Data Link USB
Communication Protocol and Database Design Guide

Specification Number: 1-S-XXXX-G

March 12, 2004
Timex Corporation

DOCUMENT REVISION HISTORY

REVISION: A	DATE: 2/04/2004	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
All	Created document.

REVISION: B	DATE: 2/10/2004	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
7	Modified ACD and ACB sections
4	Added code on reading from the HID driver

REVISION: C	DATE: 2/10/2004	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
7	Modified ACD and ACB sections
4	Added code on reading from the HID driver

REVISION: D	DATE: 2/12/2004	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
Various	Various edits

REVISION: E	DATE: 2/23/2004	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
2-4	Rearranged some of the sections so that they flow better
71	Added missing mode data to ICB table

REVISION: F	DATE: 3/11/2004	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
40	Fixed typos in Chrono section
42	Fixed incorrect arrangement of items in Chrono section
57	Fixed typos
Various	Added sample code and more info for melodies

REVISION: G	DATE: 3/12/2004	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
18	Added some more info for melodies
Various	Fixed some issues with some of the tables

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Scope

This document describes how to communicate with the Timex Data Link USB watch, as well as the format of the data that is stored on the watch. This document may be used to create new application software which may run on Linux, Apple Macintosh, or any other computer system that supports HID USB devices.

1.2. System Background

The USB Data Link watch is a product developed by Timex Corporation that lets you store various types of information on your wrist, including appointments, alarms, contacts, and notes as an example. The watch connects to the PC using the supplied interface cable.

The standard software that comes with the watch communicates with the TUCP Packager DLL which in turn handles communication to and from the watch. This DLL, the heart of the system, gives developers an API set in order to add and remove data from the watch. When a send is performed, the packager puts all of the data in the watch format and sends the information to the USB driver.

1.3. Purpose

The purpose of this document is to describe the Timex Universal Serial Bus (USB) Communications Protocol (referred to as TUCP in future references), and how it is transmitted using the USB protocol. The protocol is intended to provide a reliable communications link between the PC and the device. The protocol has been implemented with the premise that the PC has intimate knowledge of the device characteristics. This allows for a minimum number of protocol messages.

Some of the low level system details are outlined here, such as PC to device USB interaction, and interaction between various microcontrollers within the device. This interaction is described in further detail within the communications application software and hardware design description documents for the various products.

1.4. System Operation

The TUCP protocol will support a 2-way communications link between a device and a PC. The communication link utilizes the USB port that has become common on PCs. There will be a PC based program used to configure various types of data that will be downloaded to the device from the PC. There shall also be the capability to upload data from the device. This 2-way communications link provides for a various methods of synchronizing data between the PC and the device. In order to establish the communication link, a special USB cable designed for the device is used. Once all of the data has been entered, the user will connect the device to an available USB port on the PC. The connection is automatically detected on both the PC and device. The type of data transfer configured by the user can then begin. When the transfer is complete, the user will be notified.

2. COMMUNICATION SPECIFICS

2.1. Communications Flow

After the device has been connected to the PC, and the main microcontroller has indicated to the USB microcontroller that USB communications can take place, the USB microcontroller can establish its presence on the bus. The host PC then attempts to enumerate and configure the USB microcontroller within the device. After this process is complete, the PC will transmit/receive all message packets until:

- The "Communication Process Complete" protocol message is transmitted to the device.
- A receive/transmit timeout occurs during the download/upload process within the device.
- A non-recoverable error occurs during the download process.

Communications between the device and the PC will then stop. The device should be left in a useable state regardless of the condition that caused communications to stop.

The start of the communications session will determine if the PC and device are allowed to communicate. The protocol provides packets for this purpose. The packets that follow will determine the flow of data relative to the PC. The PC originates the request, and will transfer data in the appropriate direction with the device. The USB microcontroller will obtain the direction of data flow from the PC, and will transfer data in the correct direction with the main microcontroller. The main microcontroller, which has received the initial request, will setup a data transfer in the appropriate direction based on this request. The data transfer source or destination within the device has also been determined (RAM or EEPROM). The last packet sent should be the “Communication Process Complete” packet, which indicates the end of communications.

3. HW MODULE

The HW module within the main device microcontroller firmware contains the low-level interrupt handlers used to transfer data with the USB microcontroller. Within these interrupt handlers a byte will be transferred between the appropriate hardware register, and TUCP buffer. The buffer is designed to hold one TUCP protocol command, and is be shared by both transmit and receive operations. The synchronous serial transfer mechanism is used between the USB microcontroller and main microcontroller.

4. CORE MODULE

The CORE module is responsible for getting each byte received by the HW module and forming it into packets for a PC USB OUT transaction. It is also responsible for notifying the COMM module about various events related to communications. Examples include TUCP packet availability, and when a receive error occurs. When a PC USB IN transaction is taking place, the CORE is not involved in the transaction. It only provides utility functions for forming the packet checksum. The COMM module is responsible for starting the transmission, and HW module will continue the transmission.

5. COMM MODULE

The COMM module is responsible for reading the packet found in the TUCP buffer, and processing the command called out in the packet for a PC USB OUT transaction. When a PC USB IN transaction is taking place, it is responsible for storing a packet in the TUCP buffer, and informing the HW module that a packet needs to be transmitted. The COMM module forms the device User Interface (UI). The UI is used to notify the user of the communication status.

6. RECEIVED MESSAGE SUMMARY

The following table summarizes the TUCP protocol messages that may be received by COMM mode. These messages are broken down into various classes of messages as shown in the table, and may be internally or externally generated. The implementation of the protocol messages requires an intimate knowledge of the device by the PC SW. The command set has been optimized based on this condition.

Class Name	CMD Value	Description
Internally Generated Messages (Main MCU)		
Transceiver Notification Messages	0x00	Transmission Receive Error
Externally Generated Messages (PC)		
Communication Session Protocol Messages	0x01	Device Information Request
	0x02	Communication Process Complete
System Session Protocol Messages	0x03	Delete All Applications
	0x04	Beep
	0x05	Idle
	0x06	Call an Absolute Address
	0x07	Write to TOD Time Zone Data
Application Initialization Protocol Messages	0x08	Application Initialization Internal
	0x09	Application Initialization External
Single and Multiple Packet Transfer Messages	0x0A	Multiple Packet Write Started
	0x0B	Write to Absolute Address
	0x0C	Read from Absolute Address

7. HID (HUMAN INTERFACE DEVICE)

The HID (Human Interface Device) is a standard driver that ships with Microsoft Windows that the packager uses to communicate with the Data Link USB watch. For other operating systems, this driver may or may not be available.

This document does not cover communicating with the HID driver. For more information on HID Drivers, Timex recommends USB Complete (2nd Edition) by Jan Axelson. Sample code for the book is available at http://www.lvr.com/hidpage.htm.

The device uses a single HID input/output report structure consisting of 8 bytes. The usages/collections/report-descriptor-in-general can be ignored.

7.1. Vendor ID

All USB devices have a vendor and product ID that is used to verify that the device is present in the USB chain. The Timex Data Link USB vendor ID and product ID are defined as:

```
const UINT VendorID = 0x0cc2;
const UINT ProductID = 0xd700;
```

7.2. Writing to the HID driver

The packager that is used on the Windows platform links to the HID.LIB file created by Microsoft Corporation. The following code shows how the packager sends 8 byte packets to the HID driver for

download to the watch. Basically, a TUCP packet may be up to 255 bytes. However, writing to the watch is done 8 bytes at a time as shown below.

```
DWORD CUSBComm::SendDLPacket(LPBYTE lpPacket)
{
    DWORD dwResult = TIMEX_SUCCESS;
    int   nNumChars = MAX_USB_PACKET_SIZE-1; // MAX_USB_PACKET_SIZE = 9

    for (int i = 0; i < lpPacket[0]; i=i+nNumChars)
    {
        dwResult = SendUSBPacket(lpPacket+i, (BYTE)nNumChars);
        if (dwResult)
                break;
    }
    return dwResult;
}

DWORD SendUSBPacket(LPBYTE cPacket, BYTE bPacketLen)
{
    BYTE  baReport[MAX_DL_PACKET_SIZE];
    DWORD dwResult = 0;
    DWORD dwNumBytesWritten;

    // initialize the outgoing buffer to zeros
    memset(baReport, 0, sizeof(baReport));
    // populate the contents of the USB packet
    // first byte (report type) must be 0
    memcpy(&baReport[1], cPacket, bPacketLen);


    // we will try to send the packet up to thrice if necessary
    char szCode[128];
    for (int i = 0; i < NUM_RETRIES; i++)
    {
        // Send a report to the device.
        CancelIo(m_hReadDevice);
        dwResult = WriteFile(m_hWriteDevice, baReport, bPacketLen+1, &dwNumBytesWritten,
                        (LPOVERLAPPED)&m_HIDOverlapped);
        if (dwResult == 0) // there was a problem
                dwResult = GetLastError(); // error found - abort
        else
        {
                dwResult = TIMEX_SUCCESS;
                break;
        }
    }

    return dwResult;
}

#define TIMEX_SUCCESS                       0       // No error
#define DL_851_MAX_PACKET_SIZE          255      // Max packet length
#define DL_851_COMM_ACK                 0x0D
#define DL_851_COMM_NACK                0xFF

typedef struct _PACKETDATA
{
    struct _PACKETDATA *NextLink;
    HGLOBAL MemHandle;
    BYTE baPacket[DL_851_MAX_PACKET_SIZE];   // The packet data
} PACKETDATA, *LPPACKETDATA;

int SendAndReceivePacket(LPPACKETDATA lpDLPacket, LPPACKETDATA lpDLPacketUp)
{
    int nRC = TIMEX_SUCCESS;

    if (lpDLPacket)
    {
        // we will try to send the packet up to thrice if necessary
        int i;
```

```
        for (i=0; i<3; i++)
{

        nRC =SendDLPacket(baPacket);

        if (nRC != TIMEX_SUCCESS)
        break;

        // read the response packet from the device
        BYTE baReturnPacket[DL_851_MAX_PACKET_SIZE];
        memset(baReturnPacket, 0, sizeof(baReturnPacket));
                nRC = ReadDLPacket(baReturnPacket);

        // see if we got a good packet back
        if ((nRC == TIMEX_SUCCESS) &&
                // check the length of the packet
                //(baReturnPacket[0] >= 3 && baReturnPacket[0] <=
                        DL_851_MAX_PACKET_SIZE) &&
                (baReturnPacket[0] >= 3 && baReturnPacket[0] <=
                        DL_851_MAX_READ_PACKET_SIZE) &&
        // check the checksum
        (TUDLVerifyChecksum(baReturnPacket, (WORD)baReturnPacket[0])) &&
        // is it an ACK
        (baReturnPacket[1] == DL_851_COMM_ACK) &&
                // is it a response to the command we sent out
        (baReturnPacket[2] == baPacket[1]))
        {
                // process the packet here
                // if it is the ICB then store it in the ICB array
                if (baReturnPacket[2] == DL_851_COMM_DEV_INFO_REQ)
                {
                        memcpy(baICB, &baReturnPacket[3], DL_851_EE_ICB_SIZE);
                        // get the EEPROM size from the ICB
                        //m_dwAddressBottom = MAKELONG(MAKEWORD(baICB[16],
                                baICB[17]), 0) - m_dwPeriodicTaskSize;
                        // if it is the binary upload, store it in the binary
                        // upload list
                }
                // see if we got a good nack packet
                else if ((nRC == TIMEX_SUCCESS) &&
                // check the length of the packet
                (baReturnPacket[0] >= 3 && baReturnPacket[0] <=
                        DL_851_MAX_PACKET_SIZE) &&
                // check the checksum
                (TUDLVerifyChecksum(baReturnPacket, (WORD)baReturnPacket[0])) &&
                // is it a NACK
                (baReturnPacket[1] == DL_851_COMM_NACK))
                {
                        // check the error type
                        if (baReturnPacket[2] == 0x00 || baReturnPacket[2] == 0x01
                                || baReturnPacket[2] == 0x04)
                        {
                        // retry
                        if (i == 2) // this was the last try; return with an error
                        {
                                if (baReturnPacket[2] == 0x00)
                                        nRC = TIMEX_ERR_COMM_CHECKSUM;
                                else if (baReturnPacket[2] == 0x01)
                                        nRC = TIMEX_ERR_COMM_PACKET_LEN;
                                else if (baReturnPacket[2] == 0x04)
                                        nRC = TIMEX_ERR_COMM_TIMEOUT;
                                else
                                        nRC = TIMEX_ERR_COMM;
                                break;
                        }
                }
                else
                {
                        // non-recoverable error, don't retry
                        if (baReturnPacket[2] == 0x02)
                                nRC = TIMEX_ERR_COMM_DEV_INFO;
```

```
                                else if (baReturnPacket[2] == 0x03)
                                        nRC = TIMEX_ERR_COMM_DEV_MISMATCH;
                                else
                                        nRC = TIMEX_ERR_COMM;
                                break;
                        }
                }
                else // invalid response packet, non-recoverable error, don't retry
                {
                        nRC = TIMEX_ERR_COMM;
                        break;
                }
        }
    }

    // close handle to the device
    if (nRC)
        CloseDevHandle();

    return nRC;
}
```

7.3. Reading from the HID driver

Reading from the HID driver works similalrly to writing. Packets are received 8 bytes at a time. Please see the sample code below:

```
DWORD ReadDLPacket(LPBYTE lpReturnPacket)
{
    DWORD dwResult;
    BYTE  baInReport[MAX_USB_PACKET_SIZE];

    // initialize the receiving buffer to zeros
    memset(baInReport, 0, sizeof(baInReport));
    // set the report number
    //baInReport[0] = 5;
    // get the first packet from the device
    dwResult = ReadUSBPacket(baInReport, MAX_USB_PACKET_SIZE);
    if (dwResult == TIMEX_READ_FILE_TIMEOUT)
    {
        baInReport[1] = 0x04;
        baInReport[2] = 0xFF;
        baInReport[3] = 0x04;
        baInReport[4] = 0xF9;
        dwResult = TIMEX_SUCCESS;
    }
    else
    {
        // copy the result to the buffer passed from the calling function
        memcpy(&lpReturnPacket[0], &baInReport[1], MAX_USB_PACKET_SIZE-1);

        // see how many more packets we need to get
        int nNumPackets = (baInReport[1]-1) / (MAX_USB_PACKET_SIZE-1);
        for (int i = 1; (i <= nNumPackets) && (dwResult == 0); i++)
        {
                // initialize the receiving buffer to zeros
                memset(baInReport, 0, sizeof(baInReport));
                // get the next packet from the device
                dwResult = ReadUSBPacket(baInReport, MAX_USB_PACKET_SIZE);
                if (dwResult == TIMEX_READ_FILE_TIMEOUT)
                {
                        baInReport[1] = 0x04;
                        baInReport[2] = 0xFF;
                        baInReport[3] = 0x04;
                        baInReport[4] = 0xF9;
                        i = nNumPackets + 1;
                        dwResult = TIMEX_SUCCESS;
```

```
                }
                else
                {
                        // copy the result to the buffer passed from the calling function
                        int nPos = i*(MAX_USB_PACKET_SIZE-1);
                        memcpy(&lpReturnPacket[nPos], &baInReport[1], MAX_USB_PACKET_SIZE-1);
                }
        }
    }

    return dwResult;
}

//=========================================================================

DWORD ReadUSBPacket(LPBYTE lpPacket, BYTE bPacketLen)
{
    DWORD dwResult;
    DWORD dwTimeout;
    DWORD NumberOfBytesRead;

    // read a report from the device
    dwResult = ReadFile(m_hReadDevice, lpPacket, bPacketLen, &NumberOfBytesRead,
               (LPOVERLAPPED)&m_HIDOverlapped);

    dwTimeout = WaitForSingleObject(m_hEventObject, COMM_READ_TIMEOUT_PERIOD);
    if (dwTimeout == WAIT_TIMEOUT)
    {
        lpPacket[0] = 0x00;
        lpPacket[1] = 0x04;
        lpPacket[2] = 0xFF;
        lpPacket[3] = 0x04;
        lpPacket[4] = 0xF9;
        CancelIo(m_hReadDevice);
        dwResult = TIMEX_READ_FILE_TIMEOUT;
    }
    else
    {
        if (!dwResult && GetLastError() != ERROR_IO_PENDING) // there was a problem
        {
                dwResult = GetLastError();
        }
        else // readfile completed ok
        {
                dwResult = TIMEX_SUCCESS;
        }
    }

    return dwResult;
}
```

7.4. System MAP Table

The watch has a table in ROM that is called the System Map Table. The Map Table should be read first before any other communication takes place. The Map Table is read from ROM at address 0x028 (40 decimal) and contains 13 words of information. The following information is read from the Map Table:

- Heap Top Address – Location of where the top of the internal heap is stored
- Heap Top Address (After Reset) – Location of where the top of the internal heap is stored after reset
- Heap Bottom Address (After Reset) – Location of where the bottom of the internal heap is stored after reset
- Overlay – size of the overlay area
- Mode List – Base address of the mode list
- *Application Configuration Data – Address of the Application Configuration Data

- *Application Control Block – Address of the application control block
- *Option Data Address – System option data structure
- Periodic Task Control – Base address of periodic task
- *Melody Address Table – Base address of the melody table
- *Melody Init – Location of the melody audio init routine
- *Popup Custom Melody – Location of overlay area for user melody
- Melody Table – Default ROM based melody table

Note – Only those Items shown with an asterisk are read by the Timex Packager. Other fields are ignored on the PC

7.5. Application Configuration Data (ACD)

The application configuration data block (as defined in the previous section) consists of 16 bytes of data. Each item refers to a mode in the watch. Sixteen modes are supported in the watch, but one is COMM mode and the other is Time of Day, which are always present.

Displayed below are the bit definitions for each byte of the application configuration data structure:

Bit	Bit Name	Description
0	Reserved	
1	Reserved	
2	Reserved	
3	Reserved	
4	Database Modified	0 = Database not modified 1 = Database modified by user
5	Reserved	
6	Password Required	0 = Password not required 1 = Password required for access
7	User Specified Mode Name	0 = use default mode name 1 = Mode name located in EEPROM

7.6. Application Control Block (ACB)

The Application Control Block (ACB) is composed of 16 data structures with each structure having 14 bytes. Along with the Application Configuration Data, the ACB provides information on the location of the critical components of an application. These components are the following:

- Application ID (16-bytes)
- Application System Data Address (16-bit)
- Application Database Data Address (16-bit)
- Application Mode State Manager Address (16-bit)
- Application Refresh Handler Address (16-bit)
- Application Mode Banner Message Address (16-bit)
- Address of Code Block in EEPROM (16-bit)

1. Application ID (16 bit)

The Application ID has two parameters:

- Application Type (Byte 0)
- Application Instance (Byte 1)

The Application Type is used by the core to identify the application currently active in the system. It is used to search for the active application list to match the criteria for the following operations:

- Peek at Appointment Type Application
- Peek at Occasion Type Application
- Day Update occurred for appointment and occasion type application
- Hour update occurred for appointment and occasion type application
- Update primary mode LCD icon for occasion type application
- Etc.

By default, the first application type instantiated by the system will have an Application Instance of 0. If another application of the same type and instance number is instantiated by the system, the system will generate an error.

It is critical to properly assign the appropriate Application Type to an application. The following table shows the predefined Application Types supported by the system.

Code	Application Type
0x00	System
0x01	Communication
0x02	Option
0x10	TOD
0x11	Date
0x20	Chrono
0x21	Timer
0x22	Synchro Timer
0x23	Counter
0x40	Contact
0x50	Task
0x60	Notes
0x70	Schedule
0x80	Tide
0x90	Demo
0xA0	Game

0xE0	Alarm
0xE1	Appointment
0xE2	Occasion

Codes with values from 0xE0 to 0xFF are reserved for application types that are dependent on the primary time zone. If the primary time zone is modified by the user, the application's background handler will be called to update its variables and resources to reflect the new time.

2. Application System Data Address (16-bit)

The Application System Data (ASD) of all applications are located in the internal heap. The ASD address is a 16-bit absolute address in internal memory. If there is insufficient space on the heap for the heap requirements for a new application, then the core will not install the application.

The ASD stores all the application specific data that is used for the lifetime of the application.

The application state handler can access the base address of the ASD from the Application Control Block by using the following macro call:

```
CORE_SET_HL_TO_ASD_ADDRESS;
```

When an application becomes the forground application, the core will copy the ASD Address in the control block in the variable **`CORECurrentASDAddress`**.

Prior to executing the Refresh Handler of an application, the core will copy the ASD Address in the control block in the variable **`COREBackgroundASDAddress`**.

3. Application Database Data Address (16-bit)

The Application Database Data (ADD) is where an application stores its database. It can be located in either internal or external heap by setting the correct status for the flag **`bCOREACDDatabaseDataLocation`** in the Application Configuration Byte.

If located in internal heap, the ADD address is the absolute address in internal memory. It is recommended that the ADD data located in internal memory have minimal memory requirements and not subject to size changes during the entire application life. The flag **`bCOREACDDatabaseDataLocation`** is set to 0.

For database that have big memory requirements and changes size during PC-COMM session, the database should be located in external memory. The flag **`bCOREACDDatabaseDataLocation`** is set to 1. The data stored in the Application Database Data Address of the ACB is the absolute address in EEPROM where the application database is stored.

The application state handler can access the base address of the ASD from the Application Control Block by using the following macro call:

```
CORE_SET_HL_TO_ADD_ADDRESS;
```

When an application becomes the forground application, the core will copy the ASD Address in the control block in the variable **`CORECurrentADDAddress`**.

Prior to executing the Refresh Handler of an application, the core will copy the ASD Address in the control block in the variable **`COREBackgroundADDAddress`**.

4. Application Mode State Manager Address

The Application Mode State Manager Address tells the core the location in internal heap where the code for the state manager is located. The State Manager is usually a table that uses the data in **`CORECurrentState`** to jump and execute the correct state handler.

For EEPROM based applications, where only one state is loaded from EEPROM for processing, there is no need for a State Manager. The address in the ACB is the base address of the state handler.

5. Application Refresh Handler Address

The Application Refresh Handler Address tells the core the location in internal heap where the code for the refresh handler is located.

For EEPROM based applications, this is the address in the overlay area where the refresh handler routine is located. The refresh handler is part of the common code block of an application.

Background events or task are passed to the refresh handler for processing. It will use the following variables to complete its tasks:

- **`COREBackgroundEvent`**
- **`COREBackgroundAppIndex`**
- **`COREBackgroundASDAddress`**
- **`COREBackgroundADDAddress`**

6. Application Mode Banner Message Address (16-bit)

This data structure indicates the absolute address in internal memory where the mode banner message of the application is stored. The message follows a specified format for mode banner messages.

If the Application Configuration Data flag **`bCOREACDUserSpecifiedModeName`** is a 1, then the mode banner name is located in the Mode Banner Message Database.

7. Address of Code Block in EEPROM

If the Application Configuration Data flag **`bCOREACDCodeLocation`** is 1, then this address in the ACB indicates the absolute address in EEPROM where the application code block is located. The core will use this info during mode and state change operations.

8. COMMUNICATING WITH THE WATCH

The following pseudo-code shows how the typical operation of reading or writing data to and from the watch would occur.

8.1. Read Operation

Create Device Information Request Packet
Send the packet
Receive packet

Read the last session ID
Read the System Map Table

Read Application Configuration Data

Read Application Control Block

For each mode in the watch that is readable
    Check Application Control Block – See if modified
    Read database at location specified in ACB

Create Transmission complete packet
Send the packet

8.2. Write Operation

Create Device Information Request Packet
Send the packet
Receive packet

Read the last session ID
Read the System Map Table

Read Application Configuration Data
Read Application Control Block

Create Time of Day packet
Send the packet

Create Option packet
Send the packet

Create the Sound packet
Send the packet

Create Multi Write Start packet
Send the packet

Create the Delete All Apps packet
Send the packet

For each application
    Package the database
    Send the packet
    If app is a WristApp, package the code
        Send the packet
    Create the Application internal intiailzation packet
    Send the packet

Write the session ID to the watch

Create Transmission complete packet
Send the packet

9. SYSTEM SPECIFICS

9.1. System Interaction

Effective data transmission occurs at approximately 3500 bits-per-second (baud) between the PC Host and the main device microcontroller using an unloaded USB link. The measurement was taken during an actual download. This included overhead due to the protocol, initialization of applications, and the data for the applications. The physical transmission speed across the USB bus and the synchronous serial bus is faster than indicated, but due to overhead the effective data rate will be less than the actual speeds.

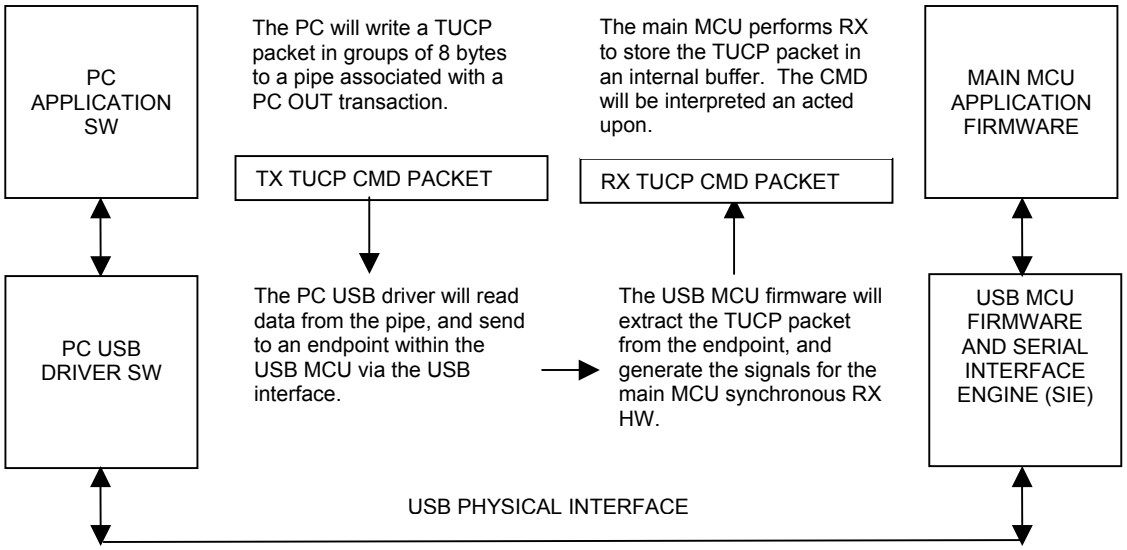
There are two levels of data flow. The low level contains the USB protocol with it's inherent flow of data between the PC USB driver, and USB microcontroller within the device. The top level contains the TUCP protocol with flow of data between the PC application software, and the main device microcontroller. The USB level is provided to get an external 2-way interface between the PC and the device by using a standard protocol. The TUCP level is provided to perform some action within the device by the PC application software.

Data is transferred bi-directionally on the USB bus by using the flow control mechanisms in place for this protocol. The USB protocol provides for handshaking between the PC and the device, error checking, and the capability for retransmission of errored data. The USB protocol will be used as the low-level transport mechanism. The TUCP protocol packets will be embedded within the USB protocol packets as data. The TUCP protocol is designed to service a request originating on the PC at the application level. The TUCP protocol has some error checking built in, but is provided just to check for the integrity of the data transfer between the device's USB microcontroller and its main microcontroller. The integrity of the transfer will be reported back to the PC application level after a transaction request has completed. The short physical link within the device has flow control, which is used to control data flow within the device. Since the physical link is short, error checking does not need to be as stringent as that for the USB link. Using all of these components data can be transferred reliably.

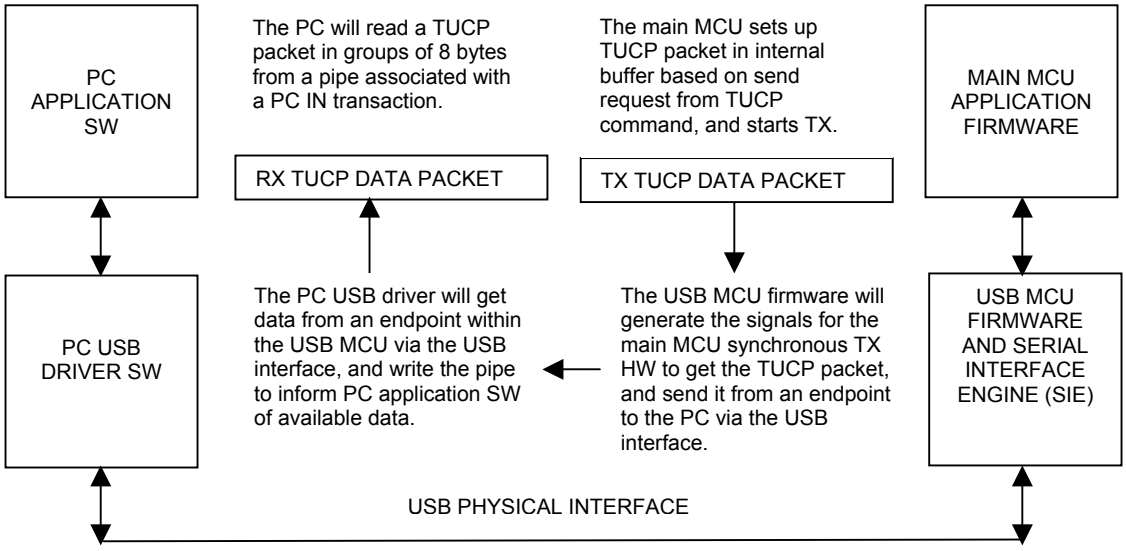
The USB microcontroller within the device is unpowered when the USB cable is detached. Once the cable is attached between the PC and device, the USB microcontroller derives its power from the USB bus power. The USB microcontroller initializes, and waits for an indication from the main microcontroller that PC communications can be perfomed. Once this indication is received the USB microntroller can interact with the PC (enumeration, device configuration, etc.). The interaction with the PC will also include a read/and or write of various items in the device. This is handled by embedding TUCP protocol commands packets within the USB protocol packets.
The first packet transmitted by the PC is a request for device information. The device information is used by the PC to determine if communication is allowed. The decision to communicate is made strictly on the PC side. In addition, the PC upon receiving the device information will adjust its communication with the device. The reasons for disallowal can be mismatches in the PC SW versus device firmware, an unsupported device, or a database mismatch. Subsequent packets will be as defined later in this document. If the data transfer is error free on the USB and within the internal transfers it may be used by the PC or device as specified by the protocol.

9.2. TUCP Packets Transmitted Using the USB Interface

The following diagrams outline some of the mechanisms that are used to transfer a TUCP packet between the PC application SW and the main device microcontroller. The SW and firmware interactions and the USB transport are not specified in detail, just the overall mechanisms.



EXAMPLE OF PC OUT TRANSACTION



EXAMPLE OF PC IN TRANSACTION

9.3. TUCP Packet

The TUCP packet contains hexadecimal data that instructs the COMM module running within the device to perform some action within the device. Some of the main actions are transferring application code/data, the ability to start internal code, or loading system melody data. The packet contents may ultimately be stored in device memory for subsequent use by the system code within the device. The device memory may be RAM or EEPROM based. The packet may also contain status information passed back to the PC by the device. Certain packets are internally generated, and are used to inform the COMM module that an error has been detected with the communications transfer. These error notification packets utilize the error checking capability built into the packets.

The TUCP packets are transmitted using one or more USB packets. There may be one or more TUCP packets required to perform an action. This will be outlined in the individual TUCP packet descriptions. Below is an example of how a TUCP packet is broken down.

The TUCP packet size was chosen based on the size of an EEPROM block, plus any overhead to transfer this block. The overhead may be the offset or address information, the application identifier, and data type being transferred. The diagram below shows the TUCP packet, along with size information for the various fields.

1 byte	1 byte	0 - 67 bytes	1 byte
Data Download/Upload to/from Watch			
Data Size	Command	Data Section	CHKSUM

3 - 70 bytes

Data Size	Specifies the size of the packet in bytes.
Command	Specifies the type of packet.
Data Section	Depending on the type of packet, this is the associated information that will be processed by the communications code within the device, and/or the data read/written from/to the device.
CHKSUM	The sum of all bytes from the start of the packet until the checksum byte, then 2's complemented. When checking this field, the sum of all bytes including the checksum should be zero for a valid transmission.

10. TRANSCIEVER NOTIFICATION MESSAGES

At any point during the communications session COMM mode can receive a transceiver notification message that contains the transmission receive error type. The device must be able to process the errors. The internally generated transceiver notification messages can be classified as recoverable errors provided that the communications link is still intact. Recoverable errors can be fixed by subsequent retransmissions, which are requested from the PC based on a NACK response to a packet.

Non-recoverable errors can't be fixed by subsequent retransmission. The non-recoverable errors should place the device in a known default state. The non-recoverable errors do not generate any messages internal or external to the device. The detection of theses errors is accomplished by utilizing the "Multiple Packet Write Started" packet in conjunction with error detection code within COMM mode. An example of a non-recoverable error is when a multiple packet database download is started, and the device is disconnected from the PC in the middle of this transaction.

11. APPLICATION DOWNLOAD

ROM/EEPROM based applications must be initialized in the device by doing a full download. This insures that the heap is properly cleaned up, and all other system data are set to its initial conditions. All applications previously present in the device will be deleted, and should be downloaded again if the user wishes them to remain in the device. Only the TOD and COMM modes are not affected by a full download.

If a full download is indicated, the PC must then send a TUCP command to clean up the heap, and leave the TOD and COMM modes enabled. This is accomplished by sending the *Delete All Applications* to make the heap ready for a full download. The PC will initialize an application in the device by sending the *Application Initialization External* or *Application Initialization Internal* packet, depending on whether the application is EEPROM or ROM based respectively. The PC may send the program code and database data in any order, but must send the *Multiple Packet Write Started* before the data is sent. This data is sent using the *Write to Absolute Address* packets with the appropriate target memory specified. These packets may be sent many times to download all the data characters comprising the code or data. The *Communication Process Complete* packet is sent after all applications and data have been sent.

Every time an application is deleted or disabled from the mode list, a full download must also be done. All other applications already present in the device are also deleted. If the user wishes to have these applications remain in the device, they must be downloaded again just like when they are enabled for the first time. Only the TOD and COMM mode data are not affected. To the device, deleting an application is like registering new applications so the same process must be done.

For non-recoverable errors during full or incremental downloads, all the applications are brought back into their POR default state, with COMM and TOD applications remaining unaffected. Any application that is not part of the default set will no longer be present in the device. The user is informed of this condition by the PC, and is given the option to attempt a retry. This will allow the device to be brought back to the state prior to the non-recoverable error.

12. DATABASE DOWNLOAD

The PC will start a database session with an application already present in the device by sending the *Multiple Packet Write Started* packet. This will indicate that the data that follows will need to be sent to completion. The PC will then send the *Write to Absolute Address* packets with the appropriate target memory specified. These packets may be sent many times to download all the data characters for the application. The *Communication Process Complete* packet is sent after all applications have had their databases modified.

13. MELODY DOWNLOAD

The system melody table download process is similar to downloading a database. The melody is considered as a special type of application with only a database that contains the actual melody data. The *Write to Absolute Address* packet is used to download the melody database, and system variables.

Prior to starting the melody database download, the system melodies are immediately changed to point to the ROM based melodies. This way, when an error occurs, valid system melodies are used. After the reception of the last *Write to Absolute Address* packet, the system melodies are pointed to the downloaded melody tables. These new melodies can be used.

The max size of the whole melody table is 384 bytes. Each individual melody for each event can be up to 36 bytes.

Some example code is shown below:

```
#define DL_851_EE_ADDR_MELODY     0x2c0
#define DL_851_COMM_CALL_ABS_ADR  0x06
#define DL_851_MAX_SOUNDS         9
#define DL_851_EE_MELODY_SIZE     (0x440 - 0x2c0) //0x180
#define DL_851_COMM_WRITE_ABS_ADR 0x0B
#define DL_851_SOUND_DB_OVERHEAD  0x08

int CreateSoundPackets()
{
    BYTE baPacket[DL_851_MAX_PACKET_SIZE];
    BYTE bIndex;
   int  i;

   // call audio init routine to initialize melody table to point to ROM table
   bIndex = 1;
    baPacket[bIndex++] = DL_851_COMM_CALL_ABS_ADR; // command
   WORD wAddress = m_waSystemMapTable[10];
    baPacket[bIndex++] = LOBYTE(wAddress); // abs address lo
    baPacket[bIndex++] = HIBYTE(wAddress); // abs address hi
    baPacket[0] = ++bIndex;   // store the size of the packet - include checksum
    // Attach the CRC
    baPacket[bIndex-1] = TUDLCalculateChecksum(baPacket, (BYTE)(baPacket[0]-1));
   int nRC = AddPacket(baPacket);

   BOOL boolUseDefaultSounds = TRUE;
   for (i = 0; i < DL_851_MAX_SOUNDS; i++)
   {
      if (m_lpSound->baSounds[i][0] != 0xFF)
      {
            boolUseDefaultSounds = FALSE;
            break;
      }
   }

   if (!boolUseDefaultSounds)
   {
      // create the melody database and download it to the appropriate location
      BYTE baBuf[DL_851_EE_MELODY_SIZE];
      CreateDatabaseSounds(baBuf);
      DWORD dwSize = (DWORD)(MAKEWORD(baBuf[2], baBuf[3]));
      nRC = AddBinaryIntData(DL_851_EEPROM, DL_851_EE_ADDR_MELODY, dwSize, baBuf);
      CreateBinaryIntPackets();
```

```
      // initialize the RAM melody table to point to a temporary RAM melody buffer
      for (i = 0; i < DL_851_MAX_SOUNDS; i++)
      {
            if (m_lpSound->baSounds[i][0] != 0xFF)
            {
                  bIndex = 1;
                  baPacket[bIndex++] = DL_851_COMM_WRITE_ABS_ADR; // command
                  wAddress = (WORD)(m_waSystemMapTable[9]+(i*2));
                  baPacket[bIndex++] = LOBYTE(wAddress); // abs address lo
                  baPacket[bIndex++] = HIBYTE(wAddress); // abs address hi
                  baPacket[bIndex++] = 0;  // memory type - 0=RAM
                  // write the melody table
                  wAddress = m_waSystemMapTable[11];
                  //for (i = 0; i < DL_851_MAX_SOUNDS; i++)
                  {
                        baPacket[bIndex++] = LOBYTE(wAddress); // abs address lo
                        baPacket[bIndex++] = HIBYTE(wAddress); // abs address hi
                  }

                  baPacket[0] = ++bIndex;   // store the size of the packet -
include checksum
                  // Attach the CRC
                  baPacket[bIndex-1] = TUDLCalculateChecksum(baPacket,
(BYTE)(baPacket[0]-1));
                  nRC = AddPacket(baPacket);
            }
      }
   }
    return nRC;
}

int CreateDatabaseSounds(LPBYTE lpBuf)
{
   int i;
   int nMaxRecSize = 0;

    // zero out the memory space
    memset(lpBuf, 0x00, DL_851_EE_MELODY_SIZE);

   // determine the maximum size of the record
    for (i = 0; i < DL_851_MAX_SOUNDS; i++)
      nMaxRecSize = max(m_lpSound->baSoundLen[i], nMaxRecSize);

    // start writing the db from the first record
    // leave placeholders for the header to be filled in later
    WORD wIndex = DL_851_SOUND_DB_OVERHEAD;

    // Write the sounds data
    for (i = 0; i < DL_851_MAX_SOUNDS; i++)
    {
      memcpy(lpBuf+wIndex, m_lpSound->baSounds[i], nMaxRecSize);
      wIndex = (WORD)(wIndex + nMaxRecSize);
   }

    // set up the db header
    WORD wDBSize = wIndex;
    // calculate the allocation size
    WORD wAllocSize = (WORD)(((int)((wDBSize-1)/DL_851_EE_PAGE_SIZE + 1)) *
DL_851_EE_PAGE_SIZE);

   wIndex = 0;
    // write alloc size
    lpBuf[wIndex++] = LOBYTE(wAllocSize);
```

```
    lpBuf[wIndex++] = HIBYTE(wAllocSize);
    // write db size
    lpBuf[wIndex++] = LOBYTE(wDBSize);
    lpBuf[wIndex++] = HIBYTE(wDBSize);
    // app specific header size
    lpBuf[wIndex++] = 3;
    // write total number of records
    lpBuf[wIndex++] = LOBYTE(DL_851_MAX_SOUNDS);
    lpBuf[wIndex++] = HIBYTE(DL_851_MAX_SOUNDS);
    // write timer record size
    lpBuf[wIndex++] = (BYTE)nMaxRecSize;

    // dump the db to a file here
    if (m_lpMain->m_szDatabasePath[0])
    {
        TCHAR szPath[MAX_PATH];
       sprintf(szPath, "%s\\sounds.bin", m_lpMain->m_szDatabasePath);
        m_lpMain->writeBinary(szPath, lpBuf, (DWORD)wAllocSize);
    }

    return 0;
}
```

14. OPTION DOWNLOAD

The Option packet is created by using a Write To Absolute Address packet. The code that is used to packetize the information is shown below. The address passed is the 8th index into the system map table.

```
int CreateOptionPacket(WORD wAddress)
{
   BYTE baPacket[DL_851_MAX_PACKET_SIZE];
   BYTE  bIndex = 1;

   baPacket[bIndex++] = DL_851_COMM_WRITE_ABS_ADR; // command
   baPacket[bIndex++] = LOBYTE(wAddress); // abs address lo
   baPacket[bIndex++] = HIBYTE(wAddress); // abs address hi
   baPacket[bIndex++] = 0;  // memory type - 0=RAM
   // write the status byte
   baPacket[bIndex] = 0;
   // nightmode status bits
   if (boolNightModeEnabled)
      baPacket[bIndex] = (BYTE)(BitSet(baPacket[bIndex], 0));
   if (boolNightModeAuto)
      baPacket[bIndex] = (BYTE)(BitSet(baPacket[bIndex], 1));
   // hourly chime status bits
   if (boolHourlyChimeEnabled)
      baPacket[bIndex] = (BYTE)(BitSet(baPacket[bIndex], 4));
   if (boolChimeAuto)
      baPacket[bIndex] = (BYTE)(BitSet(baPacket[bIndex], 5));
   // button beep status
   if (boolButtonBeepEnabled)
      baPacket[bIndex] = (BYTE)(BitSet(baPacket[bIndex], 7));
   bIndex++;
   // write the nightmode data
   baPacket[bIndex++] = bNightModeOnMin;
   baPacket[bIndex++] = bNightModeOnHour;
```

```
    baPacket[bIndex++] = bNightModeOffMin;
    baPacket[bIndex++] = bNightModeOffHour;
    // write the chime data
    baPacket[bIndex++] = 0;
    baPacket[bIndex++] = bChimeOnHour;
    baPacket[bIndex++] = 0;
    baPacket[bIndex++] = bChimeOffHour;
    // write the nightmode toggle duration
    baPacket[bIndex++] = bNightModeToggleDuration;
    // write the last editable character index
    baPacket[bIndex++] = bLastSetCharacter;
    // write the password
    baPacket[bIndex++] = baPassword[0];
    baPacket[bIndex++] = baPassword[1];
    // write the timeline status byte
    baPacket[bIndex] = 0;
    if (m_lpOption->boolApp1Timeline)
        baPacket[bIndex] = (BYTE)(BitSet(baPacket[bIndex], 7));
    if (m_lpOption->boolApp2Timeline)
        baPacket[bIndex] = (BYTE)(BitSet(baPacket[bIndex], 6));
    bIndex++;
    baPacket[bIndex++] = 8; // nightmode duration, always set to 8 hours

    baPacket[0] = ++bIndex;     // store the size of the packet - include checksum
    // Attach the CRC
    baPacket[bIndex-1] = TUDLCalculateChecksum(baPacket, (BYTE)(baPacket[0]-1));

    return AddPacket(baPacket);
}
```

15. UPLOAD AND DOWNLOAD EXAMPLES

The following sections give some examples of transactions that the PC can perform. The first group describes a download of application information. These include the initialization an instance of a ROM based application, download of a new database to an existing application, and the download of a melody table located in EEPROM. The second group describes an upload of application information. This includes an upload of a database from an existing application.
The examples show the initial handshaking that is performed at the start of a communication session. Some TUCP packets details are left out for the sake of clarity. These details are application specific data contained in the packet. The transfers show PC OUT, and the corresponding PC IN response. The PC IN response will be a simple TUCP ACK packet, or TUCP ACK packet with response data.

The examples show the format of the command/response packets. The format follows that which is defined in the Message Details section. The examples contain a mix of text strings describing a particular field or the data that is being transferred within that field, and actual hexadecimal values (e.g. packet or data length).

15.1. Initialize ROM Based Application

The following is an example of a communication session to initialize a single ROM based application, in addition to TOD, with the following application specific information: EEPROM database allocation size 128 bytes, first instance of this application, and application POR table offset is 10. The PC will first issue a sequence of commands to characterize the device, and setup any system related data. The application is then initialized, and the actual database is sent. The amount of data may be less than that allocated. The PC ID is written to EEPROM, and the session is then completed.

0x03 <DEVICE INFORMATION REQUEST> <CHKSUM>
0x44 <ACK> <DEVICE INFORMATION REQUEST> <64 byte ICB in EEPROM> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE map table ROM address> 0x00 0x18 <CHKSUM>
0x1C <ACK> <READ FROM ABSOLUTE> <24 byte CORE map table > <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE Application Configuration Data (ACD) RAM address> 0x00 0x10 <CHKSUM>
0x14 <ACK> <READ FROM ABSOLUTE> <16 byte CORE ACD> <CHKSUM>

0x03 <MULTIPLE PACKET WRITE STARTED> <CHKSUM>
0x04 <ACK> <MULTIPLE PACKET WRITE STARTED> <CHKSUM>

0x03 <DELETE ALL APLLICATIONS> <CHKSUM>
0x04 <ACK> <DELETE ALL APLLICATIONS> <CHKSUM>

0x08 <APPLICATION INITIALIZATION INTERNAL> 0x80 0x00 0x00 0x01 0x0A <CHKSUM>
0x04 <ACK> <APPLICATION INITIALIZATION INTERNAL> <CHKSUM>

0x46 <WRITE TO ABSOLUTE> <Application database EEPROM address 1> 0x01 <64 byte Data Block 1> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x16 <WRITE TO ABSOLUTE> <Application database EEPROM address 2> 0x01 <16 byte Residual Data Block 2> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x14 <WRITE TO ABSOLUTE> <PC ID EEPROM address> 0x01 <14 byte PC ID> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x03 <COMMUNICATION PROCESS COMPLETE> <CHKSUM>
0x04 <ACK> <COMMUNICATION PROCESS COMPLETE> <CHKSUM>

15.2. Download New Database to an Existing Application

The following is an example of a communication session to download a new database to an existing application, without affecting other applications previously loaded. The PC will first issue a sequence of commands to characterize the device, and setup any system related data. The application database is then sent. The amount of data may be less than that allocated. The PC ID is written to EEPROM, and the session is then completed.

0x03 <DEVICE INFORMATION REQUEST> <CHKSUM>
0x44 <ACK> <DEVICE INFORMATION REQUEST> <64 byte ICB in EEPROM> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE map table ROM address> 0x00 0x18 <CHKSUM>
0x1C <ACK> <READ FROM ABSOLUTE> <24 byte CORE map table > <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE Application Configuration Data (ACD) RAM address> 0x00 0x10 <CHKSUM>
0x14 <ACK> <READ FROM ABSOLUTE> <16 byte CORE ACD> <CHKSUM>

0x03 <MULTIPLE PACKET WRITE STARTED> <CHKSUM>
0x04 <ACK> <MULTIPLE PACKET WRITE STARTED> <CHKSUM>

0x46 <WRITE TO ABSOLUTE> <Application database EEPROM address 1> 0x01 <64 byte Data Block 1> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x16 <WRITE TO ABSOLUTE> <Application database EEPROM address 2> 0x01 <16 byte Residual Data Block 2> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x14 <WRITE TO ABSOLUTE> <PC ID EEPROM address> 0x01 <14 byte PC ID> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x03 <COMMUNICATION PROCESS COMPLETE> <CHKSUM>
0x04 <ACK> <COMMUNICATION PROCESS COMPLETE> <CHKSUM>

15.3. Download Melody Database to EEPROM

The following is an example of a communication session to download a melody database to EEPROM. For simplicity sake all ROM based application are deleted as part of the session, except for TOD. The PC will first issue a sequence of commands to characterize the device, and setup any system related data. The audio is initialized to the defaults prior to the database download, and then the actual database is sent. The AUDIO EEPROM data is first sent, and then the AUDIO melody address table is written to point to the actual melody. The PC ID is written to EEPROM, and the session is then completed.

0x03 <DEVICE INFORMATION REQUEST> <CHKSUM>
0x44 <ACK> <DEVICE INFORMATION REQUEST> <64 byte ICB in EEPROM> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE map table ROM address> 0x00 0x18 <CHKSUM>
0x1C <ACK> <READ FROM ABSOLUTE> <24 byte CORE map table > <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE Application Configuration Data (ACD) RAM address> 0x00 0x10 <CHKSUM>
0x14 <ACK> <READ FROM ABSOLUTE> <16 byte CORE ACD> <CHKSUM>

0x05 <CALL AN ABSOLUTE ADDRESS> <Audio initialization function call> <CHKSUM>
0x04 <ACK> <CALL AN ABSOLUTE ADDRESS> <CHKSUM>

0x46 <WRITE TO ABSOLUTE> <Sound Database EEPROM address> 0x01 <64 byte Data Block 1> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x16 <WRITE TO ABSOLUTE> <Sound Database EEPROM address> 0x01 <16 byte Residual Data Block 2> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x08 <WRITE TO ABSOLUTE> <AUDIO switch beep melody RAM address> 0x00 <Melody address data> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x08 <WRITE TO ABSOLUTE> <AUDIO hour chime melody RAM address> 0x00 <Melody address data> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x08 <WRITE TO ABSOLUTE> <AUDIO alarm melody RAM address> 0x00 <Melody address data> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x08 <WRITE TO ABSOLUTE> <AUDIO timer melody RAM address> 0x00 <Melody address data> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x08 <WRITE TO ABSOLUTE> <AUDIO interval timer melody RAM address> 0x00 <Melody address data> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x08 <WRITE TO ABSOLUTE> <AUDIO half timer melody RAM address> 0x00 <Melody address data> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x08 <WRITE TO ABSOLUTE> <AUDIO communication error melody RAM address> 0x00 <Melody address data> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x08 <WRITE TO ABSOLUTE> <AUDIO custom melody RAM address> 0x00 <Melody address data> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x03 <MULTIPLE PACKET WRITE STARTED> <CHKSUM>
0x04 <ACK> <MULTIPLE PACKET WRITE STARTED> <CHKSUM>

0x03 <DELETE ALL APLLICATIONS> <CHKSUM>
0x04 <ACK> <DELETE ALL APLLICATIONS> <CHKSUM>

0x14 <WRITE TO ABSOLUTE> <PC ID EEPROM address> 0x01 <14 byte PC ID> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x03 <COMMUNICATION PROCESS COMPLETE> <CHKSUM>
0x04 <ACK> <COMMUNICATION PROCESS COMPLETE> <CHKSUM>

15.4. Upload Database from an Existing Application

The following is an example of a communication session to upload a database from an existing application. The PC will first issue a sequence of commands to characterize both the device, and the application database being read. The application database is then read from EEPROM. The amount of data being read will be determined during the application database characterization phase. The PC ID is written to EEPROM, and the session is then completed.

0x03 <DEVICE INFORMATION REQUEST> <CHKSUM>
0x44 <ACK> <DEVICE INFORMATION REQUEST> <64 byte ICB in EEPROM> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE map table ROM address> 0x00 0x18 <CHKSUM>
0x1C <ACK> <READ FROM ABSOLUTE> <24 byte CORE map table > <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE Application Configuration Data (ACD) RAM address> 0x00 0x10 <CHKSUM>
0x14 <ACK> <READ FROM ABSOLUTE> <16 byte CORE ACD> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE Application Control Block (ACB) RAM address 1> 0x00 0x40 <CHKSUM>
0x44<ACK> <READ FROM ABSOLUTE> <64 byte CORE ACB group 1> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE Application Control Block (ACB) RAM address 2> 0x00 0x40 <CHKSUM>
0x44<ACK> <READ FROM ABSOLUTE> <64 byte CORE ACB group 2> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE Application Control Block (ACB) RAM address 3> 0x00 0x40 <CHKSUM>
0x44<ACK> <READ FROM ABSOLUTE> <64 byte CORE ACB group 3> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <CORE Application Control Block (ACB) RAM address 4> 0x00 0x04 <CHKSUM>
0x08<ACK> <READ FROM ABSOLUTE> <4 byte CORE ACB group 4> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <Application database size EEPROM address> 0x01 0x02 <CHKSUM>
0x06<ACK> <READ FROM ABSOLUTE> <2 byte Application database size> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <Application database EEPROM address 1> 0x01 0x40 <CHKSUM>
0x44<ACK> <READ FROM ABSOLUTE> <64 byte Application database group 1> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <Application database EEPROM address 2> 0x01 0x40 <CHKSUM>
0x44<ACK> <READ FROM ABSOLUTE> <64 byte Application database group 2> <CHKSUM>
…
0x07 <READ FROM ABSOLUTE> <Application database EEPROM address N-1> 0x01 0x40 <CHKSUM>
0x44<ACK> <READ FROM ABSOLUTE> <64 byte Application database group N-1> <CHKSUM>

0x07 <READ FROM ABSOLUTE> <Application database EEPROM address N> 0x01 0x3E <CHKSUM>
0x42<ACK> <READ FROM ABSOLUTE> <62 byte Application database group N – residual bytes> <CHKSUM>

0x14 <WRITE TO ABSOLUTE> <PC ID EEPROM address> 0x01 <14 byte PC ID> <CHKSUM>
0x04 <ACK> <WRITE TO ABSOLUTE> <CHKSUM>

0x03 <COMMUNICATION PROCESS COMPLETE> <CHKSUM>
0x04 <ACK> <COMMUNICATION PROCESS COMPLETE> <CHKSUM>

16. MESSAGE DETAILS

Each of the messages sent by the PC will be listed by class, and the details of the message will be described in the sections that follow. The protocol was designed to require that a response is sent back to the PC by the device. The response allows for retransmission of errored packets. The response format will be indicated in the message details.
The command value for each protocol message is defined in the Message Summary table. Refer to this table for the specific number.

The ACK/NACK characters used in the message response, and are derived from the standard ASCII characters, ACK – 0x0D and NACK – 0xFF. The NACK is sent along with an error code to indicate what condition caused the packet to fail. This could be that a portion of the protocol message contents was not correct or that the protocol message arrived with a transmission receive error.

16.1. Transceiver Notification Messages

The messages generated in this section are two categories external and internal. The internal messages are generated for use within the main MCU. The external messages are responses to commands, and are sent from the main MCU to the PC.

The external messages are broken down into two types of messages, ACK type and NACK type. The ACK type provides an acknowledgement for a particular command with no response data, or an acknowledgement with response data. The NACK type is used to report error conditions back to the PC.

The internally generated messages do not need to have a checksum appended to the message. The length field takes this into account. These messages are for inter module communication purposes on the main MCU. For example, after an entire packet is received a checksum error is detected. The transceiver notification message indicating “Transmission Receive Error” with the “Incorrect CHKSUM” error type is generated. This message is sent from the HW/CORE modules to the COMM module.

The table below shows the various types of errors that have been identified for the protocol. In addition, the table shows where the error is detected, and if it is passed to the PC.

The following Error Types are detected by the CORE before passing the packet to the communications:		
Error Type	**Description**	**Passed To PC**
0x00	**Incorrect CHECKSUM -** This error indicates that checksum calculation does not match the checksum in the packet.	Yes
0x01	**Invalid Message Length -** Indicates that the message length word was received but its value is smaller than expected, or larger than the TUCP buffer size. There is a minimum length of three bytes for a packet.	Yes
The following Error Types are detected by COMM mode after a valid packet has been received and is being processed:		
Error Type	**Description**	**Passed To PC**
0x02	**Device Information Request Not Received-** This error indicates that the protocol revision number protocol message was expected but not received.	Yes
0x03	**PC and Device Mismatch -** The proposed architecture should not produce this error condition. However, processing is added to detect this error condition.	Yes
0x04	**Timeout Expired -** This error indicates that a character has not been detected before the timeout period expired.	Yes

Transmission Receive Error

Description: This is an internal message that is used to notify COMM mode that a receive error occurred during the download process. The protocol provides for retransmission of corrupted commands. The current data transmission is assumed to be in error, and any additional bytes present in the incoming data stream will need to be purged. The lists of error types detected by the CORE are included in a table shown above. The message is internally generated in the main MCU, and will be formatted as follows:
<0x03> <Transmission Receive Error> <Error Type>
This is an example buffer after a checksum error is detected during a receive operation:
<0x03><CMD NUM><0x00>
Response: This message requires a NACK response to be sent to the PC, which is sent after a timeout expires. Prior to the timeout, the incoming stream is purged. See the next message for the details of the NACK response.

Transmission Receive Status

Description: This is a message sent to the PC in response to the message just received. The message sent to the PC contains either an ACK/NACK. There are two types of ACK messages. The simple ACK message is sent along with the command that is being acknowledged, but does not contain response data. When a command requires response data, in addition to the ACK, the message contains the command that is being acknowledged and response data. The NACK message is transmitted for receive errors, or for messages that contain unexpected data. The error type from the table above is included along with the NACK. The various types of messages will be formatted as follows:
Simple ACK
<0x04> <ACK> <CMD NUM> <CHECKSUM>
ACK with response data
<Packet Length Varies> <ACK> <CMD NUM> <Response Data> <CHECKSUM>
NACK
The command number is not sent for NACK responses since it may be corrupted, and does not provide any useful data. The PC is aware of the command it just sent.
<0x04> <NACK> <Error Type> <CHECKSUM>

16.2. Communication Session Protocol Messages

Device Information Request

Description: Protocol message to request data contained in the Information and Configuration Block within EEPROM. This message is required before any other messages are sent by the PC. This message is used by the PC to determine if it has ever communicated with the device. The message will be formatted as follows:
<Packet Length> <Command> <CHECKSUM>
Where: Packet Length = 3
This is an example buffer to request device information:
<0x03><CMD NUM><CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "ACK with response data". The device sends data from the Information and Configuration Block contained in EEPROM. The ACK response is sent based on an error free transmission. See the "Transmission Receive Status" message for the NACK requirements. The number of bytes within the Information and Configuration Block is specified to be 64. If additional bytes are required from EEPROM they will need to be read with an absolute read command. The message will be formatted as follows:
<0x44> <ACK> <CMD NUM> <Information and Configuration Block Data Bytes> <CHECKSUM>

Communication Process Complete

Description: Protocol message indicating the communication process is complete. The message will be formatted as follows:
<Packet Length> <Command> <CHECKSUM>
Where: Packet Length = 3
This is an example buffer after the transmission is completed:
<0x03><CMD NUM><CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the response only after any background initialization is finished based on the new applications or data stored in the device. The ACK response is sent based on an error free transmission and the background initialization completed. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

16.3. System Session Protocol Messages

Delete All Applications

Description: Protocol message to delete all applications except TOD and COMM, and ready the heap/EEPROM for a full download. The message will be formatted as follows:
<Packet Length> <Command> <CHECKSUM>
Where: Packet Length = 3
This is an example buffer after the delete all applications command is received:
<0x03><CMD NUM><CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the ACK response after performing the operation specified, and there were no errors detected in the receive message. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

Beep

Description: Protocol message to make the device beep. The message will be formatted as follows:
<Packet Length> <Command> <Beep Status> <CHECKSUM>
Where: Packet Length = 4
The following are the different Beep Status:
00 Automatic – makes the device beep on every packet afterwards
01 Controlled – makes the device beep only after the beep packet
This is an example buffer to make the device beep after every packet.
<0x04><CMD NUM><0x00><CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the ACK response after performing the operation specified, and there were no errors detected in the receive message. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

Idle

Description: Protocol message to do nothing. The message will be formatted as follows:
<Packet Length> <Command> <CHECKSUM>
Where: Packet Length = 3
This is an example buffer to make the device do nothing.
<0x03><CMD NUM><CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the ACK response after performing the operation specified, and there were no errors detected in the receive message. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

Call an Absolute Address

Description: Calls a routine which starts at a specified absolute address in RAM or ROM. The message will be formatted as follows:
<Packet Length> <Command> <Absolute Address Lo> <Absolute Address Hi> <CHECKSUM>
Where: Packet Length = 5

This is an example buffer to call a routine located at address 067F.
<0x05><CMD NUM><0x7F><0x06><CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the response only after system returns from the call. The ACK response is sent based on an error free transmission and the call completed. Functions that may be called include CORE or other embedded functions, and specialized user written functions. In either case the functions called must insure that the system watchdog timer does not expire. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

Write to TOD Time Zone Data

Description: Changes the setting of the TOD time zones data. The message will be formatted as follows:
<Packet Length> <Command> <Data> <CHECKSUM>
Where: Packet Length = 3 + <Data> size.
See the Application Database Structures section for the size and format of the TOD data.
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the ACK response after performing the operation specified, and there were no errors detected in the receive message. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

16.4. Application Initialization Protocol Messages

Application Initialization Internal

Description: Initializes the next instance of a ROM based application during a full download. The message will be formatted as follows:
<Packet Length> <Command> <Data> <CHECKSUM>
Where: Packet Length = 8 or 35.

The packet length will increase from 8 to 35 if the data for the application contains a mode banner string. The presence of this mode banner string is indicated by a bit in the <Application Configuration Data> field defined below. The format for this string is as defined in the Application Design Guide for the device.

The following bytes comprise the <Data> area:
<Database Data Heap/EEPROM Size Requirements Lo> <Database Data Heap/EEPROM Size Requirements Hi> - database heap size or EEPROM size requirements (2 bytes)
<Application Configuration Data> - bits that indicate an application's configuration. NOTE: Only bits 6 and 7 are used here. The state of these bits will be copied into the corresponding ACD bits in the device. Refer to the WristApp Design Guide for more details.
Bit 7 – set if there is a user specified mode name. The mode banner string is the last field defined for the data within this packet.
Bit 6 – set if the mode requires a password
Bit 5 – not used by PC, set to 0
Bit 4 – not used by PC, set to 0
Bit 3 – not used by PC, set to 0
Bit 2 – not used by PC, set to 0
Bit 1 – not used by PC, set to 0
Bit 0 – not used by PC, set to 0
<Application Instance> - application instance (1 byte)
<POR Application Offset> - offset into the POR address table for the application (1 byte)
<Mode Banner String> - application mode banner string (27 bytes)

All the <Data> bytes are provided by the application, except the Application Instance which is provided by the PC. The PC software has to use the data provided to it by the application in the same order.
The PC has to insure that the Application Instance is unique for each instance of a particular application type. The PC will application SW will determine how to form the instance number for each new application.
This is an example buffer when spawning an application without a mode banner string defined within the data.
<0x08><CMD NUM><Data><CHECKSUM>

Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the ACK response after performing the operation specified, and there were no errors detected in the receive message. If the request can't be granted, due to an error other than a transmission error, then the "PC and Device Mismatch" error is sent with the NACK. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

Application Initialization External

Description: Initializes the first or next instance of an EEPROM based application during a full download. The message will be formatted as follows:

<Packet Length> <Command> <Data> <CHECKSUM>
Where: Packet Length = 31 or 58.
The packet length will increase from 31 to 58 if the data for the application contains a mode banner string. The presence of this mode banner string is indicated by a bit in the <Application Configuration Data> field defined below. The format for this string is as defined in the Application Design Guide for the device.
The following bytes comprise the <Data> area:
<ACB Offset Mask> - bits that indicate whether the specified address is an absolute or an offset address. These bits are set to indicate offset address.
Bit 0 – Application system data
Bit 1 – Application database data
Bit 2 – Application state manager
Bit 3 – Application background handler
Bit 4 – Application mode name
<TOD Resource Requirements> - number of tod resources required. Total available across all applications is 4.
<Backup Timer Resource Requirements> - number of backup timer resources required. Total available across all applications is 2.
<Time Zone Check Resource Requirements> - number of time zone check resources required. Total available across all applications is 5.
<Timer Resource Requirements> - number of timer resources required. Total available across all applications is 3.
<Stopwatch Resource Requirements> - number of stopwatch resources required. Total available across all applications is 2.
<Synchro Timer Resource Requirements> - number of synchro timer resources required. Total available across all applications is 1.
<Flag Ownership 1> <Flag Ownership 2> - lcd flags requirements (2 bytes)
<Code Heap/EEPROM Size Requirements Lo> <Code Heap/EEPROM Size Requirements Hi> - code heap size or EEPROM size requirements (2 bytes)
<System Data Heap Size Requirements Lo> <System Data Heap Size Requirements Hi> - system data heap size requirements (2 bytes)
<Database Data Heap/EEPROM Size Requirements Lo> <Database Data Heap/EEPROM Size Requirements Hi> - database heap size or EEPROM size requirements (2 bytes)
<Application Configuration Data> - bits that indicate an application's configuration
Bit 7 – set if there is a user specified mode name. The mode banner string is the last field defined for the data within this packet.
Bit 6 – set if the mode requires a password
Bit 5 – set if database is invalid
Bit 4 – set if database is modified
Bit 3 – set if code is invalid
Bit 2 – set if database is in external memory
Bit 1 – set if code is in external memory
Bit 0 – set if application index is reserved
<Application Type> <Application Instance> - unique application ID. (2 bytes)
<Application System Data Address Lo> <Application System Data Address Hi> - absolute or offset application system data address (2 bytes)
<Database Address Lo> <Database Address Hi> - absolute or offset database data address (2 bytes)
<State Manager Address Lo> <State Manager Address Hi> - absolute or offset state manager address (2 bytes)
<Background Handler Address Lo> <Background Handler Address Hi> - absolute or offset background handler address (2 bytes)
<Mode Name Address Lo> <Mode Name Address Hi> - absolute or offset mode name address (2 bytes)
<Mode Banner String> - Application mode banner string (27 bytes)

All the <Data> bytes are provided by the application, except the Application Instance which is provided by the PC. The PC software has to use the data provided to it by the application in the same order.
The PC has to insure that the Application Instance is unique for each instance of a particular application type. The PC will application SW will determine how to form the instance number for each new application.
This is an example buffer when spawning an application without a mode banner string defined within the data.
<0x1F><CMD NUM>…<Data>…<CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the ACK response after performing the operation specified, and there were no errors detected in the receive message. If the request can't be granted, due to an error other than a transmission error, then the "PC and Device Mismatch" error is sent with the NACK. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

16.5. Single and Multiple Packet Transfer Messages

Multiple Packet Write Started

Description: Protocol message to indicate that multiple packet writes have been started. This is used when the entire block of data being written will need to be sent before being used. This allows for preventing this data from being used in the event of a communication disruption. This can be used for application or database downloads to RAM/EEPROM. The PC is responsible for setting the code/database invalid flags at the start of the download to prevent the memory from being used until the entire block is transmitted. The message will be formatted as follows:
<Packet Length> <Command> <CHECKSUM>
Where: Packet Length = 3
This is an example buffer to start a multiple packet database download:
<0x03><CMD NUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the ACK response after performing the operation specified, and there were no errors detected in the receive message. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

Write to Absolute Address

Description: Writes the data in the packet into consecutive memory addresses starting from the absolute memory address specified in the packet. The memory being written to is either internal or external. Internal memory is restricted to RAM due to obvious limitations. The packet length depends on the number of data bytes that will be written into memory. The message will be formatted as follows:
<Packet Length> <Command> <Absolute Address Lo> <Absolute Address Hi> <Memory Type> <Data> <CHECKSUM>
Where: Packet Length = 7 - 70
The following are the different Memory Types:
00 Internal – internal RAM memory
01 External – external EEPROM memory
This is an example buffer to write these data, 08, 45, 7A into consecutive RAM memory addresses starting at address F36B.
<0x08><CMD NUM><0x6B><0xF3><0x00><0x08><0x45><0x7A><CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "Simple ACK". The device sends the ACK response after performing the operation specified, and there were no errors detected in the receive message. See the "Transmission Receive Status" message for the NACK requirements. The message will be formatted as follows:
<0x04> <ACK> <CMD NUM> <CHECKSUM>

Read from Absolute Address

Description: Protocol message to read data from consecutive memory starting at the absolute address specified in the packet. The memory being read from is either internal or external. Internal memory is not restricted to RAM. The packet length depends on the number of data bytes that will be read from memory. The message will be formatted as follows:
<Packet Length> <Command> <Absolute Address Lo> <Absolute Address Hi> <Memory Type> <Number of Bytes> <CHECKSUM>
Where: Packet Length = 7
The number of bytes is limited by page size of external memory, and is currently 64.
The following are the different Memory Types:

00 Internal – internal memory
01 External – external memory
This is an example buffer to read 64 bytes from consecutive RAM memory addresses starting at address F36B.
<0x07><CMD NUM><0x6B><0xF3><0x00><0x40><CHECKSUM>
Response: The device is required to provide a response to the PC. The response follows the "Transmission Receive Status" format with a message type "ACK with response data". The device sends data read from the specified memory address in RAM or EEPROM. The ACK response is sent based on an error free transmission. See the "Transmission Receive Status" message for the NACK requirements. The packet length depends on the number of bytes requested. The maximum number of bytes sent is limited to 64. If more bytes are required, then additional read requests will need to be issued. The message will be formatted as follows:
<Packet Length Varies> <ACK> <CMD NUM> <Data> <CHECKSUM>

17. APPLICATIONS

The Timex Data Link USB contains the following applications in ROM which have database structures that can be accessed via the USB port using the COMM Protocol.

- Time/Date
- Chronograph
- Countdown Timer
- Interval Timer
- Alarm
- Appointment
- Contacts
- Notes
- Occasion
- Schedule

The following sections describe the database structures of these ROM-based applications.

18. APPLICATION DATABASE STRUCTURES

18.1. Time/Date Application

The following section describes the database structure for the Time and Date applications.

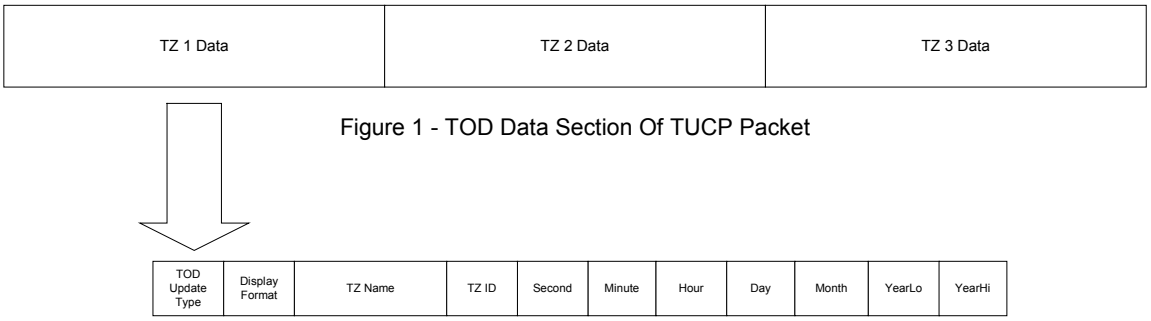


Figure 1 - TOD Data Section Of TUCP Packet

Figure 2 - TOD Single TZ Data Section

Figure 1 – TOD Data Section Of TUCP Packet

- Holds the new data and setting of the 3 time zones.

Figure 2 – TOD Single TZ Data Section

The first data section should refer to time zone 1, followed by TZ 2 then TZ 3.

The Day Of Week and Week Number will automatically be computed when the date is written to the resource.

- Update Type

- Bit 0 - Make this TZ as Primary/Secondary TZ

  Only 1 TZ should have its bit being set. If there are more than 1 TZ that has this bit set, the last TZ data section that has its bit set will be the primary TZ. If none of the TZ has its bit set, then the primary TZ will be the same as prior to PC download.

Make This TZ As..	Value
Primary	TRUE
Secondary	FALSE

- Bit 1 – Update TZ ID

Description	Value
Update TZ ID	TRUE
Don't update TZ ID	FALSE

- Bit 2 – Update display format flags

Description	Value
Update Display Format Flags	TRUE
Don't update display format flags	FALSE

- Bit 3 – Update TZ name

Description	Value
Update TZ name	TRUE
Don't update TZ name	FALSE

- Bit 4 – Update HMS data

Description	Value
Update HMS data	TRUE
Don't update HMS data	FALSE

- Bit 5 – Update MDY data

Description	Value
Update MDY data	TRUE
Don't update MDY data	FALSE

- Bit 6 – Don't care
- Bit 7 – Don't care

- Display Format Flags

  - Bit 0 & 1 – Date Format

Bit 1	Bit 0	Date Format

0	0	DMY
0	1	YMD
1	0	MDY

- Bit 2 – Hour Format

Description	Value
24-Hour Format	TRUE
12-Hour Format	FALSE

- Bit 3 – Display DOW or Week Number

Description	Value
Display Week Number	TRUE
Display Day Of Week	FALSE

- Bit 4 – TZ in DST

State/Description	Value
TZ is in DST	TRUE
TZ is not in DST	FALSE

- Bit 5 – US or EURO Format

Description	Value
EURO Format	TRUE
US Format	FALSE

- Bit 6 – TZ observes DST

Description	Value
TZ observes DST	TRUE
TZ doesn't observe DST	FALSE

- Bit 7 – TZ Entered Set State

  This bit must be set if the update MDY bit (bit 5) of update type is set. The means that current TZ has entered set state.

  When this bit is set on TZ2 and TZ3 and manually changes the TZ1 in set state, TZ1 will not be copied anymore to either TZ2 or TZ3.

Description	Value
TZ Entered Set State	TRUE
TZ Hasn't Entered Set State	FALSE

- TZ Name

- 3-character TZ name. If the TZ name is less than 3 characters, pad the remaining with space characters.
- Displays the name using 5-row dot matrix character.
- Valid characters are from the first character defined in the LCD up to the last setting character. (Editable character set)

- TZ ID
  - Used for world time wristapp.
- Second
  - BCD data from 00 – 59.
- Minute
  - BCD data from 00 – 59.
- Hour
  - BCD data from 00 – 23.
- Day
  - BCD data from 1 to the maximum number of days in the specified month.
  - The PC should send a correct data for last day of February for leap years and non-leap years.
- Month
  - BCD data from 1 – 12.
- YearLo
  - BCD data from 00 – 99.
- YearHi
  - Fixed BCD data of 20 since year boundary is from 2000 – 2009.

18.2. Chronograph Application

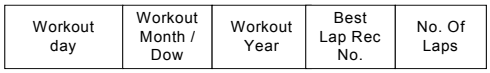
The following section describes the database structure for the Chronograph application.

Application Database Data

The structure of the Chrono application database data is tailored to the Data Link USB Database Design description Random-Fix database structure. This database structure was carefully chosen based from the collective analysis of a number of possible database structure implementation of the chrono mode. Please refer to the Data Link USB Chrono mode database design analysis document for further historical details.

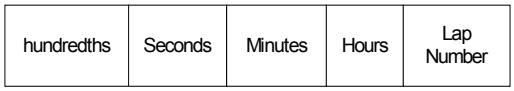
There are two types of records for the chrono database, the workout record and the split record. The following describes these record structures

Workout Record Structure



No. Of Bytes	Field Name	Description
1	Workout Day	Storage for the Days field
1	Workout Month and DOW	This is a stuffed information for the month and the DOW. This is done to reduce the number of database bytes that the chrono would have for the workout record. The lower nibble of this byte holds the information for the workout month and the upper nibble holds the workout's DOW
1	Workout Year	Storage for the Years field (low byte only)
1	Workout Best Lap Rec. No.	Storage for the Workouts Best Lap Record Number.
1	Workout No of Laps	Storage for the Workout's number of laps

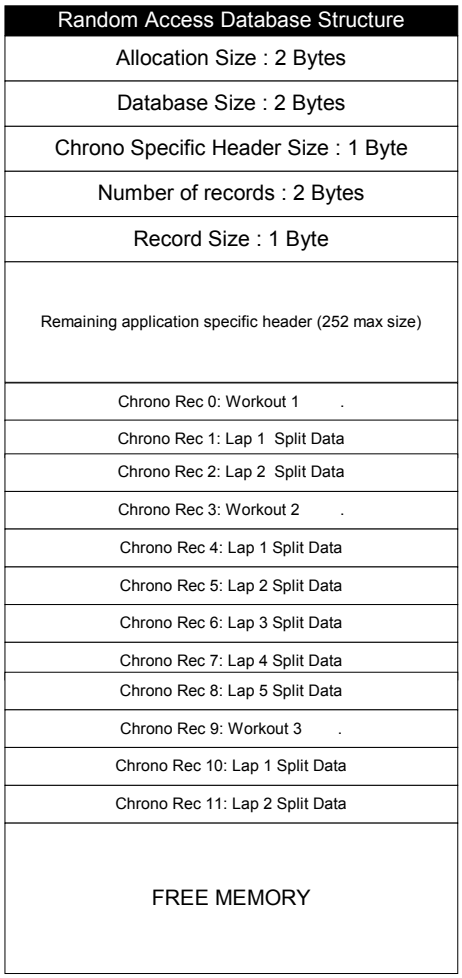
Split Record Structure



No. Of Bytes	Field Name	Description
1	Hundredths	Split's Hundredths information
1	Seconds	Split's Seconds information
1	Minutes	Split's Minutes information
1	Hours	Split's Hours information
1	Lap Number	Split's Lap Number information

Chrono Database Structure

Below is a layout of how the Chrono database structure looks like. This is a Random Access with Fix number of field size database structures. Database API to access each and every location or record are available.



The chrono database has a fixed number of byte fields in each record, where it consists of 5 bytes. A record may store workouts or split information, with both structures of the same number of fields. A set of workout may consist of the workout date stamp information, and series of laps. In a workout, information is organized and arranged in the manner that the Workout record resides in the top most record, followed by the lap's split records, depending on the number of laps taken in a workout.

Chrono Database Header Details

Offset	Offset Name	Description
0	CHRALLOCATIONSIZEOFFSET	Stores the allocation size for the entire chrono database. 2-bytes
2	CHRDATABASESIZEOFFSET	Stores the size of the entire database 2-bytes
4	CHRSPECIFICHEADERSIZEOFFSET	Stores the size of the chrono specific header information. The value should be 12. 1-byte
5	CHRNUMBEROFRECORDSOFFSET	Stores the no of records for the chrono database. This determines the amount of

		storage workout/lap capacity can be achieved. MaxNumLaps+1. 2 – bytes
7	CHRRECORDSIZEOFFSET	Stores the record size. The value should be 5. 1 – byte
8	CHRDISPLAYFORMATOFFSET	Stores the display format information. 1-byte Lap (top) / Split (bottom) = 0 Split (bottom) / Lap (top) = 1 Time (top) / Split (bottom) = 2 Time (top) / Lap (bottom) = 3
9	CHRSYSTEMFLAGOFFSET	Stores the system flag information. Bit 2, which is bCHRMemoryFull, should be set when the number of downloaded free records is less than two. 1-byte
10	CHRLAPCOUNTOFFSET	Stores the current lap count information. Initially this should be 1. Candidate for removal from ADD. 1-byte
11	CHRLAPSFREECOUNTOFFSET	Stores the no of laps free information. Laps free count should be the total number of records minus the no of used records, minus one (to account for the workout storage record in the future). Candidate for removal, see offset 14. 1-byte
12	CHRMAXLAPCOUNTCONFIGOFFSET	Store the maximum no of laps that can be stored. Maximum Lap count should be the total number of records minus one. 1-byte
13	CHRFREEMEMRECNOOFFSET	Stores the free memory record no information. The free memory record number is the total number of record minus the number of used records. 1-byte
14	CHRLASTWORKOUTRECNOOFFSET	Stores the last workout record number. This should be the record number of the last stored workout. 1-byte
15	CHRUNSTOREWORKOUTRECNOOFFSET	Stores the unstored workout record information. This record number should be the same as the free memory record number when database download occurs. 1-byte

Notice that header information that holds the record numbers allocates only one byte since the chrono never has more than 250 records.

18.3. Countdown Timer Application

The following section describes the database structure for the Countdown Timer application.

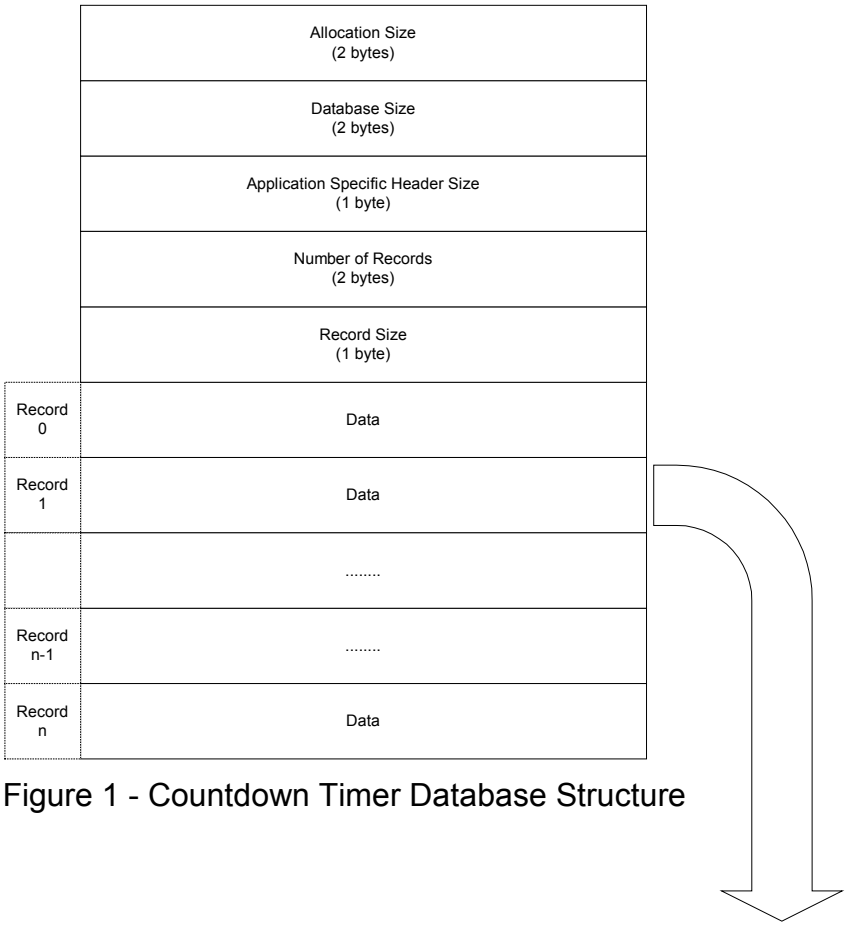


Figure 1 - Countdown Timer Database Structure

Figure 2 - Countdown Timer Record Data

Figure 1 – Countdown Timer Database Structure

- The DB and app specific header is a standard header for a fixed-sized random database structure. (From allocation size down to record size)

Figure 2 – Countdown Timer Record Data

- Flags
  - Bit 0 & 1 – Action at end.

Bit 1	Bit 0	Action At End
0	0	Stop At End
0	1	Repeat At End
1	0	Chrono At End

  - Bit 2 – Enable/Disable halfway reminder.

State	Value
Enable	TRUE
Disable	FALSE

  - Bit 3 – Timer data is 0.

Description	Value
Data is zero	TRUE
Data is non-zero	FALSE

  - Bit 4 – Timer data is less than 15 seconds.

Description	Value
Data is less 15 sec	TRUE
Data is not less 15 sec	FALSE

  - Bit 5 – Timer data is less than 1 minute.

Description	Value
Data is less 1 min	TRUE
Data is not less 1 min	FALSE

  - Bit 6 – Don't care.
  - Bit 7 – Don't care.

- Second
  - BCD data from 00 – 59.
- Minute
  - BCD data from 00 – 59.
- Hour
  - BCD data from 00 – 23.

- Name

  - 9-character entry name. If the entry name is less than 9 characters, pad the remaining with space characters.
  - Displays the name using 5-row dot matrix character.
  - Valid characters are from the first character defined in the LCD up to the last setting character. (Editable character set)

18.4.Interval Timer Application

The following section describes the database structure for the Interval Timer application.

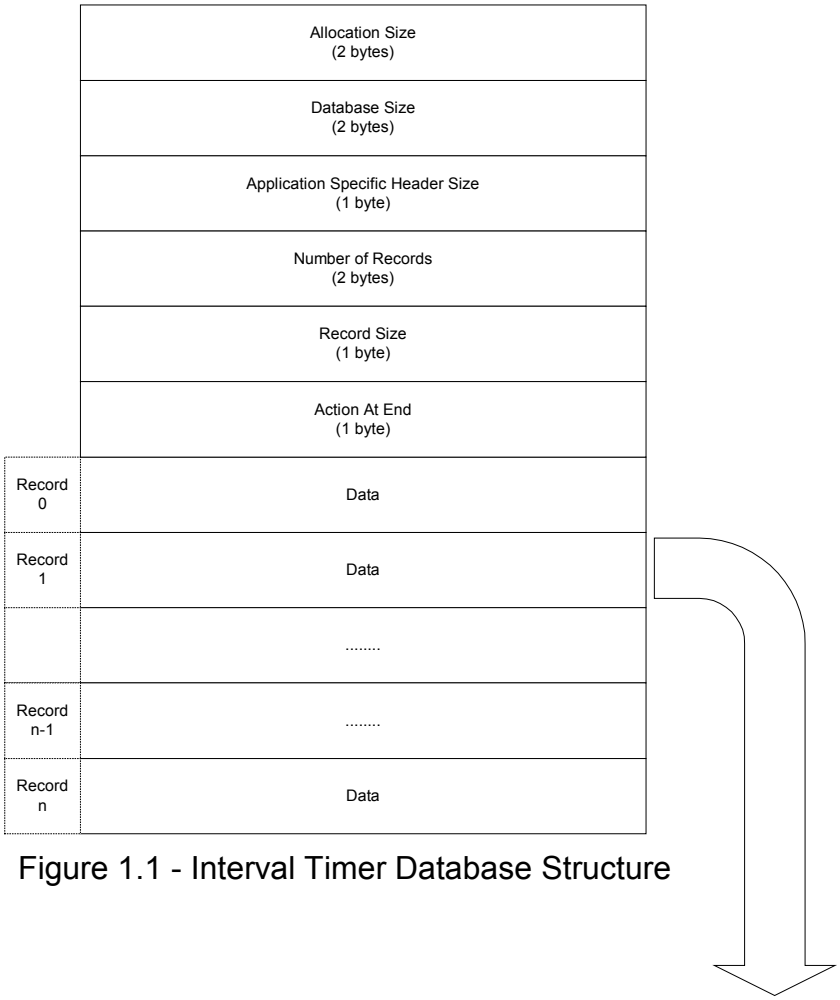


Figure 1.1 - Interval Timer Database Structure

Figure 1.2 - Interval Timer Record Data

Figure 1 – Interval Timer Database Structure

- The DB and app specific header is a standard header for a fixed-sized random database structure. (From allocation size down to record size)

- Action at end.

Bit 1	Bit 0	Action At End
0	0	Stop At End
0	1	Repeat At End
1	0	Chrono At End

Figure 2 – Interval Timer Record Data

- Flags

  - Bit 0 – Don't care.
  - Bit 1 – Don't care.

  - Bit 2 – Enable/Disable halfway reminder.

State	Value
Enable	TRUE
Disable	FALSE

  - Bit 3 – Timer data is 0.

Description	Value
Data is zero	TRUE
Data is non-zero	FALSE

  - Bit 4 – Timer data is less than 15 seconds.

Description	Value
Data is less 15 sec	TRUE
Data is not less 15 sec	FALSE

  - Bit 5 – Timer data is less than 1 minute.

Description	Value
Data is less 1 min	TRUE
Data is not less 1 min	FALSE

  - Bit 6 – Don't care.
  - Bit 7 – Don't care.

- Second

  - BCD data from 00 – 59.

- Minute
    - BCD data from 00 – 59.
- Hour
    - BCD data from 00 – 23.
- Name
    - 9-character entry name. If the entry name is less than 9 characters, pad the remaining with space characters.
    - Displays the name using 5-row dot matrix character.
    - Valid characters are from the first character defined in the LCD up to the last setting character. (Editable character set)

18.5. Alarm and Appointment Applications

The following section describes the database structure for the Alarm and Appointment applications.

8. Appt and Alarm Application Database Structure

This section introduces and discusses the Data structures used in the implementation of the Appointment and Alarm Applications. The Data Record structure is introduced along with how it is utilized in the ADD (Application Database Data) and ASD (Application System Data). Details of the ADD are provided in a section below. The details of the ASD are found in each application's SDD (Software Design Document).

In general, long-term storage (EEPROM) of appointment record data is provided for in the ADD. Short-term storage (RAM) for easy access (display and editing) is provided for in the common RAM area. Displaying and editing a record is performed with record data in the common foreground RAM area while searching algorithms utilize the common background RAM area.

For ROM efficiency both the alarm and appointment applications use identical Record Data structures as well as an identical ADD. The ASD is identical up to the SEARCHRECORDOFFSET Field. The PEEKRECORDPTROFFSET, TZCHECKRESOURCEPEEKRECORDOFFSET and TIMELINE fields are applicable only to the appointment application. Refer to the specific applications SDD for further details.

9. Record Data Structure

The record data structure below will be used in both the ASD and the ADD. A description of each element in the record is provided.

(Data Structure Rev 7)

Num Bytes	Field Name	Description
2	UTLTEMPNEXTRECORDPTROFFSET	Next Record Address in EEPROM.
2	UTLTEMPPREVRECORDPTROFFSET	Previous Record Address in EEPROM
1	UTLTEMPRECORDLENGTHOFMESSAGEOFFSET	The length of the LCDScrollMessage for this record. The total number of bytes for

		the record equals the number of header record header bytes plus the message length. The total number of bytes possible in the structure is variable based on the length of the LCDScrollMessage. For a maximum length Message of 100 character plus 1 sentinel character the total number of bytes possible in the structure will be 117 bytes.
1	UTLTEMPSTATUSOFFSET	A byte of Status information. Used by Watch and PC. Bit 0 is LSB (Right most bit). Bit 0 - 1=Used, 0=Unused. Indicates whether an appointment is Used (on the Used List) or Unused (on the Unused List). Configured by the PC prior to dload and by the watch when record are edited or deleted. Bit 1 - 1=Armed (Enabled), 0=Unarmed (Disabled). If Armed = 1 then the alert has been armed to cause a popup and the Icon is displayed when viewed in the default state. Only Used records (see bit 1) can be armed. Set by PC and Watch. Bit 2 - 1=Modified by the User via watch UI, 0=Unmodified. Set to 0 by the PC prior to dload. Set to 1 by watch when user edits the record. Bit 3 - 1=Deleted by the User via watch UI, 0=Undeleted. If the record was Used to start then deleted via the watch UI then it becomes Unused and the Deleted Bit gets set = 1. If the same record then gets Used again, the Deleted bit remains set = 1. Set to 0 by the PC prior to dload. Set to 1 by watch when user deletes the record. Bit 4 – 1=Appt Record type, 0=Alm record type. Used by the background search routines to distinguish between which application has called the proc in order to provide special processing for each application. Set or cleared by the PC prior to dload. Never edited by the watch. Bit 5 – 1=EndofList “Record” is displayed, Set to 0 by the PC prior to dload. Set = 1 by watch whenever it creates a temporary end-of-list record in RAM. Set = 0 by watch when temporary end-of-list record is overwritten.
1	UTLTEMPFREQOFFSET	Freq - Occurrence Frequency of the Alarms

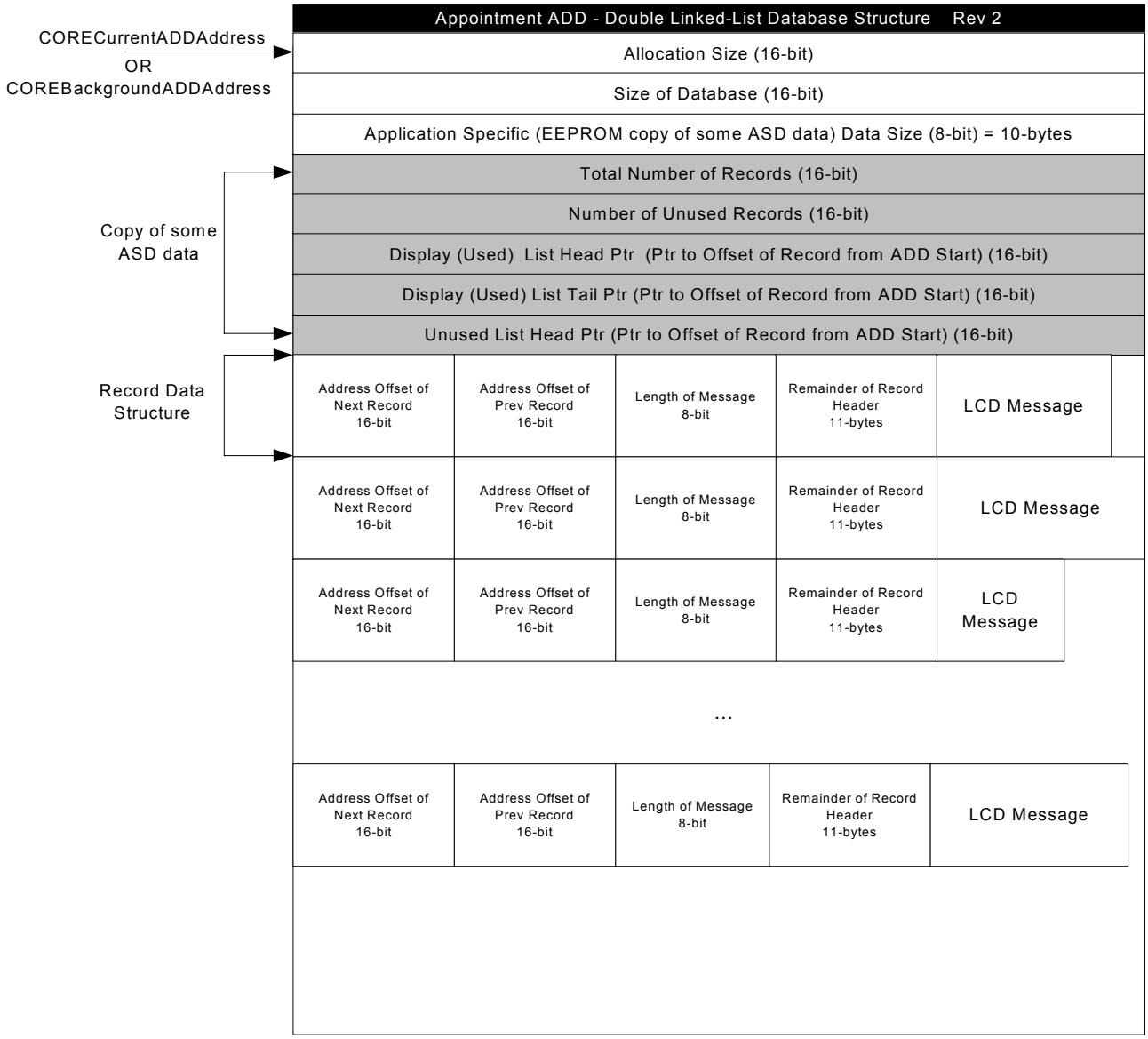
		and Appointments. Displayed in the Upper section of the LCD in the Segment area. For One-Day, Weekly, Monthly, and Yearly Appointments the calculated day of the week for the first occurrence, if armed, is displayed in the Upper section of the LCD in the Small Dot. The frequencies Daily through WklySA are used by the alarm application. The Appointment application uses One_Day through Yearly, but does not use WklySU through WklySA. This is a hex value after upper nibble is masked off. Set by the PC prior to dload and set by the watch when a record is edited. One_Day = 0 Daily = 1 WkDay = 2 WkEnd = 3 WklySU = 4 WklyMO = 5 WklyTU = 6 WklyWE = 7 WklyTH = 8 WklyFR = 9 WklySA = 10 Weekly = 11 Monthly = 12 Yearly = 13
1	UTLTEMPPRENOTIFICATIONOFFSET	The Hex value is an index into a lookup table. The lookup table returns the BCD of minute or hour interval that needs to be subtracted from the Appointment schedule time in order to calculate the actual appointment alert time. Set by the PC prior to dload and set by the watch when a record is edited. Index(Hex) → Return Value (BCD) 0 → 0 Mins 1 → 5 Mins 2 → 10 Mins 3 → 15 Mins 4 → 30 Mins 5 → 60 Mins 6 → 2 Hrs 7 → 3 Hrs 8 → 4 Hrs 9 → 5 Hrs 10 → 6 Hrs 11 → 8 Hrs 12 → 10 Hrs 13 → 12 Hrs 14 → 24 Hrs 15 → 48 Hrs
1	UTLTEMPMINOFFSET	Minute at which the record is set to expire, if armed, Range 0-59. Displayed in the middle section of the LCD in the Lower Dot

		matrix. BCD format. Set by the PC prior to dload and set by the watch when a record is edited.
1	UTLTEMPHROFFSET	Hour at which the record is set to expire, if armed. Range 0-23. Displayed in the middle section of the LCD in the Lower Dot matrix. BCD format. Set by the PC prior to dload and set by the watch when a record is edited.
1	UTLTEMPDATEOFFSET	The Date of the month in which the first Appt \ alm occurs. BCD format. Range 1–31. Set by the PC prior to dload and set by the watch when a record is edited. ALARM record DMY should be set to 01-01-2000 at these times.
1	UTLTEMPMONTHOFFSET	The Month in which the first Appointment occurs. BCD Format. Range 1–12. Set by the PC prior to dload and set by the watch when a record is edited. ALARM record DMY should be set to 01-01-2000 at these times.
1	UTLTEMPYRLOOFFSET	Lo byte of Year. BCD Format. “01” for year ‘2001’. Set by the PC prior to dload and set by the watch when a record is edited. ALARM record DMY should be set to 01-01-2000 at these times.
1	UTLTEMPYRHIOFFSET	Hi byte of Year. BCD Format. “20” for year ‘2001’. Set by the PC prior to dload and set by the watch when a record is edited. ALARM record DMY should be set to 01-01-2000 at these times.
2	UTLTEMPPCRECORDIDOFFSET	Used by the PC for the PC. Never used by the watch. This OFFSET is defined but the watch never reads these two bytes into RAM.
101	UTLTEMPLCDSCROLLMSGOFFSET	A variable length message. It is defined by the user at the PC prior to download to the watch. The Max number of characters is 100 plus 1 byte for a sentinel character. The default character length for records stored in the watch prior to download is the maximum. The message is displayed in the lower section of the LCD on the Lower Dot matrix.

Header of 16 bytes.
Max LCDMessage 101 bytes
Max Record Size 117 bytes.

10. ADD (Application Database Data)

The ADD (Application Database Data) is located in EEPROM. The following diagram illustrates the information included within the ADD. The ADD is also referred to as the database or dbase for short. These terms are used interchangeably throughout this document.



Total Allocation Size (in bytes) of the ADD at Default = 512 = [(485 / 64) + 1] * 64,
where 64 is the EEPROM page byte size.
Total DBase Size (in bytes) of the ADD at Default = 485
Default Appt Record Size = 47
Default Number of Records = 10

The Record Data is used to store an Appointment entry. The Appointment Database stored in EEPROM will consist of a number of Header bytes and one or more Data Structures, also referred to an Appointment Record or record or entry. Details on the record can be found in the Record Data Section of this document.

NOTE: The fields “address of next record”, “address of prev record” and “Length of Message” are included as part of the record data. The diagram below gives the incorrect impression that they are not. See the section on “Record Data Structure” for details.

Notice that when this application is running in foreground the CORE variable CORECurrentADDAddress contains the value of the EEPROM start address for the ADD. When called in background the CORE variable COREBackgroundADDAddress contains the value of the EEPROM start address for the ADD.

Also notice the duplication of some of the ASD specific data. The ADD field “Application Specific Data Size” indicated by the definition ADDAPPLICATIONSPECIFICSIZE indicates the number of ASD specific data bytes that are maintained in EEPROM. This data is maintained in ASD RAM and copied to the ADD EEPROM when the PC requests a data read from the watch. On application refresh after a PC-download this data needs to be read from the ADD into the ASD.

Records in the ADD are connected using a double linked-list Database structure. Refer to the section on the Database list for details of this data structure for this application.

The maximum number of records that could possibly be stored in the ADD is limited only by the size of the EEPROM. Note that as the number of record in the dbase increases the time to search the dbase for the next alert occurrence also increases. See the table that follows for access times given various numbers of records in the dbase.

Number of (13 byte) records in Database	10	20	50	100	250	500	1000
Access time for complete Query of Database (Secs)	0.0773	0.1546	0.3865	0.773	1.9325	3.865	7.73

As a result of this limitation the search for the next appointment to popup is performed by a background task. Refer to the section on “Searching for the Next Occurrence” and “TZ Change Resource” for more details. A side effect of this background search is that a search may be in progress when the user tries to perform a PEEK operation. If this occurs an the search is in progress then a message “SORTING TRY LATER” message will be displayed on the LCD.

The initial default conditions for the database will be 10 record, each unused and each containing an editable 30 character LCDScrollMessage. Downloading more or less records from the PC software will change this configuration.

11. Database Access Type

Since each record in the database needs to be an element of a linked list and is variable in length the database access type would need to be of type Double Linked-List. A database type Random Variable could be used, but the Linked-List is more straightforward with respect to deleting records. Refer to the Database design document for details on these database types.

18.6. Contacts Application

The following section describes the database structure for the Contacts application.

1. Data

The section introduces and discusses the data structures used in the implementation of the Contact Application. The Contact data record structure is introduced along with how it is utilized in the Application Database Data (ADD) and Application System Data (ASD). Details of both the ADD and the ASD for this mode are provided in separate sections.

Application Database Data (ADD)

The ADD for this application, also referred to as the Contact Database is located in EEPROM. The diagram in this section illustrates the information included within the ADD. Details of this diagram can be found in the sub-sections that follow.

The application can handle a database with no contact record or a maximum of 65534 records. Data Link USB database design restricts the total accessible memory size of a database should be less than or equal 65536 bytes or dependent upon actual physical external memory size fitted into the system.

The application does not provide a delete or edit operation. With the simple database requirements, the records in the database are accessed using a random variable size access type.

A record stores only one phone number and a message field. A message can be as long as 100 characters. To maximize external memory usage for contact information with multiple phone numbers, the message field can be removed, replaced with a record pointer field to indicate which record contains the message to display. With this setup, one contact message can have a maximum of 256 phone numbers.

Ordering of the records within the database is as received by the watch during PC download and usually sorted in ascending order sorted off the message field.

2. Contact Database Structure

The contact database follows the database format specified for random access for variable sized records. This type of database access is used due to the simple read operations required by the application UI since there is no delete function; record size varies.

The database structure is shown below:

CONTACT DATABASE
Allocation Size (16-bits)
Database Size (16-bits)
Application Specific Header Size (8-bits) 0x02
Number of Records (16-bits)
Record #0 Offset (16-bits)
Record #1 Offset (16-bits)
. . .
Record #n Offset (16-bits)
Record #0
Record #1
Record #2
. . .
Record #n

Allocation Size
The allocation size specifies the total number of bytes (in 64-byte increments) allocated in the EEPROM for the contact database. This allows the database size to grow within the specified allocation size to support partial download.
Database Size
The database size indicates the actual database usage. This will be used by the PC to determine how many bytes to request from the watch for upload.
Application Specific Header Size
This specifies the number of bytes in the database header section that is used by the application. In the contact database, this header size is always 2. This is the number of records in the database (16-bit value).

Record Offset Pointers
The record offset pointer section is used by the database drivers to access any records in the database.

Records
The record section stores all the contact information accessed by the application.

3. Contact Record Structure

The record structure is shown below:

CONTACT RECORD				
Contact Type (2 bytes)	Packed Phone Number (7 bytes)	Message Record Offset (1 byte)	Message Length (1 byte)	Message (0 to 101 bytes)

Contact Type
This 2-byte field specifies the following:

1. the characters that will be displayed in the upper dot matrix region;
2. left arrow should be displayed in the leftmost position of the line 1 in the main dot matrix region;
3. right arrow should be displayed in the rightmode character position of line 1 in the main dot matrix region.

The type has the following format:

Left Arrow Display (1 bit)	Type Character 1 (7 bits)	Right Arrow Display (1 bit)	Type Character 2 (7 bits)

The left arrow and right arrow bits are setup by the PC during database creation to indicate that the phone number extends to the next or previous record.

The allowed values for Type Character 1 and Type Character 2 is any two characters from the LCD dot matrix character set to be displayed. Possible character combinations are:

Phone Type	Possible Display
HOME	'HM' or 'H'
WORK1	'W1'
WORK2	'W2'
FAX	'FX'
PAGER	'PG' or 'P'
CELL	'C'

NOTE: A new contact application UI will not make use of the Left-Arrow display bit. It will now specify Type Character 1 to store a LEFT-ARROW character with Type Character 2 to be a SPACE to signify previous record. It will still utilize Right-Arrow display bit to indicate continuation on the next record. Only 2 records are allocated for long phone numbers.

Packed Phone Number

This field stores 14 character indexes (4-bits per index) in a 7-byte field structure. The character position indicates where in the display area the indexed character is to be displayed. The display regions are indicated below:

- For Segmented Display (6 characters)
- For Dot-matrix Display (8 characters)

For the first 6 character indexes, it will use the following table to determine the actual character to display:

Index	Character Equate	Character Value
0	SEG_0	0x00
1	SEG_1	0x01
2	SEG_2	0x02
3	SEG_3	0x03
4	SEG_4	0x04
5	SEG_5	0x05
6	SEG_6	0x06
7	SEG_7	0x07
8	SEG_8	0x08
9	SEG_9	0x09
10	SEG_SPACE	0x0A
11	SEG_DASH	0x21
12	SEG_DASH	0x21
13	SEG_PLUS	0x22
14	SEG_OPENPAR	0x0D
15	SEG_CLOSEPAR	0x23

The next 8 character indexes will use the following table to determine the actual character to display:

Index	Character Equate	Character Value
0	DM5_0	0x00
1	DM5_1	0x01
2	DM5_2	0x02
3	DM5_3	0x03
4	DM5_4	0x04
5	DM5_5	0x05
6	DM5_6	0x06
7	DM5_7	0x07
8	DM5_8	0x08
9	DM5_9	0x09
10	DM5_SPACE	0x0A
11	DM5_DASH	0x31
12	DM5_PERIOD	0x32
13	DM5_PLUS	0x2F
14	DM5_ASTERISK	0x2E
15	DM5_NUMBER	0x27

Message Record Offset

The 1-byte field indicates the negative offset from the current record number where the message is to be read from. This allows one message entry to have 256 phone numbers. This maximizes external memory usage for contact information with multiple phone numbers, the message field can be removed, replaced with a record pointer field to indicate which record contains the message to display.

The figure below shows a typical contact info with 4 phone numbers:

Record n+0	Contact Type ‘W1‘	Packed Phone Number (7 bytes)	Message Record Offset 0	Message Length 15	Message “NINO SARMIENTO” + sentinel
Record n+1	Contact Type ‘W2 ‘	Packed Phone Number (7 bytes)	Message Record Offset 1	Message Length 15	
Record n+2	Contact Type ‘C ‘	Packed Phone Number (7 bytes)	Message Record Offset 2	Message Length 15	
Record n+3	Contact Type ‘F ‘	Packed Phone Number (7 bytes)	Message Record Offset 3	Message Length 15	

Message Length

The message length specifies the size in bytes of the message field. It is required for all records (even those without the message field) to eliminate a separate read access to determine the size of the message prior to reading the actual message itself.

Message

The message field can be 0 bytes to 101 bytes long. The last character should be DM_SENTINEL. Valid values for this are specified in the Data Link USB Dot Matrix Character Set. Values range from 0x00 to 0x69.

NOTE: *If this field is empty, make sure that the ‘Message Record Offset’ field is not 0.*

18.7.Notes Application

The following section describes the database structure for the Note application.

1. Note Record Data Structure

The data structure below is that which will be used to store a Note Record in EEPROM. A description of each field in the record is provided.

The record data structure will be used in the ADD and portions will be used in the ASD. Refer to each section for details.

Number of Bytes	Field Name	Description
2	NextRecordAddress	The address of the next record in the database.
2	PreviousRecordAddress	The address of the previous Record in the database.
1	RecordSize	The total number of bytes in the message + 1 (to account for the sentinel character). The total number of bytes possible in the structure is variable based on the length of the LCDScrollMessage. For a maximum length Message the total number of bytes possible in the structure will be 106 bytes.
1	Record Status	Bit0, 0 = record is unused, 1 = record is used. Set to 0 by the watch when record is deleted. Bit1, 0 = record is unmodified, 1 = record is modified by the watch.
2	PC Record ID	Used by the PC. Never used by the watch. The watch doesn’t need to load it into RAM.

101	LCDScrollMessage	A variable length message. The user defines the message content at the PC prior to download to the watch or by editing the message. The Max number of characters is 100 plus 1 byte for a sentinel character. The default LCDScrollMessage length for records stored in the watch prior to download is 30 with blanks for all the characters. The message is displayed in the lower section of the LCD on the Lower Dot matrix.

2. Database Structure (ADD)

The table below shows the structure of the database table in the EEPROM:

NOTE ADD Double-Linked List Access Database Structure						
Allocation Size of EEPROM reserved for Application (32-byte boundaries) : 16-bit						
Database Size (subset of Allocation Size): 16-bit						
Application Specific Header Data Size (8-bit) = 10 bytes						
Total Number of NOTE Records: 16-bit						
Number of Unused NOTE Records: 16-bit						
Display (Used) List Head Ptr (Ptr to Offset of Record from ADD Start) (16-bit)						
Display (Used) List Tail Ptr (Ptr to Offset of Record from ADD Start) (16-bit)						
Unused List Head Ptr (Ptr to Offset of Record from ADD Start) (16-bit)						
Next Record Address 2-byte	Previous Record Address 2-byte	Record Size 1-byte	Record Status 1 byte	PC Record ID 2 bytes	LCDScrollMessage	
Next Record Address 2-byte	Previous Record Address 2-byte	Record Size 1-byte	Record Status 1 byte	PC Record ID 2 bytes	LCDScrollMessage	
Next Record Address 2-byte	Previous Record Address 2-byte	Record Size 1-byte	Record Status 1 byte	PC Record ID 2 bytes	LCDScrollMessage	
.....						
Next Record Address 2-byte	Previous Record Address 2-byte	Record Size 1-byte	Record Status 1 byte	PC Record ID 2 bytes	LCDScrollMessage	

3. Application system data structure (ASD)

The table below shows the structure of the application system data used by the note mode:

NOTE Mode Application System Data (ASD) REV 2
NOTE ASD Flags: 8-bit Bit 0: 1=Scrolling has stopped, 0=Scrolling in Progress. Bit 1: 1=Unused entry is beening displayed, 0=unused entry is not displayed. Bit 2: 1=End-Of-List is being displayed, 0=EOL is not displyed.
Total Number of Records (16-bit)
Number of Unused Records (16-bit)
Display (Used) List Head Ptr (Offset of Record from ADD Start) (16-bit)
Display (Used) List Tail Ptr (Offset of Record from ADD Start) (16-bit)
Unused List Head Ptr (Offset of Record from ADD Start) (16-bit)
Ptr to Currently Viewed Record (Offset of Record from ADD Start) (16-bit)
Currently Viewed Next Record Address Ptr (Ptr to Offset of Record from ADD Start) (16-bit)
Currently ViewedPrevious Record Address Ptr (Offset of Record from ADD Start) (16-bit)
Currently Viewed SizeOfRecord (8-bit)
Current Record Status (8-bit)

Total 19 bytes

18.8. Occasion Application

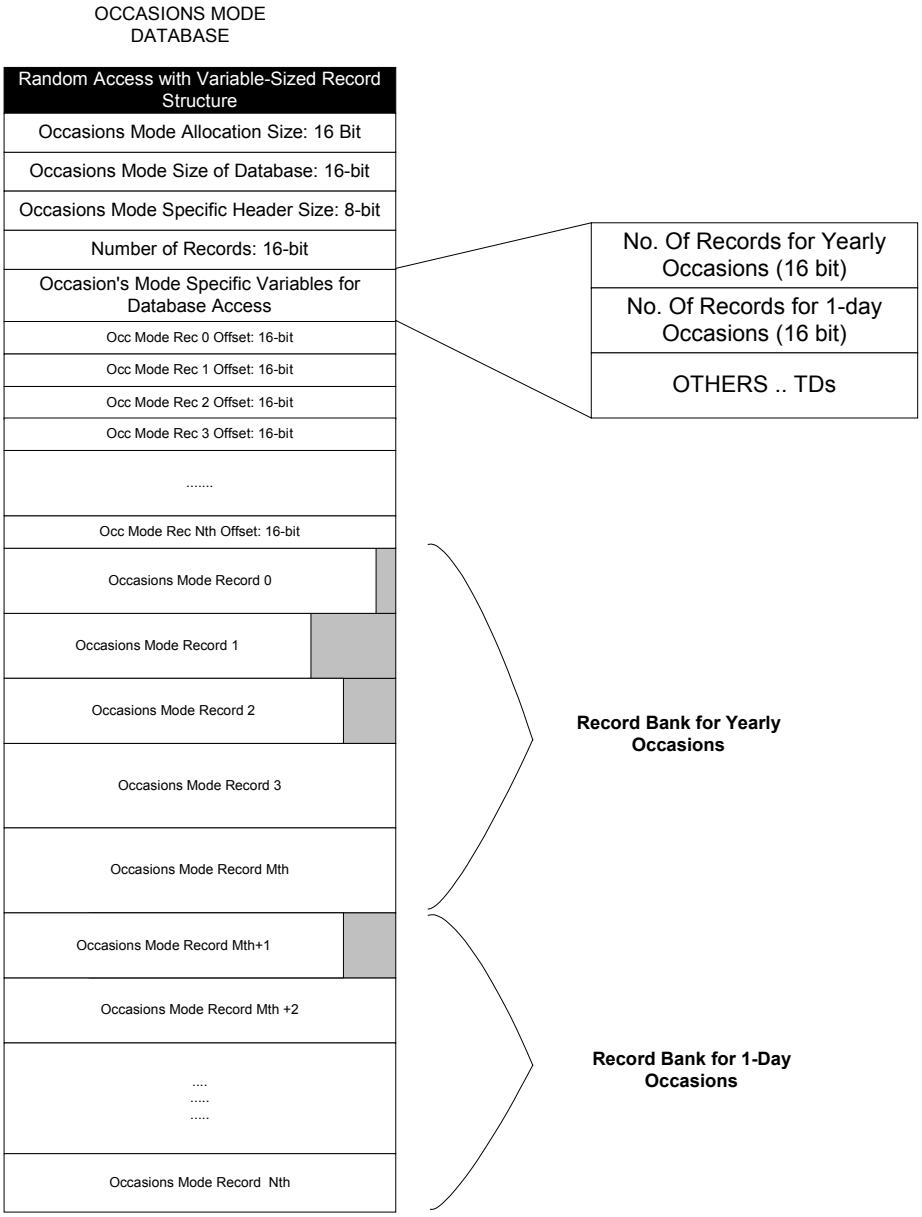
The following section describes the database structure for the Occasion application.

1. Application Database Data

The structure of the Schedule Mode application database data is tailored to the Data Link USB Database Design description Random-Variable database structure. One unique implementation that the was done was to have a nested Random-variable database structure. This is done to serve a database of different schedule groups and a database of entries in each group.

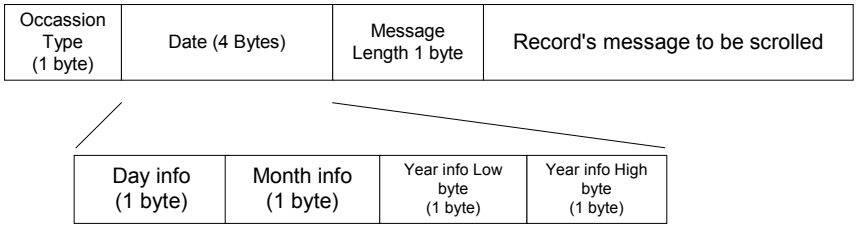
2. The Entire Database structure.. a big picture

The occasions mode uses a random variable size database structure as shown.



The single database may be divided into two banks. The first bank should hold all recurring entries, sorted in an ascending order with respect to the entry's **DAY and MONTH ONLY**. The second bank contains all non-recurring occasion entries, sorted accoriding to their respective date: **DAY, MONTH and YEAR.** Splitting up the entries into two different banks is manageable and easy to manipulate and fast from the standpoint of the search routines, traverse forward and traverse backward routines.

3. Occasions mode Record entry structure:



Above is a layout of what the occasion's mode record entry structure should consist of. Each entry has a fixed sized Occasion Type (1 byte), fixed sized Date field (day, month Year Low byte and Year High Byte consisting of 1 byte each), fixed sized entry's message length (1 byte) and a variable sized Entry message. Due to the variability of the record entry's message made the reason why we must use a variable sized random access database structure. It is also assumed that the mode's UI states that entries would never be edited nor deleted.

4. Occasion's Mode Database Header:

All offsets with respect to the base address of the occasions mode database:

Offset	Offset Name	Description
0	Allocation Size	Allocation size info. 2-bytes
2	Database Size	Database Size Info 2-bytes
4	Occasions Mode Specific Header Info Size	Occasion Specific Header Info Size. This should have a value of 10. 1-Byte
5	Total Number of Records	Number of Records in the database. This is the total number of records (Recurring + Non Recurring) 2-Byte
7	Number of records in the non-recurring bank.	Total number of records for non-recurring entries. 2-Byte
9	Number of records in the recurring bank	Total number of records for recurring entries. 2-Byte
11	Non-recurring entry last record number	Last Record number for a non-recurring entry. 2-Byte
13	Recurring entry last record number	Last Record number for a recurring entry. 2-Byte

18.9.Schedule Application

The following section describes the database structure for the Schedule application.

1. Application Database Data

The structure of the Schedule Mode application database data is tailored to the Data Link USB Database Design description Random-Variable database structure. One unique implementation that was done was to have a nested Random-variable database structure. This is done to serve a database of different schedule groups and a database of entries in each group.

2. The Entire Database structure.. a big picture

There is two-database structure that we should be dealing with in this mode. The Main Database structure, which consist of records pointing to the group schedules, and the Sub Database, which consist of records of each individual entries in a group. The graphical representation of these two databases is shown, the sub database is expanded as an image inside a record from the main database.

3. Main Database Record Structure:

Each record in the main database contains a sub-database with a random variable sized database structure.

4. Entry Record Structure:

TIME 2 bytes	Date 4 bytes	Message Length 1 byte	Entry Message

Above is a layout of what the Entry's record structure should consist of. Each entry has a fixed sized Time field; Fixed sized Date Field, the entry's message length and a variable sized Entry message. Due to the variability of the entry's message made the reason why we must use a variable sized random access database structure.

Time structure (2-bytes) should be in BCD where the lower byte address contains the minutes and the upper byte address contains the hour's information.

Date structure (4-bytes) should be in BCD with the day information on the lower byte and the DOW in the highest byte as shown below:

Day 1 byte	Month 1 byte	Year -Lo 1 byte	DOW 1 byte

The DOW should start as Sunday as the first day with a value of "0" and Saturday as the last day with a value of "6".

Message length varies between 1 to 101, which accounts for the sentinel character and the Entry's message should contain the sentinel character as the last character in the message.

5. Details of the Main Database Header

Offset	Offset Name	Description
0	SCHRALLOCATIONSIZEOFFSET	Stores the allocation size for the entire schedule mode database. 2-bytes
2	SCHRDATABASESIZEOFFSET	Stores the size of the entire database 2-bytes
4	SCHSPECIFICHEADERSIZEOFFSET	Stores the size of the schedule mode specific header information. The value should be 22. 1-byte
5	SCHTOTALGROUPRECORDSOFFSET	Stores the no of group records (number of group member description which means number of sub database) for the schedule mode database. This determines the amount of groups (or sub-database) the mode has. This is limited up to 250 group records only. 2 – bytes
7	SCHGROUPTYPEOFFSET	Stores the group type classification that the schedule mode should have which is used to identify which of the following fields are needed to compare/compute for the next occurring

		schedule in a group: 1 – DATE only 2 – DOW and Time only 3 – DATE and Time 1 - Byte
8	SCHSELECTMESSAGEPOSITIONLINE1	Start display position in the Line1 dot matrix for the select description message. (Refer to the LCD equates position). Value should be 0xF900 + starting column address in the LCD. 2-bytes
10	SCHSELECTMESSAGEPOSITIONLINE2	Start display position in the Line2 dot matrix for the select description message. (Refer to the LCD equates position). Value should be 0xFA00 + starting column address in the LCD. 2-bytes
11	SCHSELECTMESSAGELENGTHOFFSET	Stores the length of the message select setting. Maximum characters are 14. It should **not** include the sentinel character. 1-Byte
12	SCHSELECTMESSAGEOFFSET	Stores the selection message. 14-bytes (if the message is less than 14 characters, fill the other unused bytes with zeros)

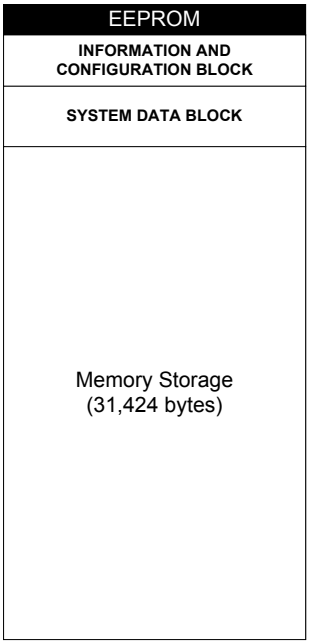
6. Details of the Entry Database (Sub-database) Header:

Offset	Offset Name	Description
0	SCHENTRYDBALLOCATIONSIZEOFFSET	Stores the allocation size for the sub-database. 2-bytes
2	SCHENTRYDBDATABASESIZEOFFSET	Stores the size of the sub-database 2-bytes
4	SCHENTRYDBSPECIFICHEADERSIZEOFFSET	Stores the size of the schedule mode specific header information. The value should be: 3 + ScheduleNameMessageLength 1-byte
5	SCHTOTALENTRYRECORDSOFFSET	Stores the total no of entry records for the sub-database. This determines the amount of entries the group database has. This is limited up to 250 entry records only. 2 – byte
7	SCHSCHEDULENAMEMESSAGELENGTHOFFSET	Stores the group schedule name message length. The value varies from 1 to 101. The message ends with a sentinel character. 1-byte
8	SCHSCHEDULENAMEMESSAGEOFFSET	Stores the schedule name message (should include the sentinel character)

18.10. EEPROM Utilization

The Data Link USB watch uses an EEPROM to mainly store application database and EEPROM based applications. It also stores information and configuration data that determines default watch operation during power up from reset.

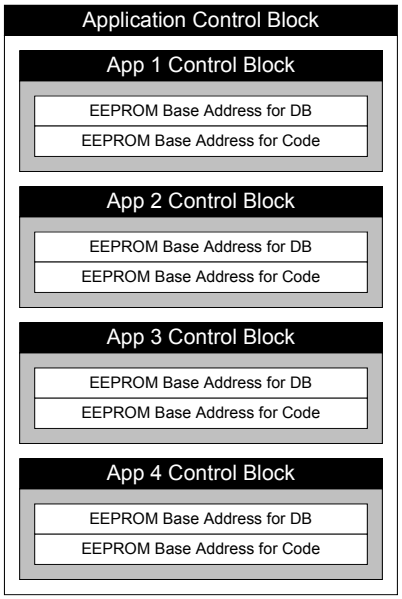
The figure below shows the basic allocation of the EEPROM.

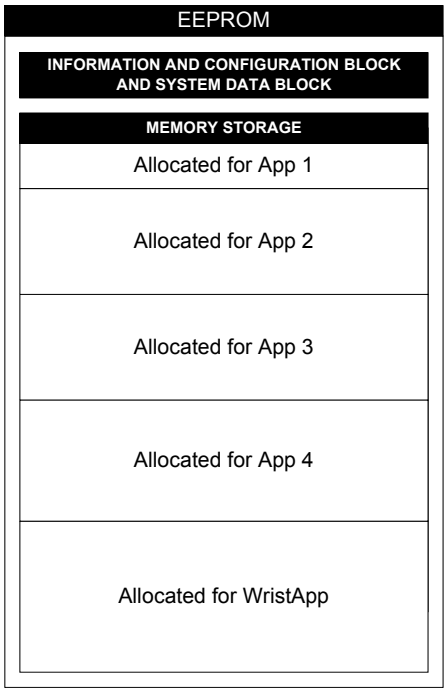


Application data or code is stored in a contiguous manner. Any adjustment to the size of an application data during run time will require a full reload of all the data blocks currently in the EEPROM. This prevents any fragmentation of memory to occur in the EEPROM.

When an application requests EEPROM storage, the system will always allocate a contiguous section that begins in a 64-byte page and the size will be multiple of 64. This minimizes the time it takes for the watch to write the data in EEPROM should the PC download the database in 64-byte chunks through the communications module (average write time of 5ms). If we do not force this restriction using the EEPROM page size, it would take an average of 10ms to write the 64-byte data packet.

The figure below shows the relationship between the Application Control Block of the Kernel and the external memory structure.





18.11. Information and Configuration Block (ICB)

This section shows the usage of the 64-byte EEPROM block that is used to store system information as well as configuration that affects how the watch will operate.

When the watch powers up, it will check if the EEPROM has been programmed before by looking at the information stored in a pre-determined EEPROM page block. The first three bytes should be "851" and the checksum is correct for the entire block. If EEPROM is not programmed or the checksum is incorrect, it will download default parameters into the information block and resets the watch.

If the EEPROM is already programmed after a reset condition, it will configure the watch software to operate according the parameters set in this information and configuration block.

Some configuration data should be stored in the appropriate system variable. Model and Serial information structures are used only by the PC.

The Information and Control Block stores the following information:

EEPROM PAGE 0[1]		
Offset	Info or Control Name	Description
0	SEG_8	Model and Version Number Byte 0
1	SEG_5	Model and Version Number Byte 1
2	SEG_1	Model and Version Number Byte 2
3	SEG_0	Model and Version Number Byte 3
4	SEG_0	Model and Version Number Byte 4
5	SEG_0	Model and Version Number Byte 5
6	0x00	Serial Number Byte 0, 1, 2
7	0x00	Serial Number Byte 1

[1] Values shown are not initial values.

8	0x00	Serial Number Byte 2
9	0x00	Bit 0: reserved Bit 1: 0 = CrownSet, 1 = RingSet Bit 2: 0 = Reserved Bit 3: 0 = Mask Part, 1 = Flash Part Bit 4: reserved Bit 5: reserved Bit 6: reserved Bit 7: reserved
10	0x77	Bit 7..4: Battery Detection Setting (0x0 – 0xF) Bit 3..0: LCD Contrast Parameter (0x0 to 0xF)
11	0x??	Bit 7..0: Scroll Speed (put more info)
12	0x00	Password Byte 0
13	0x00	Password Byte 1
14	0x00	Maximum Setting Character
15	0x04	NightMode Activation/Deactivation Duration
16	0x00	EEPROM Size (low Byte of 16-bit number)
17	0x20	EEPROM Size (high Byte of 16-bit number)
18	0x00	Periodic Task Control Block Bit 7: Daily Task Available in EEPROM Bit 6: Hourly Task Available in EEPROM Bit 5: Minute Task Available in EEPROM Bit 4..0: Must be zero.
19	0x00	Periodic Task EEPROM Location (low-byte 16-Bit)
20	0x00	Periodic Task EEPROM Location (high-byte 16-Bit)
21	0x00	BG Config
22	0x00	POR Application Index Pointer 0 (for TOD)
23	0x00	POR Application Index Pointer 1 (for COMM)
24	0x00	POR Application Index Pointer 2 (0xFF if unused)
25	0x00	POR Application Index Pointer 3 (0xFF if unused)
26	0x00	POR Application Index Pointer 4 (0xFF if unused)
27	0x00	POR Application Index Pointer 5 (0xFF if unused)
28	0x00	POR Application Index Pointer 6 (0xFF if unused)
29	0x00	POR Application Index Pointer 7 (0xFF if unused)
30	0x00	POR Application Index Pointer 8 (0xFF if unused)
31	0x00	POR Application Index Pointer 9 (0xFF if unused)
32	0x00	POR Application Index Pointer 10 (0xFF if unused)
33	0x00	POR Application Index Pointer 11 (0xFF if unused)
34	0x00	POR Application Index Pointer 12 (0xFF if unused)
35	0x00	POR Application Index Pointer 13 (0xFF if unused)
36	0x00	POR Application Index Pointer 14 (0xFF if unused)
37	0x00	POR Application Index Pointer 15 (0xFF if unused)
38	0x00	Usage Tracking: 0x00: usage tracking data is invalid non 0x00: usage tracking data is valid
39	0x00	LCD Contrast for Message Scrolling (0x0 to 0xF)
40	0x00	reserved
41	0x00	reserved
42	0x00	reserved
43	0x00	reserved
44	0x00	reserved
45	0x00	reserved
46	0x00	reserved

47	0x00	Checksum (8-bit)
48	0x00	PC-Watch Synchronization ID (must be 0x00 during POR)
49	0x00	PC-Watch Synchronization ID
50	0x00	PC-Watch Synchronization ID
51	0x00	PC-Watch Synchronization ID
52	0x00	PC-Watch Synchronization ID
53	0x00	PC-Watch Synchronization ID
54	0x00	PC-Watch Synchronization ID
55	0x00	PC-Watch Synchronization ID
56	0x00	PC-Watch Synchronization ID
57	0x00	PC-Watch Synchronization ID
58	0x00	PC-Watch Synchronization ID
59	0x00	PC-Watch Synchronization ID
60	0x00	PC-Watch Synchronization ID
61	0x00	PC-Watch Synchronization ID
62	0x00	PC-Watch Synchronization ID
63	0x00	PC-Watch Synchronization ID (Write 0xFF)

Description

Offset	Notes
0	**Firmware Model Number (Offset 0 to 2).** **Firmware Revision Level (Offset 3 to 5).** *These bytes should match with the 6-byte firmware model and revision number located in ROM. If a mismatch is detected, the firmware will program a default ICB into EEPROM.*
6	**Serial Number (3-Bytes)** Manufacturing option to provide a unique serial number for the watch. *Not used by firmware.*
9	**(i) Software Operation Bond Option** Specifies how the watch will operate based on the type of hardware the firmware is used on. ***Offset 9:*** • *Bit 1: 0 = CrownSet, 1 = RingSet* • *Bit 2: Reserved* • *Bit 3: 0 = Mask Part, 1 = Flash Part* **Offset 10:** • *Bit 7..4: Battery Detection Setting (0x0 – 0xF)* • *Bit 3..0: LCD Contrast Parameter (0x0 to 0xF)*
11	**(ii) Scroll Speed** *A computed value that is used to setup a timing parameter to generate the required frequency for message scrolling.* Formula: *TBD*
12	**(iii) Password**

	Specifies the 2-byte character string required for verifying a valid password. The character specified here must be less than the value specified in the Maximum Setting Character.

14	**(iv) Maximum Setting Character**
	Specificies the maximum character index that can be used during a character edit operation.
	
	Typical values are:
	
	`DM5_MACRON equ 86`
	`DM5_TILDE equ 68`

15	**(v) NightMode Activation/Deactivation Duration** *Time in seconds before manual activation or deactivation of the nightmode state is toggled.*
16	**EEPROM Size (16-bit)** Indicates the size of the external memory chip attached to the system. For a 32KB chip, the value stored in this offset is 08000.
18	**(vi) Periodic Task Control Block** *Indicates that a valid periodic task code image is stored in EEPROM. On POR, the system will map out the area used by the periodic task to be used by application as data storage.* It will use the flags set in this offset on what task to load and when to load the task into the common code overlay area. • *Daily Task Available in EEPROM (bit 7). Load and execute daily task code every 12AM.* • *Hourly Task Available in EEPROM (bit 6). Load and execute hour task code every hour.* • *Minute Task Available in EEPROM (bit 5). Load and execute minute task code every minute.*
19	**Periodic Task EEPROM Location (16-bit)** Specifies the 16-bit location in EEPROM space where the periodic task code is stored. The code image must be stored at the bottom of the external memory heap. The system will not map out the area used by the periodic task code from being used as application database memory. This will be used only if the Periodic Task Control Block is not zero.
21	**(vii) Business Group Watch Configuration** Reserved
22	**Pointer offset to TOD Parameter Table.** *Do not modify.*
23	**Pointer offset to COMM Parameter Table.** *Do not modify.*
24	**Default POR Application List (offsets 24 to 38)**

	Pointer offsets (16-bit) to a Parameter Pointer List to applications that will be initialized/activated during POR. The application must support the event COREEVENT_PORINIT to be included in this list.
	
	The available applications are:
	
	`Time of Day = 0`
	`Communications = 2,`
	`Chrono = 4,`
	`Timer = 6,`
	`Interval Timer = 8,`
	`Alarm = 10,`
	`Appointment = 12,`
	`Note = 14,`
	`Option = 16,`
	`Occasion = 18,`
	`Contact = 20,`
	`Schedule = 22,`
	`Synchro Timer = 24,`
	
	Store only in the integer values in the ICB.

38	**Usage Tracking.** *By default, this is 0x00. When the TOD year is modified, this offset will be changed to a value other than 0x00.*
39	**LCD Contrast for Message Scrolling**. *Specifies the register setting for the LCD Contrast Control Byte.*
47	**ICB Checksum.** *Negate the LSB of the sum of all the bytes from offset 0 to 46.*
48	**(viii) PC-Watch Synchronization ID** *A value stored in both the PC and watch to indicate a unique communication session ID. This section will be cleared to 0x00 during firmware POR.* *A mismatch between the PC and the watch will indicate that the current watch data was loaded from a different PC.*

18.12. EEPROM-Based Application Code Structure

The kernel will support multiple active EEPROM based application. The application will be executed after the kernel loads the required code blocks into an overlay area located in internal memory. Although they are EEPROM-based, the ASD section is required to be located in internal memory. This prevents swapping back to EEPROM the current ASD session to make way for another EEPROM-based application.

The Kernel will allocate an overlay area of about 900 bytes. All EEPROM based applications must be designed with this limitation.

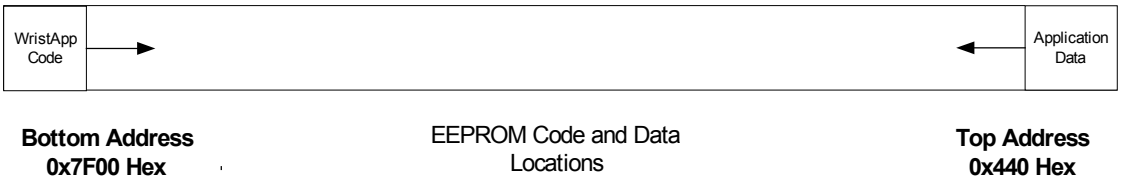
The structure is required to allow the Kernel to locate and load into the overlay area the required code block.

<table>
<tr><th colspan="3">EEPROM-BASED APPLICATION STRUCTURE</th></tr>
<tr><td colspan="3">Allocation Size (2-byte)</td></tr>
<tr><td colspan="3">Database Size (2-byte)</td></tr>
<tr><td colspan="3">App Specific Header Size (1-byte)</td></tr>
<tr><td colspan="3">Number of Records (2-byte)</td></tr>
<tr><td colspan="3">Remaining App Specific Header
(variable size up to 253 bytes)</td></tr>
<tr><td colspan="3">Record #0: COMMON CODE Block 16-Bit EEPROM Address</td></tr>
<tr><td colspan="3">Record #1: STATE 0 CODE Block 16-Bit EEPROM Address</td></tr>
<tr><td colspan="3">Record #2: STATE 1 CODE Block 16-Bit EEPROM Address</td></tr>
<tr><td colspan="3">Record #3: STATE 2 CODE Block 16-Bit EEPROM Address</td></tr>
<tr><td colspan="3">Record #4: STATE 3 CODE Block 16-Bit EEPROM Address</td></tr>
<tr><td>16-Bit Record Size</td><td>8-Bit Reserved</td><td>COMMON CODE BLOCK</td></tr>
<tr><td>16-Bit Record Size</td><td>8-Bit Reserved</td><td>STATE 0 CODE BLOCK</td></tr>
<tr><td>16-Bit Record Size</td><td>8-Bit Reserved</td><td>STATE 1 CODE BLOCK</td></tr>
<tr><td>16-Bit Record Size</td><td>8-Bit Reserved</td><td>STATE 2 CODE BLOCK</td></tr>
<tr><td>16-Bit Record Size</td><td>8-Bit Reserved</td><td>STATE 3 CODE BLOCK</td></tr>
</table>

18.13. Positioning Databases in the EEPROM

Now that the database formats have been described for each application, there needs to be a description of how this information sits in the watch's EEPROM.

The standard Timex Data Link USB products contain 32768 Kb of EEPROM. The actual usable amount is 31,424 Kb (the reason why will be explained later)



The above diagram displays how the EEPROM is filled with data. The bottom (or left) address (0x7F00 hexadecimal) is where code for WristApp applications is added. Conversely, the code for the databases is added to the top (or right) address (0x0440). The two areas eventually converge thus maxing out the memory.

The bottom 0x100 bytes (0x7F00 – 0x7FFF) are not usable because there are special routines programmed at the factory within that area. These routines, located in that area of EEPROM, are reserved for patch code.

The top 0x440 bytes are used for the watch to determine what modes are present in the watch, as well as the watch's configuration. Please refer to the ICB section for more information.

When setting up the allocation size for each database, in order to speed up download speed, it is imperative that everything is placed on a 64 byte (page) boundary.