookup	Palm OS keycodes (see <code>chars.h</code>)
19	2	keyCode	0x00	Reserved - Always set to 0.
CRC Field				
20	2	CRC	calculated	Computed using table method; see Section 1.4, " CRC Computation ," for more information. Uses big-endian byte ordering.

The following subsections provide more details on the contents of the header, body, and CRC.

1.2. Remote UI Packet Header

The Remote UI Packet header consists of eight fields as shown in Table 3. Each field consists of the number of bytes indicated.

Table 3: Remote UI Packet Header

signature1	signature2	dest	src	type	bodySize	transactionID	checksum
2 (bytes)	1	1	1	1	2	1	1

The first two fields (`signature1` and `signature2`) contain a predefined code that indicates to the handheld that the incoming packet is a serial link packet.

The `dest` and `src` fields refer to the logical socket used for communication for remote UI. For keyboard applications, these fields are both set to 0x02.

The `type` field indicates that this packet is a system packet. It is hardcoded to 0x00.

The 16-bit `bodySize` field must contain the size of the body portion of the packet (in bytes). Do not include the size of the header or CRC bytes in this calculation.

The `transactionID` field is a running message counter, and is typically used for two-way communications over the serial port. Reply messages are tagged with the `transactionID` of the original message. Because the handheld does not send responses to keyboard packets, this field is not used and can be set to any value. However, other types of remote UI devices should increment `transactionID` by one for each packet sent.

The checksum value is a simple eight-bit addition of the bytes in the header. If a checksum mismatch occurs, the handheld searches all incoming data bytes for `signature1` in order to resynchronize to the sender. Sample code to generate a checksum is shown below.

```

/*****
 * SlkChecksum      PrvCalcHdrChecksum(Checksum SlkChecksum, BytePtr bufP,
 *                               Long count);
 *
 * Computes the 16-bit checksum of bytes in a buffer.
 *
 * Arguments:
 *      SlkChecksum start      -- starting checksum value
 *      BytePtr bufP          -- ptr to the data buffer
 *      Long count            -- number of bytes in buffer
 *
 * Returns:
 *      8-bit checksum of the data
 *
 *****/
static SlkPktHeaderChecksum
PrvCalcHdrChecksum(SlkPktHeaderChecksum start, BytePtr bufP, Long count)
{
    // The compiler produces the fastest code with a while(--count) loop...
    do {
        start += *bufP++;
    } while(--count);

    return( start );
}

```

1.3. Remote UI Packet Body

Table 4 shows the Remote UI Packet body. Each field consists of the number of bytes indicated.

Table 4: Remote UI Packet Body

command	filler	pen Down	filler	penX	penY	filler	key Press	key Modifier	keyAscii	key Code
1 (byte)	1	1	1	2	2	1	1	2	2	2

The `command` field is always set to 0x0D, indicating that the packet contains remote event data. The `penDown`, `penX`, and `penY` fields define pen events; they are all reset to 0 for keyboard input. The `keyPress` field contains a flag that indicates a keypress has occurred on the remote UI device. The `keyModifier` and `keyAscii` fields define the keypress. The `keyCode` field is reserved for future purposes and should always be set to 0x0.

The `keyAscii` values supported by Palm OS are defined in the `chars.h` include file within the Palm OS source code. The `keyModifier` values are defined in the Palm OS source file `event.h`; they are also listed in Table 5 below.

Table 5: Key Modifiers for Palm OS

Key	KeyModifier Field Values
shiftKey	0x0001
capsLock	0x0002
numLock	0x0004
commandKey	0x0008
optionKey	0x0010
controlKey	0x0020
autoRepeatKey	0x0040
doubleTapKey	0x0080
poweredOnKey	0x0100
appEvtHookKey	0x0200
libEvtHookKey	0x0400

1.4. CRC Computation

The 16-bit cyclic redundancy check (CRC) value is calculated using the contents of both the header and the body. Note that the handheld device uses big-endian byte ordering for computing a CRC, so keyboards must generate CRCs using big-endian byte ordering. CRCs are computed using the table look-up method. Source code to generate CRCs is shown in the example below.

If the handheld detects a bad CRC, the packet is ignored, but no error response is sent back to the keyboard.

```

/*****
* FUNCTION: Crc16CalcBlock
*
* DESCRIPTION: Calculate the 16-bit CRC of a data block using the table lookup method.
*
* PARAMETERS:
*     bufP          -- pointer to the data buffer;
*     count         -- the number of bytes in the buffer;
*     crc           -- the seed crc value; pass 0 the first time
*                   -- function is called, pass in new crc result
*                   -- as more data is added to packet and crc is updated.
* RETURNS:
*     A 16-bit CRC for the data buffer.
*
*****/
Word Crc16Calc (VoidPtr bufP, Word count, Word crc)
{
    register BytePtr byteP = (BytePtr)bufP;
    WordPtr crctt          = (WordPtr)crctt_16;           // CRC
    translation table

```

```
//
// Calculate the 16 bit CRC using the table lookup method.
//
    if ( count ) {
        do {
            crc = (crc << 8) ^ crctt[ (Byte)((crc >> 8) ^ *byteP++) ];
        } while ( --count );
    }

    return( crc & 0xffff );
}

// This is the lookup table used when performing the 16-bit CRC calculation. //

static Word crctt_16[ 256 ] =
{
0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
0xFF9F, 0xEFBE, 0xDFDD, 0-CFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};
```

1.5. Remote UI Packet Example

The following example shows a hex dump of a packet representing the keystroke ‘a’ (lowercase ‘a’):

Sample Packet

BE	EF	ED	02	02	00	00	10
02	B0	0D	CC	00	CC	00	00
00	00	01	CC	00	00	00	61
00	00	2C	D8				

Table 6 categorizes the above data into the respective packet fields.

Table 6: Packet Example Breakdown

Header	
BE EF	signature1
ED	signature2
02	dest
02	src
00	type
0010	bodySize
02	transactionID
B0	checksum
Body	
0D	command
CC	filler
00	penDown
CC	filler
00 00	penX
00 00	penY
01	keyPress
CC	filler
00 00	keyModifier
00 61	keyAscii
00 00	keyCode
Checksum	
2C D8	CRC