M851 – USB Data Link Ironman

M851 “Packager” API

Design & Usage Guide

Specification Number: 1-S-XXXX-H

April 11, 2003

Brigham W. Thorp
Timex Corporation

DOCUMENT REVISION HISTORY

REVISION: A	DATE: 7/17/2002	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
All	Created document.

REVISION: B	DATE: 8/08/2002	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
17	GETITEM types referreed to Occasions
Various	Section references were updated
14	Added enum definition for ALARM_STATUS

REVISION: C	DATE: 8/13/2002	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
Various	Changes to some of the structures and added some additional comments

REVISION: D	DATE: 8/23/2002	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
6	TUDLAddTODItem was missing the DateFormat specifier, so this was added

REVISION: E	DATE: 9/10/2002	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
36	Added more information about creating sound data for sending to the API. Used a modified version of the WristApp SDK documentation.
15,16	Modifications to the way that the TUDLGetAlarmItem and TUDLGetApptItem functions are defined. This was due to ongoing changes to the packager.

REVISION: F	DATE: 10/27/2002	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
26	Modified values for ScheduleType
15	Added description for TUDLSetTimeLine
8	Added TUDLForceFullDownload API call
30	Removed TIMEX_ERR_INV_RESOURCE since PIM's now handle their own resource checking

REVISION: G	DATE: 02/05/2003	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
5	Added TUDLSetAppTitle API call

REVISION: H	DATE: 04/11/2003	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
38	Changed sounds section to refer to frequencies as Hz…not KHz.

Table of Contents

1. INTRODUCTION

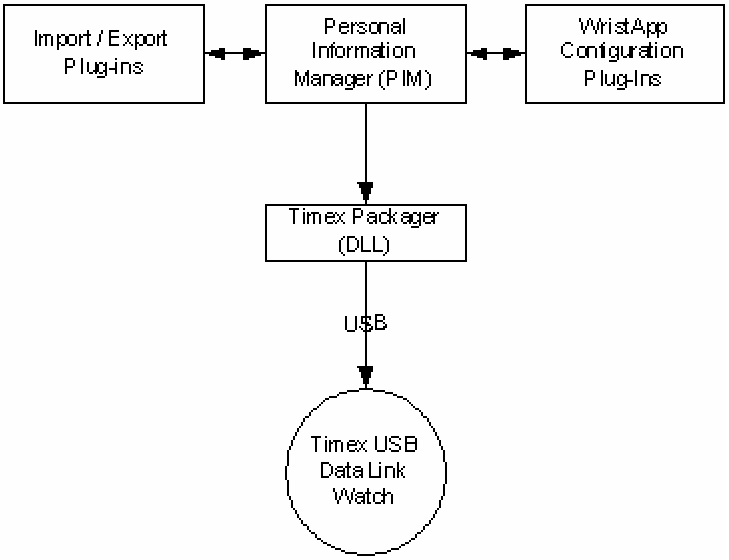
1.1. Scope

This document describes the API's (application programming interfaces) that are present to communicate between the PC and the USB Data Link watch. These API's may be used to create an application that interfaces to the watch similarly to the way the Timex software interfaces with the watch.

1.2. Watch 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. Once connected, the software is then able to read and write information to and from the watch.

The standard watch software consists of multiple modules that all work together. The diagram below shows these modules and how they are connected.


Import / Export Plug-ins
Personal Information Manager (PIM)
WristApp Configuration Plug-Ins
Timex Packager (DLL)
USB
Timex USB Data Link Watch

The PIM software that comes with the watch allows you to enter and store information such as phone numbers and appointments and then download them to the watch. This software also controls some of the other settings such as sounds, options, and modes present in the watch.

The Import and Export DLL's allow developers to write programs that can import and export data to and from watch modes. For example, an import DLL could be written to import appointments from Lotus Domino into the appointment editing facility within the PIM. This module could also support exporting appointments back to Lotus Domino

The Application Configuration DLL's are plug-in programs that are distributed by WristApp developers to allow you to configure certain options for the specified WristApp. For example, the World Time WristApp that ships with the product has a configuration DLL that allows you to configure the time zones that are present in the World Time application.

The packager is a DLL written in 'C++' that handles all low-level USB communication between the watch and the PC. By utilizing the packager, it is possible to add and remove data and applications to and from the watch respectively by following a certain set of API calls. The file has the name TUCP.DLL which stands for **T**imex **U**SB **C**ommunications **P**rotocol.

This document describes how to access functions within the TUCP DLL, which contains all of the low-level USB communications as well as the API's to modify data in the watch.

2. APPLICATIONS

The Timex USB watch contains the following applications in ROM:

- **Time of Day** – This is the normal timekeeping display. Three time zones can be tracked and each time zone supports displaying the week of the year, or the day of the week. Also, each time zone can have different time and date formats such as 12 hour versus 24 hour time display, and MM-DD-YY, DD-MM-YY, or YY-MM-DD formats.
- **Alarms** – The alarm mode can store multiple alarms (as much as the watch memory will allow) with each alarm capable of being set to a different time. Typically, alarms are used to wake you up or remind you to take your medication.
- **Appointment** – The appointment mode is similar to alarm mode, however you can set an appointment to notify you before the appointment begins. For example, you might want a 30 minute prenotification for a 3:00 doctor's appointment to give you time to drive there.
- **Occasion** – Occasions can be set to keep track of birthdays, anniversaries, holidays, vacations.
- **Contact** – Contact mode is where you can store names, phone numbers, and other information such as home or work addresses as well as e-mail addresses.
- **Schedule** – Schedule mode is a mode that allows you to store schedules of items such as television schedules, sports team schedule, or class schedules. The schedules support either Day of the week and Time, Date and Time, or Date only formats.
- **Note** – Note mode can be used to store notes up to 100 characters long. Notes can be used to store phone numbers entered manually on the watch, as well as credit card numbers and passwords as an example.
- **Chronograph** – Chronograph mode is a stopwatch that supports storing and saving workouts.
- **Countdown Timer** – Countdown Timers allows you to set a time to countdown from. When the time expires, an alert is sounded.
- **Interval Timer** – Interval Timers are very similar to countdown timers, however when the first timer in sequence completes, the next timer in sequence that is not a zero is started automatically.
- **Option** – Option mode lets you set items in the watch such as the Night-Mode and Hourly chime functionality, as well as the button beep setting.
- **Synchro Timer** – The Synchro Timer starts automatically when the chronograph or timer (countdown or interval) is started. It continues to count up even when the chronograph and/or timer is stopped.

The above applications contain fixed code in the watch, however they can be enabled or disabled as you wish using specific API calls. The above watch modes use three types of memory. First is the ROM where all of the built-in application code resides. Note that disabling ROM based application will have no effect on ROM based memory since ROM is fixed for the device. The databases for the above applications are all stored in EEPROM. Finally, each application uses a certain amount of RAM for processing, so RAM can also be saved by not enabling certain applications.

3. INSTALLING THE SDK

To setup the SDK, you will need to add the TUCP.LIB to your project if you are using Visual C++ 6.0. In addition, the TUCP.H file should be included as it defines the structures necessary for using the API's. Once you have compiled your application, the TUCP.DLL should be placed in the same directory as your executable.

If you are using Visual Basic, you must create a Declare Function XX entry to access the API calls. Please refer to the Visual Basic section for information on how to define the API's.

4. WRISTAPPS

Other applications, called WristApps, may be downloaded to the watch to add functionality. WristApps contain specialized code to run in the watch, and in some cases have an associated database. Both the code and database are stored in EEPROM once they are downloaded. When the WristApp becomes active, the code is copied from the EEPROM and runs in RAM.

Adding and deleting data from the watch has an effect on memory. Please check the device documentation to determine how much EEPROM memory is available for the product (the initial version of the device contains 32 Kilobytes of EEPROM).

Please note that certain watch modes use resources. The resources available in this watch are:

Resource Type	Maximum Allowed	Offset (Bytes)	Notes
Time of Day	4	1	Unavailable to other applications. Used by built in Time of Day mode (1 for each time zone and 1 for the popup clock)
Backup	2	2	Typically, 1 for appointment and 1 for alarm mode.
Time Zone Check	5	3	2 for appointment (peek and popup), 1 for alarm, and 2 extra
Timer	3	4	2 used by interval timer and 1 for the countdown timer
Stopwatch	2	5	2 for chronograph mode (1 for split, 1 for lap)
Synchro	1	6	1 for Synchro mode

If you try to instantiate an application and you are using more resources than available, the communications will fail.

WristApp's have a minimum of two files associated with them. First is the code file, which contains all of the executable code that runs when the WristApp becomes active in the watch. Next is the parameter file. The parameter file contains information required by the watch that determines how the watch behaves in the system and its resource requirements. Typically the filenames for the two files are WristAppName_code_018.bin and WristAppName_par_018.bin for the code file and parameter file respectively. The 018 refers to the fact that the WristApp was compiled to execute only in version XXX018 of the firmware.

When downloading WristApps, the resource usage needs to be checked in the parameter file. The offsets specified above are the locations (from a zero based index) into the parameter file where the resource usage is stored. Please refer to the WristApp Design Guide for more information.

5. PERIODIC TASKS

In addition to WristApps, another application type called a periodic task may be downloaded to the watch. These are very similar to WristApps, however there is no parameter file associated with it (only code). The periodic task is called every second, minute, hour, and day rollover. The device comes with periodic tasks for changing the way the minute is displayed when changing. Other periodic tasks that could be written are programs that handle the automatic change from Daylight Savings Time to Standard Time and vice-versa.

6. APPLICATION PROGRAMMING INTERFACES

Typically, the order of calling the API functions is to read all of the data first and process that data according to your applications data (e.g. merging or displaying the modified items). Next, all of the applications that the user would like to put in the device are then added. Once the applications have been added, all of the items for the particular mode are then added to the application. Finally, the WristApps and their associated data are then added via the API.

Finally, once all of these calls have been performed, the data is written to the watch. The packager handles partitioning the watch memory so that modifications to a few items will not require downloading all modes and data back down to the watch.

Many of the API calls have reserved variables. These variables are not currently used, so passing a 0 to them is necessary. The reserved variables are setup for future expansion of the API set.

Most of the API's refer to structures that are defined in section 6.35. Also, many enumerations are also used in the following API calls, and these are defined in section 6.36. You may want to bookmark these sections for easy reference.

Please note that the following API definitions used to communicate with the device are shown with C/C++ references.

6.1. TUDLSetDeviceType

int TUDLSetDeviceType(WORD wNewDeviceType);

Purpose:

Selects a new device to communicate with. This should be done when the program first starts up. The Timex Ironman USB watch is selected by default, so if this is the only device you will be using, then this call does not have to be made.

Parameters:

wNewDeviceType – Value specifying the new device type. DL_DEV_851 (which has the value of 1) is an example defined parameter.

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

TUDLSetDeviceType(DL_DEV_851);

6.2. TUDLSetAppTitle

int TUDLSetDeviceType(char* szTitle);

Purpose:

Allows you to create a custom application title which is shown on the top of dialog boxes.

Parameters:

szTitle – Null terminated string identifying the application title.

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

TUDLSetAppTitle("My application");

6.3. TUDLSetOption

int TUDLSetOption(WORD wSetWhat, DWORD dwToWhat, DWORD dwExtra=0);

Purpose:

Sets options for the selected device

Parameters:

wSetWhat – The option you wish to set. These are defined as the DL_OPT value. Refer to Section 9 for information about these options.
dwToWhat – The value that you want to set the option to
dwExtra – Some options may require more than one value to set. The dwExtra variable provides this capability.

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
int nRc;
// Set the chrono format to have the lap on top and split on bottom of the display
nRc = TUDLSetOption(DL_OPT_DEV_CHRONO_FORMAT, CHRONOFORMAT_LAP_SPLIT, 0);
// Set the chrono to have 200 laps
nRc = TUDLSetOption(DL_OPT_DEV_CHRONO_NUM, 200, 0);
```

6.4. TUDLGetOption

int TUDLGetOption(WORD wGetWhat, LPDWORD lpdwRetVal, DWORD dwExtra=0);

Purpose:

Gets options from the selected device. These can include the maximum number of entries allowed for a given mode's items, as well as getting the total amount of memory used by the device.

Parameters:

wGetWhat – The option you wish to retrieve. These are defined as the DL_OPT values. Refer to Section 9 for information about these options.
lpdwRetVal – This is the option value returned from the packager
dwExtra – Some options may return more than one value. The dwExtra variable provides this capability

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
DWORD dwAlmMsgLen;
// Get the maximum length of the alarm mesages
TUDLGetOption(DL_OPT_DEV_ALARM_LEN, &dwAlmMsgLen);
CString sMsg;
// Make sure we only take the leftmost max characters from the current alarm message
sMsg = sAlarmMessage.Left(dwAlmMsgLen);
```

6.5. TUDLSendData

int TUDLSendData(HWND hWnd);

Purpose:
Build the transmission file and send it out the selected port

Parameters:
hWnd - handle to the parent window

Returns:
0 - Success
< 0 – Error
> 0 - Warning

Example:

```
int nRc = TUDLSendData(hWnd);           // Send all data
// If we receive success, close the window
if (nRc == TIMEX_SUCCESS)
      OnOK();
else
      MessageBox(hWnd, "There was an error during transmission ", "", MB_OK);
```

6.6. TUDLGetData

int TUDLGetData(HWND hWnd);

Purpose:
Retrieves the data from the device from the selected port

Parameters:
hWnd - handle to the parent window

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:

```
int nRc = TUDLGetData(hWnd);
if (nRc == TIMEX_SUCCESS)
{
      ReadAndDisplayChrono(); // Display the chrono data stored in the watch
}
```

6.7. TUDLGetDeviceVersion

int TUDLGetDeviceVersion(LPSTR lpszDevice1ID, WORD wDev1IDLen,
LPSTR lpszDevice2ID, WORD wDev2IDLen);

Purpose:
Returns the IDs of the most recently connected device

Parameters:
lpszDevice1ID - pointer to a string to receive the main device ID.
wDev1IDLen - maximum number of characters to copy into the string pointed-to by lpszDevice1ID.
lpszDevice2ID - pointer to a string to receive the secondary device ID.
wDev2IDLen - maximum number of characters to copy into the string pointed-to by lpszDevice2ID.

A NULL character ('\0') will always be installed in the last character position.

Returns:
0 - Success
< 0 - Error

> 0 - Warning

Example:

```
lpstrDev1 = new char[256];
lpstrDev2 = new char[256];
TUDLGetDeviceVersion(lpstrDev1, 256, lpstrDev2, 256);
if (strlen(lpstrDev1) != 0)
      m_sCypressVer.Format("Epson Version %s", lpstrDev1);
if (strlen(lpstrDev2) != 0)
      m_sEpsonVer.Format("Cypress Version %s", lpstrDev2);
```

6.8. TUDLShowDialog

int TUDLShowDialog(BOOL bShow);

Purpose:

Displays or hides the progress dialog that is seen during communication. Call before communication occurs (not during).

Parameters:

bShow – TRUE if the dialog is to be displayed during communication. FALSE if the dialog is to be hidden. By default, the packager shows the progress dialog.

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
TUDLShowDialog(FALSE);      // Hide the communications dialog
```

6.9. TUDLForceFullDownload

int TUDLForceFullDownload(void);

Purpose:

Forces the packager to perform a complete download and bypass the partial download feature. The partial download feature reduces the time it takes to download, however in some cases it may be necessary to download completely (e.g. when a new version of a WristApp is being downloaded).

Parameters:

None

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
Int nRet = TUDLForceFullDownload ();
```

6.10.TUDLAddItem

int TUDLAddItem(AppIndex nAppIndex, BYTE bAppInstance, void* pItemStruct);

Purpose:

Adds an item to the packager for download to the device

Parameters:

nAppIndex - type of the item being added to the database. Refer to Section 6.35 for the AppIndex definition
bAppInstance - instance of the application
pItemStruct - application specific item structure

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
CAlarm pItem;            // Created from database
sMsg = pItem->m_sMessage.Left(m_dwAlmMsgLen);
m_AlarmItem.wItemID = wItemID++;
m_AlarmItem.pszText = (LPSTR)(LPCTSTR)sMsg;
m_AlarmItem.cchTextMax = sMsg.GetLength();
m_AlarmItem.nStatus = (AlarmStatus)pItem->m_nEnable;
m_AlarmItem.nFreq = (AlarmFrequency)(pItem->m_nFreq+1);
m_AlarmItem.bHour = (BYTE)pItem->m_oDateTime.GetHour();
m_AlarmItem.bMinute = (BYTE)pItem->m_oDateTime.GetMinute();
TUDLAddItem(ALARM_APP, nInstance, (void*)&m_AlarmItem);
```

6.11.TUDLAddTODItem

int TUDLAddTODItem(BYTE bAppInstance, BYTE bItemID, LPCSTR pszText, int cchTextMax, LONG lTimeOffset, TimeZoneIndex nTZIndex, TimeFormat nTimeFormat, DateFormat nDateFormat, BOOL boolDSTObserved, BOOL boolIsInDST, BOOL boolShowWeekNumber, BYTE b1stDayOfWeek);

Purpose:

Adds a Time of Day item to the packager for download to the device

Parameters:

bAppInstance - instance of the application
bItemID – the ID of the item you wish to add. Must be unique
pszText – the 3 character city code
cchTextMax – the length of the 3 character city code
lTimeOffset – the offset in seconds from GMT
nTZIndex – the index into the time zone table (leave 0)
nTimeFormat – the time format – 0 = AM/PM, 1 = 24 hour
nDateFormat – the date format – 0 = DMY, 1 = YMD, 2 = MDY
boolDSTObserved – TRUE if the time zone observes daylight savings time
boolIsInDST – TRUE if the time zone is currently in daylight savings time
boolShowWeekNumber – TRUE if the week number should be displayed or FALSE to display the day of week
b1stDayOfWeek – 0 if the first day of the week is Sunday. 1 if the first day of the week observed is on Monday

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
TUDLAddTODItem(0, 1, "NYC", 3, 18000, 0, 0, TRUE, TRUE, FALSE, 0)
```

6.12.TUDLAddChronoItem

int TUDLAddChronoItem(BYTE bAppInstance, WORD wItemID, WORD wWorkoutID, BYTE bType, BYTE bNumLaps, DATE dtDateTime, BYTE bLapNum, BYTE bHour, BYTE bMinute, BYTE bSecond, BYTE bHundredth);

Purpose:

Adds an chronograph item to the packager for download to the device

Parameters:

bAppInstance - instance of the application
bItemID – the ID of the item you wish to add. Must be unique
bWorkoutD – the ID of the workout you wish to add. Must be unique
bType – 1 if adding a workout, 2 if adding a lap for a workout
bNumLaps – 0 if adding a lap, otherwise any number < 200 if adding a workout
dtDateTime – The date of the workout
bLapNum – The current lap number
bHour – The current hour for the current split
bMinute – The current minute for the current split
bSecond – The current second for the current split
bHundredth – The current hundredth for the current split

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
COleDateTime timeNow;
timeNow = COleDateTime::GetCurrentTime();
DATE dtDate = DATE(timeNow);
// Add a workout with 2 laps/splits
// Note that each lap is a running total of all the previous laps plus the current lap. This is defined as a split
TUDLAddChronoItem(0, 0, 0, 1, 2, dtDate, 1, 0, 0, 3, 69);
TUDLAddChronoItem(0, 1, 0, 2, 2, dtDate, 1, 0, 0, 2, 07);
TUDLAddChronoItem(0, 2, 0, 2, 2 dtDate, 2, 0, 0, 3, 69);
// Add a workout with 1 lap/split
TUDLAddChronoItem(0, 0, 1, 1, 1, dtDate, 1, 0, 0, 1, 58);
TUDLAddChronoItem(0, 1, 1, 2, 1, dtDate, 1, 0, 0, 1, 58);
```

6.13.TUDLAddTimerItem

int TUDLAddTimerItem(BYTE bAppInstance, WORD wItemID, LPCSTR pszText, int cchTextMax, BYTE bHour, BYTE bMinute, BYTE bSecond, TimerAction nAction, BYTE bNumOfReps, BYTE bOptions);

Purpose:

Adds a Timer item to the packager for download to the device

Parameters:

bAppInstance - instance of the application
wItemID – the ID of the item you wish to add. Must be unique
pszText – The text of the timer message
cchTextMax – The size of the timer message in pszText
bHour – The number of hours to set for the current timer
bMinute – The number of minutes to set for the current timer
bSecond – The number of seconds to set for the current timer
nAction – 0 to stop at end, 1 to repeat at end, or 2 to start the chrono at the end
bNumOfReps – 0 when adding a standard timer
bOptions – 0 for no halfway reminder alert, 1 to enable the halfway reminder alert. The halfway reminder alerts you when the timer has reached the halfway point, if the timer is set for more than 1 minute.

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:
TUDLAddTimerItem(0, 1, “TIMER #1”, 8, 1, 0, 0, 0, 0, 1);

6.14.TUDLAddIntTimerItem

int TUDLAddIntTimerItem(BYTE bAppInstance, WORD wItemID, LPCSTR pszText, int cchTextMax, BYTE bHour, BYTE bMinute, BYTE bSecond, TimerAction nAction, BYTE bNumOfReps, BYTE bOptions);

Purpose:
Adds an Interval Timer item to the packager for download to the device

Parameters:
bAppInstance - instance of the application
wItemID – the ID of the item you wish to add. Must be unique
pszText – The text of the timer message
cchTextMax – The size of the timer message in pszText
bHour – The number of hours to set for the current timer
bMinute – The number of minutes to set for the current timer
bSecond – The number of seconds to set for the current timer
nAction – 0 to stop at end, 1 to repeat at end, or 2 to start the chrono at the end
bNumOfReps – The number of times to repeat each series of timers
bOptions – 0 for no halfway reminder alert, 1 to enable the halfway reminder alert

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
TUDLAddIntTimerItem(0, 1, “TIMER #1”, 8, 1, 0, 0, 0, 2, 1);

6.15.TUDLAddAlarmItem

int TUDLAddAlarmItem(BYTE bAppInstance, WORD wItemID, LPCSTR pszText, int cchTextMax, AlarmStatus nStatus, AlarmFrequency nFreq, BYTE bHour, BYTE bMinute, DATE dtReserved1);

Purpose:
Adds an Alarm item to the packager for download to the device

Parameters:
bAppInstance - instance of the application
wItemID – the ID of the item you wish to add. Must be unique
pszText – The text of the alarm message
cchTextMax – The size of the alarm message in pszText
nStatus – Please refer to the AlarmStatus enum in Section 6.35 for information about this value
nFreq – Please refer to the ALarmFrequency enum in Section 6.35 for information about this value
bHour – The hour to set for the alarm
bMinute – The minute to set for the alarm
dtReserved1 – Set to 0

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
TUDLAddAlarmItem(0, 1, “ALARM #1”, 8, 3, 1, 12, 0, 0);

6.16.TUDLAddApptItem

int TUDLAddAlarmItem(BYTE bAppInstance, WORD wItemID, LPCSTR pszText, int cchTextMax, DATE dtStartDateTime, ApptStatus nStatus, Frequency nFreq, Reminder nReminder, DATE dtReserved1, DATE dtReserved2);

Purpose:
Adds an Appointment item to the packager for download to the device

Parameters:
bAppInstance - instance of the application
wItemID – the ID of the item you wish to add. Must be unique
pszText – The text of the appointment message
cchTextMax – The size of the appointment message in pszText
dtStartDateTime – The starting date of the appointment
nStatus – Please refer to the enum for information about this value
nFreq – Please refer to the enum for information about this value
nReminder - Please refer to the enum for information about this value
dtReserved1 – Set to 0
dtReserved2 – Set to 0

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
TUDLAddAppttem(0, 1, “DOCTOR’S APPT”, 13, dtDate, 3, 0, 5, 0, 0);

6.17.TUDLAddNoteItem

int TUDLAddNoteItem(BYTE bAppInstance, WORD wItemID, LPCSTR pszText, int cchTextMax, BYTE bStatus);

Purpose:
Adds an Note item to the packager for download to the device

Parameters:
bAppInstance - instance of the application
wItemID – the ID of the item you wish to add. Must be unique
pszText – The text of the appointment message
cchTextMax – The size of the appointment message in pszText
bStatus – 0 = unused, 1 = used

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
TUDLAddNoteItem(0, 1, “Visa Card 1234-5678-1234-5678”, 29, 1);

6.18.TUDLAddOptionItem

int TUDLAddOptionItem(BYTE bAppInstance, BOOL boolNightModeEnabled, BOOL boolNightModeAuto, BYTE bNightModeOnMin, BYTE bNightModeOnHour, BYTE bNightModeOffMin, BYTE bNightModeOffHour, BYTE bNightModeToggleDuration, BOOL boolChimeEnabled, BOOL boolChimeAuto, BYTE bChimeOnHour, BYTE

bChimeOffHour, BOOL boolButtonBeepEnabled, BYTE bLastSetCharacter, LPCSTR pszPassword, int cchPasswordMax, BOOL boolApp1Timeline, BOOL boolApp2Timeline, DWORD dwReserved);

Purpose:
Adds an Option item to the packager for download to the device.

Parameters:
bAppInstance - instance of the application
boolNightModeEnabled – True if Night-Mode is enabled
boolNightModeAuto – True if Night-Mode is in auto mode
bNightModeOnMin – The minute when Night-Mode will come on when in auto mode
bNightModeOnHour – The hour when Night-Mode will come on when in auto mode
bNightModeOffMin – The minute when Night-Mode will go off when in auto mode
bNightModeOffHour – The hour when Night-Mode will go off when in auto mode
bNightModeToggleDuration – How long the user must press the Indiglo button before Night-Mode goes On or Off
boolChimeEnabled – True if the hourly chime is enabled
boolChimeAuto – True if the hourly chime is in auto mode
bChimeOnHour – The hour when the Hourly Chime will become active when in auto mode
bChimeOffHour – The hour when the Hourly Chime will turn off ehen in auto mode
boolButtonBeepEnabled – True if the button beep is enabled
bLastSetCharacter – Set to 69 for US, or 87 for European; last character found during crown editing of text
pszPassword – Up to a two character password used for protecting modes
cchPasswordMax – The length of the password text
boolApp1Timeline – Set to True to enable the first application time line
boolApp2Timeline – Set to True to enable the second application time line
dwReserved – Set to 0

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
TUDLAddOptiontem(0, FALSE, TRUE, 0, 20, 0, 8, 3, TRUE, FALSE, 8, 19, TRUE, 69, “YZ”, 2, TRUE, FALSE, 0);

6.19.TUDLAddOccasionItem

int TUDLAddOccasionItem(BYTE bAppInstance, WORD wItemID, LPCSTR pszText, int cchTextMax, DATE dtStartDateTime, BYTE bStatus, OccasionType nType, BOOL bIgnoreYear, DATE dtReserved1);

Purpose:
Adds an Occasion item to the packager for download to the device

Parameters:
bAppInstance - instance of the application
wItemID – the ID of the item you wish to add. Must be unique
pszText – The text of the appointment message
cchTextMax – The size of the appointment message in pszText
dtStartDateTime – The starting date for the occasion
bStatus – 0 = non-recurring, 1 = recurring
nType – Please refer to the OccasionType enum in Section 6.35 for information about this value
bIgnoreYear – set to TRUE to not use the year on an occasion item. This is used when you don’t know the starting date of an occasion
dtReserved1 – Set to 0

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
COleDateTime timeNow;

```
timeNow = COleDateTime::GetCurrentTime();
DATE dtDate = DATE(timeNow);
TUDLAddOccasionItem(0, 1, “Mom’s Birthday”, 14, dtDate, 1, 1, FALSE, 0);
```

6.20.TUDLAddContactItem

int TUDLAddContacttem(BYTE bAppInstance, WORD wItemID, LPCSTR pszNameText, int cchNameTextMax, LPCSTR pszTypeText, int cchTypeTextMax, LPCSTR pszPhoneText, int cchPhoneTextMax, BYTE bOptions);

Purpose:
Adds a Contact item to the packager for download to the device

Parameters:
bAppInstance - instance of the application
wItemID – the ID of the item you wish to add. Must be unique
pszNameText – The name of the contact
cchNameTextMax – The size of the text in pszNameText
pszTypeText – The text for the phone type (e.g. “WF” for work fax)
cchTypeTextMax – The size of the text in pszTypeText
pszPhoneText – The actual phone number
cchPhoneTextMax – The size of the text in pszPhoneText
bOptions – bit 0 = don't apply phone number formatting
bit 1 = display left arrow in the first pos of the main dot matrix
bit 2 = display right arrow at the end of the phone number
bit 3 = display left arrow in the top dot matrix (in place of phone type)

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:

```
TUDLAddContacttem(0, 1, “Ralph’s Pizza”, 13, “B”, 1, “800-555-3456”, 12, 0);
```

6.21.TUDLAddScheduleItem

int TUDLAddScheduleItem(BYTE bAppInstance, WORD wGroupID, LPCSTR pszGroupText, int cchGroupTextMax, LPCSTR pszGroupLabel, int cchGroupLabelMax, ScheduleType nType, WORD wItemID, LPCSTR pszText, int cchTextMax, DATE dtDateTime, int nDayOfWeek);

Purpose:
Adds an Occasion item to the packager for download to the device

Parameters:
bAppInstance - instance of the application
wGroupID – the ID of the group you wish to add. Must be unique and start with 0
pszGroupText – The group name text
cchGroupTextMax – The size of the group message in pszGroupText
pszGroupLabel – The watch label display for the group
cchGroupLabelMax – The size of the group label text in pszGroupLabel
nType – Please refer to the enum for information about this value
wItemID – The ID of the item within the current group that you wish to add
pszText – The text for the current item
cchTextMax – The size of the text in pszText
dtDateTime – The time and date of the item
nDayOfWeek – The day of the week for the current item

Returns:
  0 - Success
  < 0 - Error
  > 0 - Warning

Example:

```
COleDateTime timeNow;
timeNow = COleDateTime::GetCurrentTime();
DATE dtDate = DATE(timeNow);
TUDLAddScheduleItem(0, 0, “NBA”, 3, “BULLS”, 5, 3, 0, “VS KNICKS”, 9, dtDate, 0);
```

6.22.TUDLAddSoundItem

int TUDLAddSoundItem(BYTE bAppInstance, SoundType nItemID, int nSoundsMax, LPBYTE lpbSounds);

Purpose:
  Adds a Sound item to the packager for download to the device

Parameters:
  bAppInstance - instance of the application
  nItemID – Please refer to the SoundType enum in Section 6.35 for information about this value
  nSoundsMax – Number of bytes in the buffer pointed to by lpbSounds
  lpbSounds – A buffer containing the sound data. Please refer to the section on Sounds for more information.

Returns:
  0 - Success
  < 0 - Error
  > 0 - Warning

Example:

```
TUDLAddSoundItem(0, 0, “NBA”, 3, “BULLS”, 5, 3, 0, “VS KNICKS”, 9, dtDate, 0);
```

6.23.TUDLSetTimeLine

int TUDLSetTimeLine(BOOL bTimeeLine1, BOOL bTimeLine2);

Purpose:
  Enables or disables the timeline feature.

Parameters:
  bTimeLine1 – Set this to TRUE to enable the time line for the first instance of a mode that supports the time line.
  bTimeLine2 – Set this to TRUE to enable the time line for the second instance of a mode that supports the time line.

Returns:
  0 - Success
  < 0 - Error
  > 0 - Warning

Example:

```
TUDLSetTimeLine(TRUE, FALSE);        // Turn on the first time line and turn off the second time line
```

6.24.TUDLGetItem

int TUDLGetItem(AppIndex nAppIndex, BYTE bAppInstance, void* pItemStruct, GetItemType nItemType);

Purpose:
Retrieves an item from the packager after upload from the device

Parameters:
nAppIndex - type of the item being added to the database. Refer to Section 4.34 for the AppIndex definition
bAppInstance - instance of the application
pItemStruct - application specific item structure
nItemType – Which item to retrieve – Refer to the GetItemType enum in Section 6.35.

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:

```
CHRONO_ITEM cItem, cLastItem;
for (BYTE x = 0; x < 14; x++)
{
    int nIndex = 0;

    // Get each workout stored in the chrono database
    int nRet = TUDLGetItem(CHRONO_APP, x, &cItem, GETITEM_FIRST);
    while (nRet != TIMEX_ERR_NO_ITEM)
    {
            if (cItem.bType == 1)         // workout
            {
                    // process workout
            }
            else if (cItem.bType == 2)  // splits
            {
                    // process split
            }
            cLastItem = cItem;
            nRet = TUDLGetItem(CHRONO_APP, x, &cItem, GETITEM_NEXT);
    }
}
```

6.25.TUDLGetChronoItem

int TUDLGetChronoItem(BYTE bAppInstance, LPWORD wItemID, LPWORD wWorkoutID, LPBYTE bType, LPBYTE bNumLaps, DATE* dtDateTime, LPBYTE bLapNum, LPBYTE bHour, LPBYTE bMinute, LPBYTE bSecond, LPBYTE bHundredth, BOOL *bModified, GetItemType nItemType);

Purpose:
Retrieves a Chrono item from the packager after upload from the device

Parameters:
bAppInstance - instance of the application
bItemID – the ID of the item you are retrieving
bWorkoutD – the ID of the workout you are retrieving
bType – 1 if a workout was retrieved, or 2 if a lap was retrieved
bNumLaps – If a workout, the number of laps for the workout
dtDateTime – The date of the workout
bLapNum – The current lap number if retreving a lap
bHour – The current hour for the current lap
bMinute – The current minute for the current lap
bSecond – The current second for the current lap
bHundredth – The current hundredth for the current lap
bModified – 1 if the entry was modified on the watch, 0 otherwise.
nItemType – Used to control which item you are retrieving. Please refer to the GetItemType enum in Section 6.35 for information about this value

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
CHRONO_ITEM stItem;
BOOL bModified;
int nRet = TUDLGetChronoItem(0, &stItem.bItemID, &stItem.nWorkoutID, &stItem.bType, &stItem.bNumLaps,
                             &stItem.dtDateTime, &stItem.bLapNum, &stItem.bHour, &stItem.bMinute,
                             &stItem.bSecond, &stItem.bHundredth, &bModified, GETITEM_FIRST);
while (nRet != TIMEX_ERR_NO_ITEM)
{
    if (cItem.bType == 1)       // workout
    {
            // process workout
    }
    else if (cItem.bType == 2)  // splits
    {
            // process split
    }
    cLastItem = cItem;
    int nRet = TUDLGetChronoItem(0, &stItem.bItemID, &stItem.nWorkoutID, &stItem.bType, &stItem.bNumLaps,
                                 &stItem.dtDateTime, &stItem.bLapNum, &stItem.bHour, &stItem.bMinute,
                                 &stItem.bSecond, &stItem.bHundredth, &bModified, GETITEM_NEXT);
}
```

6.26.TUDLGetAlarmItem

int TUDLGetAlarmItem(int bAppInstance, int* wItemID, char pszText, int* nStatus, int* nFreq, int* bHour, int* bMinute, DATE* dtReserved1, BOOL *bModified, int nItemType);**

Purpose:

Retrieves an Alarm item from the packager after upload from the device

Parameters:

bAppInstance - instance of the application
wItemID – the ID of the alarm you are retrieving
pszText – The text of the alarm you are retrieving
nStatus – The status of the alarm you are retrieving. Please refer to the AlarmStatus enum for information about this value.
nFreq – The frequency of the alarm you are retrieving. Please refer to the AlarmFrequency enum for information about this value.
bHour – The hour of the alarm you are retrieving
bMinute – The minute of the alarm you are retrieving
dtReserved1 – Will be 0
bModified – 1 if the entry was modified on the watch, 0 otherwise.
nItemType – Used to control which item you are retrieving. Please refer to the GetItemType enum for information about this value

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
int wItemID;
char* pszText[250];
int nStatus;
```

```
    int nFreq;
    int nHour;
    int nMinute;
    DATE dtreserved1;
    BOOL bModified;

    int nRet = TUDLGetAlarmItem(0, &wItemID, &pszText, &nStatus, &Freq, &nHour, &nMinute, &dtReserved1, 0);
    // process the data here
    while (nRet != TIMEX_ERR_NO_ITEM)
    {
        nRet = TUDLGetAlarmItem(0, &wItemID, &pszText, &nStatus, &Freq, &nHour, &nMinute, &dtReserved1,
&bModified, 0);
        // process the data
    }
```

6.27.TUDLGetApptItem

int TUDLGetApptItem(int bAppInstance, int wItemID, char pszText, DATE* dtStartDateTime, int* nStatus, int* nFreq, int* nReminder, DATE* dtReserved1, DATE* dtReserved2, BOOL *bModified, int nItemType);**

Purpose:
Retrieves an Appointment item from the packager after upload from the device

Parameters:
- bAppInstance - instance of the application
- wItemID – the ID of the appointment you are retrieving
- pszText – The text of the appointment you are retrieving
- dtStartDateTime – The start date of the appointment
- nStatus – The status of the appointment you are retrieving. Please refer to the ApptStatus enum in Section 6.35 for info about this value.
- nFreq – The frequency of the appointment you are retrieving. Please refer to the Frequency enum in Section 6.35 for info about this value.
- nReminder – The prenotification reminder time for the appointment you are retrieving. Please refer to the Reminder enum in Section 6.35 for info about this value.
- dtReserved1 – Will be 0
- dtReserved2 – Will be 0
- bModified – 1 if the entry was modified on the watch, 0 otherwise.
- nItemType – Used to control which item you are retrieving. Please refer to the GetItemType enum in Section 6.35 for information about this value

Returns:
- 0 - Success
- < 0 - Error
- > 0 - Warning

Example:

```
    int wItemID;
    char* pszText[250];
    DATE dtStartDateTime;
    int nStatus;
    int nFreq;
    int nReminder;
    DATE dtReserved1;
    DATE dtReserved2;
    BOOL bModified;
    int nRet = TUDLGetApptItem(0, &wItemID, &pszText, &nStatus, &Freq, &nReminder, &dtReserved1,
                              &dtReserved2, &bModified, 0);
    // process the data here
    while (nRet != TIMEX_ERR_NO_ITEM)
    {
        int nRet = TUDLGetApptItem(0, &wItemID, &pszText, &nStatus, &Freq, &nReminder, &dtReserved1,
                              &dtReserved2, &bModified, 0);
        // process the data
    }
```

6.28.TUDLGetNoteItem

int TUDLGetNoteItem(BYTE bAppInstance, LPWORD wItemID, LPSTR* pszText, LPBYTE bStatus, BOOL *bModified, GetItemType nItemType);

Purpose:
    Retrieves a Note item from the packager after upload from the device

Parameters:
    bAppInstance - instance of the application
    wItemID – the ID of the note you are retrieving
    pszText – The text of the note you are retrieving
    bStatus – The status of the note you are retrieving; 0 = unused, 1 = used.
    bModified – 1 if the entry was modified on the watch, 0 otherwise.
    nItemType – Used to control which item you are retrieving. Please refer to the GetItemType enum in Section 6.35 for information about this value

Returns:
    0 - Success
    < 0 - Error
    > 0 - Warning

Example:

```
    NOTE_ITEM stItem;
    BOOL bModified;
    int nRet = TUDLGetNoteItem(0, &stItem.wItemID, &stItem.pszText, &stItem.bStatus, &bModified,
GETITEM_FIRST);
```

6.29.TUDLRemoveItem

int TUDLRemoveItem(AppIndex nAppIndex, BYTE bAppInstance, WORD wItemID);

Purpose:
    Retrieves an item from the packager after upload from the device

Parameters:
    nAppIndex - type of the item being removed from the database. Refer to Section 6.35 for the AppIndex definition
    bAppInstance - instance of the application
    wItemID – The identifier of the item (the same as the number when it was originall added)

Returns:
    0 - Success
    < 0 - Error
    > 0 - Warning

Example:

```
    BYTE bInstance = 0;
    TUDLRemoveItem(CONTACT_APP, bInstance, 3);          // remove the 3rd item from the 1st  contact instance
```

6.30.TUDLRemoveAllItems

int TUDLRemoveAllItems(AppIndex nAppIndex, BYTE bAppInstance);

Purpose:
    Removes all items from all applications

Parameters:

nAppIndex - type of the item being removed from the database. Refer to Section 6.35 for the AppIndex definition
bAppInstance - instance of the application

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
```
BYTE bInstance = 0;
TUDLRemoveAllItems(ALARM_APP, bInstance);              // remove all of the alarms
```

6.31.TUDLRemoveAllApps

int TUDLRemoveAllApps();

Purpose:
Removes all applications from the device

Parameters:
None

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
```
TUDLRemoveAllApps();
```

6.32.TUDLRemovePeriodicTask

int TUDLRemovePeriodicTask;

Purpose:
Removes the Periodic Task from the device

Parameters:
None

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:
```
TUDLRemovePeriodicTask();
```

6.33.TUDLAddApp

**int TUDLAddApp(AppIndex nAppIndex, int nAppInstance, BOOL boolPassword,
LPTSTR lpszModeName);**

Purpose:
Adds an application to the device. Also specifies whether a password is required for the mode, as well as specifying the mode name that is shown upon mode entry.
NOTE: When adding a chrono application to the device, you must specify the number of laps being download (or add the workouts manually) to see the chrono mode in the device.

Parameters:
nAppIndex - type of the item being added to the database. Refer to Section 6.35 for the AppIndex definition
bAppInstance - instance of the application

boolPassword - TRUE if the mode is password protected, FALSE otherwise
lpszModeName - pointer to a string containing the mode name. Any character to left of a newline character will be on the first line and any characters to the right of the newline character will be on the second line

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:

```
CString sText;
sText = “CHRONO”;
// Add the chrono as the first instance, setting password use to true, and supplying a mode banner as CHRONO
TUDLAddApp(CHRONO_APP, 0, TRUE, (LPTSTR)(LPCTSTR)sText);
```

6.34.TUDLAddAppExt

```
int TUDLAddAppExt(AppIndex nAppIndex, int nAppInstance,
                  BOOL boolPassword, LPTSTR lpszModeName,
                  LPBYTE lpbParam, BYTE bParamLen,
                  LPBYTE lpbCode, WORD wCodeLen,
                  LPBYTE lpbDatabase, WORD wDatabaseLen);
```

Purpose:
Adds a WristApp to the device. Also specifies whether a password is required for the mode, as well as specifying the mode name that is shown upon mode entry

Parameters:
nAppIndex - type of the item being added to the database. Refer to Section 6.35 for the AppIndex definition
bAppInstance - instance of the application
boolPassword - TRUE if the mode is password protected, FALSE otherwise
lpszModeName - pointer to a string containing the mode name. Any character to left of a newline character will be on the first line and any characters to the right of the newline character will be on the second line
lpbParam – pointer to an array of bytes containing the parameter file data
bParamLen – length of the parameter data pointed to lpbParam
lpbCode – pointer to an array of bytes containing the WristApp code
bCodeLen – length of the WristApp code pointed to lpbParam
lpbDatabase – pointer to an array of bytes containing the database if it exists
bParamLen – length of the database pointed to lpbParam. If the WristApp doesn’t have a database, this is 0

Returns:
0 - Success
< 0 - Error
> 0 - Warning

Example:

```
int nRet = TUDLAddAppExt(WRISTAPP_APP, pMain->m_nInstance, bUsePassword,
                         (LPSTR)(LPCTSTR)sText,  lpbParam, (BYTE)bParamSize,
                         lpbCode, (WORD)wCodeSize, lpbDB, (WORD)wDBSize);
```

6.35.TUDLAddPeriodicTask

```
int TUDLAddAppExt(WORD wCodeSize, LPBYTE lpbCode);
```

Purpose:
Adds a Periodic Task to the device. The Periodic Task code is designed to run either every second, every minute, every hour, or every day. Only one Periodic Task may be present in the device at any given time.

Parameters:
wCodeSize – length of the Periodic Task code pointed to lpbCode

lpbCode – pointer to an array of bytes containing the code

Returns:

0 - Success
< 0 - Error
> 0 - Warning

Example:

```
WORD wCodeSize;
BYTE *lpbCode;
lpbCode = new BYTE[1024];
wCodeSize = LoadPeriodicTask(&lpbCode);
int nRet = TUDLAddPeriodicTask(wCodeSize, lpbCode);
```

6.36.TUDLBeep

int TUDLBeep(WORD wFrequency, WORD wDuration);

Purpose:

To produce a sound (beep) through the PC speaker of a given frequency and duration

Parameters:

wFrequency - The frequency of the sound in Hz from 37 to 32767
wDuration - The duration of the sound in msec from 75 up

Returns:

TRUE (always)

Example:

```
TUDLBeep(2048, 500);  // sound a 2 KHz tone for ½ a second
TUDLBeep(4096, 500);  // sound a 4 KHz tone for ½ a second
```

6.37.Structure Definitions

Certain structures are used to collect information into a single variable for passing to the API functions. These structs are defined on the following pages.

TOD_ITEM

BYTE bItemID
    Zero-based time zone index, 0=primary time, must be unique for each time zone
LPTSTR pszText
    Pointer to a null-terminated string containing city code abbreviation
int cchTextMax
    Number of characters in the buffer pointed to by pszText. Must be <= 3
LONG lTimeOffset
    Time offset to the current system time (UTC) in seconds
TimeZoneIndex nTZIndex
    Index of the time zone
DateFormat nDateFormat
    Date format according to the date format enumeration. Must be between 0 and 2
TimeFormat nTimeFormat
    Time format according to the time format enumeration. Must be between 0 and 1
BOOL boolDSTObserved
    TRUE if the time zone observes daylight saving time
BOOL boolIsInDST
    TRUE if the the time zone is currently in daylight saving time
BOOL boolShowWeekNumber
    show week number in place of day of week
BYTE b1stDayOfWeek

0=Sun, 1=Mon
DWORD dwReserved
Reserved for future use

CHRONO_ITEM
WORD wItemID
Must be a unique index of the item to which this structure refers
WORD wWorkoutID
Must be a unique index of the workout to which this structure refers
BYTE bType
0=unused, 1=workout, 2=lap
Workout data
BYTE bNumLaps
Number of laps in the workout
DATE dtDateTime
The date and time of the workout
Lap data
BYTE bLapNum
Lap number
BYTE bHour
Lap hour
BYTE bMinute
Lap minute
BYTE bSecond
Lap second
BYTE bHundredth
Lap hundredth of a second

TIMER_ITEM
WORD wItemID
Must be a unique index of the item to which this structure refers
LPTSTR pszText
Pointer to a null-terminated string that contains the item text, 9 chars max
int cchTextMax
Number of characters in the buffer pointed to by pszText
BYTE bHour
Timer hour 0-23
BYTE bMinute
Timer minute 0-59
BYTE bSecond
Timer second 0-59
TimerAction nAction
Action at end of timer. Refer to the TimerAction enum in section 4.34
BYTE bNumOfReps
Number of repetitions
BYTE bOptions
Timer options: 0=half-way reminder not set, 1=reminder set

SCHEDULE_ITEM
WORD wGroupID
Must be a unique index of the group to which this structure refers
LPTSTR pszGroupText
Ppointer to a null-terminated string that contains the group text
int cchGroupTextMax
Number of characters in the buffer pointed to by pszGroupText
LPTSTR pszGroupLabel

  - Pointer to a null-terminated string that contains the group label
- int cchGroupLabelMax
  - Number of characters in the buffer pointed to by pszGroupLabel
- ScheduleType nType
  - Schedule type according to the enum. Refer to section 4.34
- WORD wItemID
  - Must be a unique index of the item to which this structure refers
- LPTSTR pszText
  - Pointer to a null-terminated string that contains the item text
- int cchTextMax
  - Number of characters in the buffer pointed to by pszText
- DATE dtDateTime
  - The date and time of the schedule item
- int nDayOfWeek
  - Day of the week for schedule type DOW_TIME; 0=sun, 1=mon,... 6=sat

ALARM_ITEM

- WORD wItemID
  - Must be a unique index of the item to which this structure refers
- LPTSTR pszText
  - Pointer to a null-terminated string that contains the item text
- int cchTextMax
  - Number of characters in the buffer pointed to by pszText
- AlarmStatus nStatus
  - Alarm status: 0=unused, 1=used/disarmed, 3=used/armed. Refer to the enum in section 4.34
- AlarmFrequency nFreq
  - Frequency (or occurence) of the alarm. Refer to the enum in section 4.34
- BYTE bHour
  - Alarm hour
- BYTE bMinute
  - Alarm minute
- DATE dtReserved1
  - Not used

APPT_ITEM

- WORD wItemID
  - Must be a unique index of the item to which this structure refers
- LPTSTR pszText
  - Pointer to a null-terminated string that contains the item text
- int cchTextMax
  - Number of characters in the buffer pointed to by pszText
- DATE dtStartDateTime
  - The date and time of the first occurrence of the appt
- ApptStatus nStatus
  - Appointment status: 0=unused, 1=disarmed, 3=armed. Refer to the enum in section 4.34
- Frequency nFreq
  - Frequency (or occurence) of the appt. Refer to the enum in section 4.34
- Reminder nReminder
  - Reminder offset (used to calculate prenotification time). Refer to the enum in section 4.34
- DATE dtReserved1
  - Not used
- DATE dtReserved2
  - Not used

OCC_ITEM

- WORD wItemID

Must be a unique index of the item to which this structure refers
LPTSTR pszText
Pointer to a null-terminated string that contains the item text
int cchTextMax
Number of characters in the buffer pointed to by pszText
DATE dtStartDateTime
The date and time of the original occurrence of the occasion
BYTE bStatus
occasion status: 0=non-recurring, 1=recurring
OccasionType nType
Type of the occasion. Refer to the enum in section 4.34
DATE dtReserved1
Not used

PHONE_ITEM

WORD wItemID
Must be a unique index of the item to which this structure refers
LPTSTR pszNameText
Pointer to a null-terminated string that contains the item text
int cchNameTextMax
Number of characters in the buffer pointed to by pszText
LPTSTR pszTypeText
Pointer to a null-terminated string that contains the item text, 2 chars max
int cchTypeTextMax
Number of characters in the buffer pointed to by pszText
LPTSTR pszPhoneText
Pointer to a null-terminated string that contains the item text
int cchPhoneTextMax
Number of characters in the buffer pointed to by pszText
BYTE bOptions
bit 0 = don't apply phone number formatting
bit 1 = display left arrow in the first pos of the main dot matrix
bit 2 = display right arrow at the end of the phone number
bit 3 = display left arrow in the top dot matrix (in place of phone type)

NOTE_ITEM

WORD wItemID
Must be a unique index of the item to which this structure refers
LPTSTR pszText
Pointer to a null-terminated string that contains the item text
int cchTextMax
Number of characters in the buffer pointed to by pszText
BYTE bStatus
Note status: 0=unused, 1=used

OPTION_ITEM

BOOL boolNightModeEnabled
TRUE if nightmode enabled
BOOL boolNightModeAuto
TRUE if nightmode auto on/off enabled
BYTE bNightModeOnMin
The time (in minutes) night mode will be turned on on the device
BYTE bNightModeOnHour
The time (in hours) night mode will be turned on on the device
BYTE bNightModeOffMin
The time (in minutes) night mode will be turned off on the device

BYTE bNightModeOffHour
    The time (in hours) night mode will be turned off on the device
BYTE bNightModeToggleDuration
    For how long the user needs to press the indiglo button to turn on the nightmode; min=4, max=30
BOOL boolChimeEnabled
    TRUE if hourly chime enabled
BOOL boolChimeAuto
    TRUE if hourly chime auto on/off enabled
BYTE bChimeOnHour
    The time (in hours) chime will be turned on on the device
BYTE bChimeOffHour
    The time (in hours) chime will be turned off on the device
BOOL boolButtonBeepEnabled
    TRUE if button beep enabled
BYTE bLastSetCharacter
    Index to the last editable character during setting. Typically 68(~) for US and 86 (overscore) for international
LPTSTR pszPassword
    Pointer to a null-terminated string that contains the password
int cchPasswordMax
    Number of characters in the buffer pointed to by pszPassword
DWORD dwReserved
    Not used

SOUND_ITEM
SoundType nItemID
    Must be a unique index of the item to which this structure refers
int nSoundsMax
    Number of bytes in the buffer pointed to by lpbSounds
LPBYTE lpbSounds
    Pointer to an array of sounds - up to 16 bytes long

6.38.Enum Definitions

The AppIndex enum defines the application types that are used when performing certain API calls. For example, when calling TUDLAddItem, if you wanted to add an Appointment item, the APPT_APP parameter would be specified in the TUDLAddItem call. All of the AppIndex values are shown below:

AppIndex	Value
TOD_APP	0
COMM_APP	1
CHRONO_APP	2
TIMER_APP	3
INTTIMER_APP	4
ALARM_APP	5
APPT_APP	6
NOTE_APP	7
OPTION_APP	8
OCCASION_APP	9
CONTACT_APP	10
SCHEDULE_APP	11
SYNCHRO_APP	12
COUNTER_APP	13
SOUND_APP	16
WRISTAPP_APP	17

The DateFormat enum defines the values that can set in the TOD_ITEM structure for the format of the date.

DateFormat	Value	Example
DMY	0	dd.mm.yy
YMD	1	yy-mm-dd
MDY	2	mm-dd-yy

The TimeFormat enum defines the values that can set in the TOD_ITEM structure for the format of the time.

TimeFormat	Value	Example
HOUR12	0	am/pm time
HOUR24	1	24 hour time

The ChronoFormat enum defines the chrono format that can be set using TUDLSetOption.

ChronoFormat	Value	Example
CHRONOFORMAT_LAP_SPLIT	0	Lap on top line Split on bottom line
CHRONOFORMAT_SPLIT_LAP	1	Split on top line Lap on bottom line
CHRONOFORMAT_TIME_SPLIT	2	Time of day on top line Split on bottom line
CHRONOFORMAT_TIME_LAP	3	Time of day on top line Lap on bottom line

The TimerFormat enum defines the values that can be set in the TIMER_ITEM structure for the function at the end of the timer.

TimerFormat	Value	Example
TIMERACTION_STOP	0	Stop at the end of the timer
TIMERACTION_REPEAT	1	Repeat at the end of the timer
TIMERACTION_CHRONO	2	Start the chrono at the end of the timer

The ScheduleType enum defines the values that can be set in the SCHEDULE_ITEM structure for the type of schedule.

ScheduleType	Value	Example
SCHEDULE_DATE	1	Set the schedule format to date only
SCHEDULE_DOW_TIME	2	Set the schedule type to day of week and time only
SCHEDULE_DATETIME	3	Set the schedule type to date and time

The AlarmStatus enum defines the values that can be set in the ALARM_ITEM structure to arm or disarm the alarm, as well as making the alarm unused.

AlarmStatus	**Value**	**Example**
ALM_UNUSED	0	Set an alarm to unused
ALM_DISARMED	1	Set an alarm to disarmed
ALM_ARMED	2	Set an alarm to armed

The AlarmFrequency enum defines the values that can be set in the ALARM_ITEM structure for the format of the time.

AlarmFrequency	**Value**	**Example**
ALMFREQ_DAILY	1	Set an alarm to go off every day
ALMFREQ_WEEKDAY	2	Set an alarm to go off on weekdays only
ALMFREQ_WEEKEND	3	Set an alarm to go off on weekends only
ALMFREQ_WEEKLY_SU	4	Set an alarm to go off on Sundays only
ALMFREQ_WEEKLY_MO	5	Set an alarm to go off on Mondays only
ALMFREQ_WEEKLY_TU	6	Set an alarm to go off on Tuesdays only
ALMFREQ_WEEKLY_WE	7	Set an alarm to go off on Wednesdays only
ALMFREQ_WEEKLY_TH	8	Set an alarm to go off on Thursdays only
ALMFREQ_WEEKLY_FR	9	Set an alarm to go off on Fridays only
ALMFREQ_WEEKLY_SA	10	Set an alarm to go off on Saturdays only

The ApptStatus enum defines the values that can be set in the APPT_ITEM structure to arm or disarm the appointment, as well as making the appointment unused.

ApptStatus	**Value**	**Example**
APPT_UNUSED	0	Set an appointment to unused
APPT_DISARMED	1	Set an appointment to disarmed
APPT_ARMED	2	Set an appointment to armed

The Frequency enum defines the values that can be set in the APPT_ITEM structure for which days an appointment alert will be generated.

Frequency	**Value**	**Example**
APPTFREQ_1DAY	0	Set an appointment to go off once
APPTFREQ_DAILY	1	Set an appointment to go off every day
APPTFREQ_WEEKDAY	2	Set an appointment to go off on weekdays only
APPTFREQ_WEEKEND	3	Set an appointment to go off on weekends only
APPTFREQ_WEEKLY	11	Set an appointment to go off every day for the selected week
APPTFREQ_MONTHLY	12	Set an appointment to go off every

		day for the selected month
APPTFREQ_YEARLY	13	Set an appointment to go off the same day every year

The Reminder enum defines the values that can be set in the APPT_ITEM structure to set the prenotification time for an appointment.

Reminder	**Value**	**Example**
REMINDER_0MINS	0	Set the prenotification time to 0 minutes before the appointment
REMINDER_5MINS	1	Set the prenotification time to 5 minutes before the appointment
REMINDER_10MINS	2	Set the prenotification time to 10 minutes before the appointment
REMINDER_15MINS	3	Set the prenotification time to 15 minutes before the appointment
REMINDER_30MINS	4	Set the prenotification time to 30 minutes before the appointment
REMINDER_60MINS	5	Set the prenotification time to 1 hour before the appointment
REMINDER_2HRS	6	Set the prenotification time to 2 hours efore the appointment
REMINDER_3HRS	7	Set the prenotification time to 3 hours efore the appointment
REMINDER_4HRS	8	Set the prenotification time to 4 hours before the appointment
REMINDER_5HRS	9	Set the prenotification time to 5 hours before the appointment
REMINDER_6HRS	10	Set the prenotification time to 6 hours before the appointment
REMINDER_8HRS	11	Set the prenotification time to 8 hours before the appointment
REMINDER_10HRS	12	Set the prenotification time to 10 hours before the appointment
REMINDER_12HRS	13	Set the prenotification time to 12 hours before the appointment
REMINDER_24HRS	14	Set the prenotification time to 24 hours before the appointment
REMINDER_48HRS	15	Set the prenotification time to 48 hours before the appointment

The OccasionType enum defines the values that can be set in the OCCASION_ITEM structure to set the type of occasion such as a birthday or anniversary.

OccasionType	**Value**	**Example**
OCCASION_NONE	0	Set the occasion to have no special type
OCCASION_BDAY	1	Set the occasion to have the birthday type
OCCASION_ANNIV	2	Set the occasion to have the anniversary type
OCCASION_HOLIDAY	3	Set the occasion to have the holiday type
OCCASION_VACATION	4	Set the occasion to have the vacation type

The SoundType enum defines the indexes into the sound structure to be used when setting values in the SOUND_ITEM structure for creating sounds for the different events.

SoundType	Value	Example
SOUND_BUTTON_BEEP	0	Button beep
SOUND_HOURLY_CHIME	1	Hourly chime
SOUND_ALARM	2	Alarm
SOUND_APPOINTMENT	3	Appointment
SOUND_TIMER	4	Timer
SOUND_INT_TIMER	5	Interval Timer
SOUND_HALF_TIMER	6	Halfway Alert
SOUND_COMM_ERROR	7	Communications Error
SOUND_CUSTOM	8	Custom

The GetItemType enum defines the values that can be used when getting modified data for a certain mode.

GetItemType	Value	Example
GETITEM_FIRST	0	Get the first item in the database
GETITEM_NEXT	1	Call this to get each subsequent item in the database

7. VISUAL BASIC NOTES

Visual Basic does not support structures with variable length strings. In light of this, there are special AddItem and GetItem functions added for Visual Basic. Instead of TUDLAddItem or TUDLGetItem, the individual functions (such as TUDLGetChronoItem or TUDAddAlarmItem) should be used instead.

Please refer to TUCP.BAS for the Visual Basic definitions of the API functions

8. RETURN CODES

Most API functions return a value. These values can be warnings or errors. Errors are all fatal. Communication can not continue when one of these errors occur. In most cases, when a warning is received, communication can still continue, however, typically something is wrong with the data such as too many records have been added, so some records won’t be added.

If an API function returns a value, then the value will be one of the ones specified below:

NAME	VALUE	DESCRIPTION
TIMEX_SUCCESS	0	No error
TIMEX_ERR_TOO_MUCH_DATA	3	Attempt to add too much data
TIMEX_ERR_NO_MEM	4	Attempt to allocate memory failed. Close other apps and try again.
TIMEX_ERR_FILE_ERR	7	Error creating temporary file
TIMEX_ERR_INV_PACKET_LEN	8	Invalid packet length when creating USB packet
TIMEX_ERR_INV_PARM	9	API parameter was invalid
TIMEX _ERR_DEV_NOT_FOUND	10	Device was not found (make sure it is plugged in)
TIMEX_ERR_COMM	11	Generic communication error occurred
TIMEX_ERR_COMM_CHECKSUM	12	Invalid checksum returned from watch
TIMEX_ERR_COMM_PACKET_LEN	13	Invalid packet length returned from watch

TIMEX_ERR_COMM_DEV_INFO	14	Could not retrieve device information block
TIMEX_ERR_COMM_DEV_MISMATCH	15	Incompatible device attached
TIMEX_ERR_COMM_TIMEOUT	16	Timed out after attemping to send packet multiple times
TIMEX_ERR_NO_ITEM	18	No entries are present for the selected application and/or item type
TIMEX_ERR_DEV_API_MISMATCH	27	Unsupported operation for the selected device (API does not exist) This error occurs when the selected devices do not support programs, melodies, system data or international date formats but an attempt is made to set this info; determining device support is as follows: Support for downloadable programs is determined by using the TUDLGetOption API with the DL_DEV_SUP_PROGRAM identifier. Support for downloadable melodies is determined by using the TUDLGetOption API with the DL_DEV_SUP_MELODY identifier. Support for system data is determined by using the TUDLGetOption API with the DL_DEV_SUP_SYSTEM_DATA identifier. Support for international date formats is determined by using the TUDLGetOption API with the DL_DEV_SUP_INTL_DATE_FORMATS identifier.
TIMEX_ERR_PROG_EXISTS	30	A program with the same name and instance was attempted to be added
TIMEX_ERR_ITEM_EXISTS	32	An item with the same name and ID was attempted to be added
TIMEX_ERR_BAD_CHAR_IN_PROG	33	A character was found that wasn't 0-9, A-F, or a-f
TIMEX_ERR_INVALID_DEVICE	34	Device types tested to be invalid
TIMEX_ERR_PROGRAM_TOO_LARGE	46	An attempt was made to add a program larger than the reciever's downloadable program region
TIMEX_ERR_MELODY_TOO_LARGE	47	An attempt was made to add a melody larger than the reciever's downloadable melody region
TIMEX_ERR_TOO_MUCH_BINARY_DATA	48	An attempt was made to add binary data larger than the reciever's downloadable binary data region
TIMEX_ERR_COMM_LOST_SUSPEND_MODE	49	The PC went into suspend mode, so communication was terminated.
TIMEX_WRN_OVER_MAXDATA	-1	Added more data than the max data parameter allows
TIMEX_WRN_TEXT_TRUNCATED	-2	Text string was truncated
TIMEX_WRN_CHAR_LOST	-3	Untranslateable character
TIMEX_WRN_NO_DATA	-5	No data to transmit
TIMEX_WRN_DEFAULT_DEV_SET	-6	Error while processing INI file so the device type was set to the default
TIMEX_WRN_USER_ABORT	-8	User aborted send

TIMEX_WRN_TOO_MANY_RECORDS	-9	Number of records added has exceeded the amount supported by the device
TIMEX_WRN_NUMBER_ADJUSTED	-10	Number has been adjusted to the closest amount supported by the device

9. OPTION VALUES

The option values can be set using the TUDLSetOption API call and retrieved by using the TUDLGetOption API call.

NAME	VALUE	DESCRIPTION
DL_OPT_DEV_APPT_LEN	1	Returns max allowable length of appointment messages for the selected device. This is typically 100 characters.
DL_OPT_DEV_NOTE_LEN	2	Returns max allowable length of note item messages for the selected receiver / device.
DL_OPT_DEV_PHONE_LEN	3	Returns max allowable length of phone messages (name) for the selected receiver / device.
DL_OPT_DEV_PHONE_NUM_LEN	4	Returns max allowable length of phone book numbers for the selected receiver / device. This is typically 12 characters.
DL_OPT_DEV_OCC_LEN	5	Returns max allowable length of occasion messages for the selected receiver / device.
DL_OPT_DEV_ALARM_LEN	6	Returns max allowable length of alarm messages for the selected receiver / device.
DL_OPT_DEV_CITY_LEN	7	Returns max allowable length of city code messages for the selected receiver / device. This is typically 3 characters. "PST" is an example of an alarm message.
DL_OPT_DEV_MAX_PACKET_LEN	8	Returns max allowable packet length for the selected receiver / device. This ID is for internal use only, and should not be used by application developers.
DL_OPT_DEV_TOD_MAX	9	Returns max number of Time Of Day zones supported by the selected device
DL_OPT_SYS_DEVICE	10	Returns the currently-selected device type. For example, DL_DEV_851. Use the SetDeviceType() API to set a new device type.
DL_OPT_DEV_MAX_DATABASE	12	Returns the max size of database information for the selected receiver / device. The database consists of all application entries. The database is a single block in most devices; this means that these info types are combined into a single structure, which is downloaded as a unit. It is therefore impossible to update a device's phonebook yet leave the appointments unaffected. A PIM / front-end app can typically accept more information than can be stored in a device. This ID is therefore used to control the amount of database data to be downloaded. This is accomplished by adding all the database, request the max allowable database size using this ID, then request the

		current total database size using the DL_OPT_DEV_TOTAL_DATA identifier (described next). A warning should be displayed once the total exceeds the max, and actual download must be inhibited until the total is less than or equal to the max allowable.
DL_OPT_DEV_TOTAL_DATA	13	Returns the current size of the database information structure for the selected receiver / device. This value may exceed the maximum allowable database size for the selected device. It is therefore required that this value be compared to the value returned using the DL_OPT_DEV_MAX_DATABASE identified (described above). The actual download must be inhibited until the current database structure size is less or equal to the max allowable.
DL_OPT_DEV_TOTAL_USER_MEMORY	14	Returns a value specifying percentage of the memory used by currently selected data for the current device. This parameter can be used in place of the previous two to show the percentage of memory used.
DL_OPT_DEV_MAX_APPS	15	Returns a value specifying the maximum number of applications (modes). Time of day mode and communication mode are excluded from this value.
DL_OPT_DEV_MAX_PROGRAM	27	Returns the max size of downloadable program area for the selected receiver / device. Downloadable programs are small programs which can be downloaded to the device using Timex Data Link. The value returned is the memory size allocated by the device to store the downloadable program (in bytes). Use the DL_DEV_SUP_PROGRAM identifier with TUDLGetOption to determine if the selected device supports downloadable programs.
DL_OPT_DEV_MAX_MELODY	28	Returns the max size of downloadable melody area for the selected receiver / device. Downloadable melodies are small sound-scapes which can be downloaded to the device using Timex Data Link.

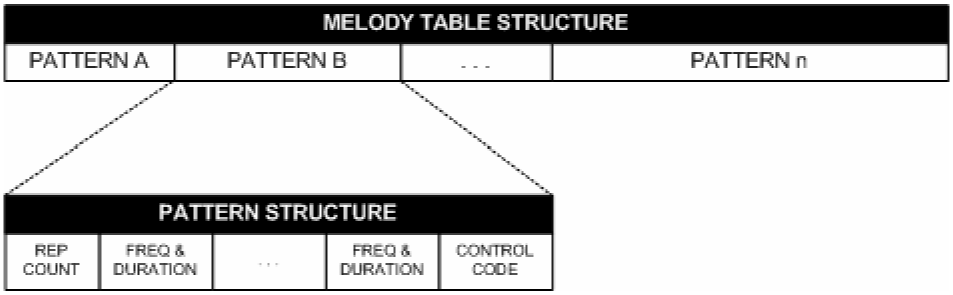
		The value returned is the memory size allocated by the device to store the downloadable melody (in bytes). Use the DL_DEV_SUP_MELODY identifier with TUDLGetOption to determine if the selected device supports downloadable melody info.
DL_OPT_DEV_TIMER_MAX	29	Returns max number of timer entries supported by the selected receiver
DL_OPT_DEV_INTTIMER_MAX	30	Returns max number of interval timer entries supported by the selected receiver
DL_OPT_DEV_SCHEDULE_MAX	31	Returns max number of schedule entries supported by the selected receiver
DL_OPT_DEV_SCHEDULE_GROUP_MAX	32	Returns max number of schedule group entries supported by the selected receiver
DL_OPT_DEV_SCHEDULE_GROUP_NAME_LEN	33	Returns max length of the group name. This is a label for the schedule mode set position such as "LOCATION" or "TEAM"
DL_OPT_DEV_SCHEDULE_NUM	34	Returns current number of schedule entries. The dwExtra parameter should contain the application's instance number
DL_OPT_DEV_INTTIMER_NUM	35	Returns current number of interval timers. The dwExtra parameter should contain the application's instance number
DL_OPT_DEV_TIMER_NUM	36	Returns current number of timers. The dwExtra parameter should contain the application's instance number
DL_OPT_DEV_ALARM_NUM	37	Returns current number of alarms. The dwExtra parameter should contain the application's instance number
DL_OPT_SYS_ERR_CODE	38	Returns the specific error code for the most recent API call. With few exceptions, all API functions return a value of 0 for success, a value less than 0 for an error and a value greater than 0 for a warning. Most often, it is sufficient to simply test the return value and if an error then cancel current operation and shut-down, and if a warning then just continue on knowing that a non-critical error occurred.
DL_OPT_DEV_SUP_PROGRAM	45	Returns a boolean value indicating the support for downloadable programs for the selected receiver / device. Downloadable programs are small programs which can be downloaded to the device using Timex Data Link. Because front-end support of downloadable programs is under development, please contact Timex to obtain more technical detail than can be included here.
DL_OPT_DEV_SUP_MELODY	46	Returns a boolean value indicating the support for downloadable melody information

		for the selected receiver / device. Downloadable melodies are small sound-scapes which can be downloaded to the device using Timex Data Link. Because front-end support of downloadable melodies is under development, please contact Timex to obtain more technical detail than can be included here.
DL_OPT_DEV_SUP_SYSTEM_DATA	47	Returns a boolean value indicating the support for downloadable system information for the selected receiver / device. System information consists of settings such as hourly chime control and keypress beeps on the device.
DL_OPT_DEV_SUP_INTL_DATE_FORMAT	48	Returns a boolean value indicating the support for international date formats for the selected receiver / device. International date formats allow the device to display date information in MMDDYY, YYMMDD and DDMMYY ordering, using "." or "-" characters as seperators.
DL_OPT_SYS_NUM_DEVICES	49	Returns the number of device types the system is aware of.
DL_OPT_DEV_APPT_NUM	50	Returns current number of appointments
DL_OPT_DEV_NOTE_NUM	51	Returns current number of note entries
DL_OPT_DEV_PHONE_NUM	52	Returns current number of phone numbers
DL_OPT_DEV_OCC_NUM	53	Returns current number of anniversaries
DL_OPT_DEV_ALARM_MAX	54	Returns max number of alarms supported by the device
DL_OPT_DEV_APPT_MAX	55	Returns max number of appointments supported by the selected receiver / device.
DL_OPT_DEV_NOTE_MAX	56	Returns max number of note entries supported by the selected receiver / device.
DL_OPT_DEV_PHONE_MAX	57	Returns max number of phone numbers supported by the selected receiver / device.
DL_OPT_DEV_OCC_MAX	58	Returns max number of occasions supported by the selected receiver / device.
DL_OPT_DEV_TIMER_LEN	60	Returns max allowable length of timer messages for the selected receiver / device. This is typically 8 characters. Boil Egg" is an example of an timer message.
DL_OPT_DEV_CHRONO_NAME_LEN	61	Returns max allowable length of chrono name
DL_OPT_DEV_CHRONO_NUM	62	Returns or sets current number of chrono laps.
DL_OPT_DEV_CHRONO_MAX	63	Returns max allowable number of chrono laps.
DL_OPT_DEV_SUP_CHRONO	64	Returns a boolean value indicating the support for configurable/controllable chronographs for the selected receiver / device.

		Controllable chronos allow a user to configure the chrono embedded in the receiver. Options such as number of laps of storage are set.
DL_OPT_DEV_SUP_TIMER	65	Returns a boolean value indicating the support for configurable/controllable timers for the selected receiver / device. Controllable timers allow a user to configure the timer embedded in the receiver. Options such as timeout duration are set.
DL_OPT_DEV_SUP_BINARY_DATA	66	Returns a boolean value indicating the support for downloadable binary data to the selected receiver / device. Downloadable binary data is device-specific information formatted according to a Timex download specification and a receiver-specific data format specification.
DL_OPT_DEV_MAX_BINARY_DATA	67	Returns the max size of binary data area for the selected receiver / device. Downloadable binary data is device-specific information formatted according to a Timex download specification and a receiver-specific data format specification. The value returned is the memory size allocated by the device to store the binary data (in bytes). Use the DL_DEV_SUP_BINARY_DATA identifier with TUDLGetOption to determine if the selected device supports binary data downloads.
DL_OPT_DEV_SUP_DATABASE	68	Returns a boolean value indicating the support for any or all the database types (appointments, anniversaries, phone books and to-do lists) for the selected receiver / device. Databases allow storage of the 4 information types outlined above. Most devices will support all the database types, but some will only support a subset of the database types.
DL_OPT_DEV_CHRONO_MIN	71	Returns min allowable number of chrono laps.
DL_OPT_DEV_CHRONO_FORMAT	72	Sets or returns chrono display format which can be one of the ChronoFormat enum types
DL_OPT_DEV_SUP_NIGHTMODE_OPT	73	Returns a boolean value indicating the support for setting Night-Mode options.

10. CREATING SOUNDS

Using the API calls, one can create the sound database by setting up a melody table for each sound event. A melody table consists of one or more melody patterns. A melody pattern contains a repetition count, frequency and duration codes, and a control code. The control code is used to indicate the end of a melody pattern. For a melody pattern that is repeated more than once, the control code signals the audio driver to repeat the melody pattern. When the repetition count of the melody pattern is complete, the control code indicates whether it is the end of the melody table or that another melody pattern exists. The maximum size for each pattern is 255 bytes.



REP COUNT

Indicates number of repetitions for current pattern. Maximum value is 255.

FREQ & DURATION

A byte value indicating the frequency and duration. The frequency data is stored in the upper nibble. The duration data is stored in the lower nibble.

The available frequency codes are:

(0 Hz)	**0**
(1170 Hz)	**1**
(1365 Hz)	**2**
(1638 Hz)	**3**
(2048 Hz)	**4**
(2340 Hz)	**5**
(2731 Hz)	**6**
(3277 Hz)	**7**
(4096 Hz)	**8**

The Duration is a value from 0 to 15. The actual time duration is computed using the formula:

$$\frac{(Duration+1)}{32}$$

Maximum Time: 0.5 seconds
Minimum Time: 21.35 milliseconds

CONTROL CODE

Indicates an end of a current melody pattern. Depending on the control code being used, it signifies that another melody pattern exists or this is the last pattern.

The available control codes are:

0xFE Continue pattern
0xFF End pattern

A sample melody pattern used to generate the Timex Step Tone (alarm tone). This is just one event, and more buffers can be created for the other events. There is a 32 byte length limit per event. The length of the buffer below is 11 bytes.

Buffer Index	Contents	Description
0	10	Number of Repititions
1	0x43	Frequency and Duration
2	0x03	Frequency and Duration
3	0x43	Frequency and Duration
4	0x0F	Frequency and Duration
5	0x03	Frequency and Duration
6	0xFE	Continue Pattern
7	40	Number of Repititions
8	0x43	Frequency and Duration
9	0x03	Frequency and Duration
10	0xFF	End Melody Pattern

Sounds have already been defined with the Timex PIM using the file format specified below. You may opt to use this file yourself, and convert the entries to the above sound array.

```
; EASY.SF
; The sound file is defined with sections that represent events.
; Each Event is enclosed within brackets [ ]. The exception is the description
; field which contains one entry called name that is the text to display in the
; sound selected dialog

; The event contains a repeat count that determines how many
; times the sequence of sounds will repeat for an event. Each
; event also contains individual sounds. Each sound is defined
; as SoundX where X is a number starting with 1. The sound is
; defined as a frequency,duration. The frequency ranges available
; for the Epson S1C88349 ICU are the following:
;
;     0 - for no sound
;   1170
;   1365
;   1638
;   2048
;   2340
;   2731
;   3277
;   4096
;
; The duration field is the number of milliseconds that the sound will sound.
; So, if an entry is found with the following:
;
; Sound2=4096,100
;
; The above sound would sound a 4 Khz tone for approximately 100 ms. Please note that the watch
; is only capable of handling durations that are a multiple of 31.25 ms. So, whatever you specify
; for a duration wile be rounded down to the closest mutliple of 31.25 ms.
;
; Sound1=-1,40
;
```

```
; Each sound entry that has a -1 will not be treated as a sound. Instead, the second entry is the number
; of times to repeat the series of sounds until the next -1 is found, or the end of the sound event
; has been found. The above description shows that the 1st series of sounds would repeat 40 times
;
; Any event that is not accounted for in this file will inherit the default
; tone for the event.
;
; Sounds can be composed of multiple patterns. To start a new pattern, the
; sound entry should be a -1 followed by the repeat count for the next pattern.
; So, the entry for the end of pattern would be:

[Description]
Name=Easy Does It

[Button Beep]
Sound1=-1,1
Sound2=4096,32

[Halfway Timer]
Sound1=-1,1
Sound2=2730,32

[Hourly Chime]
Sound1=-1,1
Sound2=2340,125
Sound3=0,125
Sound4=2340,125

[Alarm]
Sound1=-1,10
Sound2=2048,125
Sound3=1638,63
Sound4=2048,125
Sound5=0,500
Sound6=0,125

[Appointment]
Sound1=-1,10
Sound2=2048,125
Sound3=0,125
Sound4=2048,125
Sound5=0,500
Sound6=0,125

[Timer]
Sound1=-1,40
Sound2=1638,125
Sound3=0,125

[Interval Timer]
Sound1=-1,2
Sound2=1638,125
Sound3=0,125
Sound4=1638,125
Sound5=0,125
```

```
Sound6=1638,375
Sound7=0,125

[Comm Error]
Sound1=-1,3
Sound1=4096,250
Sound2=2730,250
Sound3=2048,250
Sound4=1170,250
```