# Hot Smart Watch SDK Technical Specification

## Draft v0.01

# Table of Contents

## Contents

# *1. Introduction*

The Hot Smart Watch SDK allows application developers to create watch native apps as well as connected smartphone apps to utilise the power of HOT Smart Watch and create watchfaces and applications.

## 1.1 Audience

The audience of these documents are developer community who are interested in developing Smart Watch applications in different platforms. The initial release of SDK will allow Android Developers to develop Phone application that communicate directly with the watch. A separate Native SDK will allows developers skilled in C language to develop application running directly on the watch. An iPhone SDK will be released soon to allow iPhone developer to access the power of HOT Smart Watch in their applications.

# *2. SDK Details*

There are three components in the SDK which are meant for different group of application developers. These components are described below.

## 2.1 Android and iOS Development Platforms

The first two components of the SDK, Android and iOS development platforms are for smart phone application developers. These SDK platforms allow Android and iOS apps to use watch as a peripheral device. These SDK platforms are implemented in such a way that any Android or iOS developer can develop applications that utilises the power of smart watch without much learning curve.

## 2.2 Native Watch Apps

The smart watch Native SDK allows programmers to develop watch apps that reside in watch memory. These watch apps can be a standalone application or an extension of phone app which may work in conjunction with corresponding phone app. Developing native watch app requires good knowledge in C language and GCC compiler. Effort has been put to insulate programmers from the hardware details and to make it fairly simple. A reasonably good C language programmer can build standalone watch apps in Native SDK.

# *3. Android Development Platform*

This section describes the API functions available on android platform. This is s similar to the API available on iOS platform that will be released in the next version. Android Platform SDK is more feature rich compared to iOS API. This will reflect in the HOT Smart Watch SDK as well.

### *HOT Smart Watch SDK Context*

The implementation of android SDK allows multiple Phone application to communicate simultaneously with the Smart Watch. In order to provide this feature, SDK creates a virtual environment, which is further referenced in the document as “SDK Context”. SDK context is also responsible for keeping track of all installed applications which use HOT Smart Watch SDK. Applications using the SDK must register themself to SDK Context by calling “Register” function.

Android application which use the SDK services are divided in to two different categories. They are detailed below.

### *HOT Smart Watch Resident Apps*

These applications reserve a screen space on the watch that user can easily browse to. The watch supports up to 10 custom resident apps. This space is shared between Native apps and Phone Resident apps. The Phone Resident apps have following properties

- A screen space is reserved for the application.
- User can easily scroll to these application screens. It can use cache in the watch to keep icons bitmaps etc.
- Maximum 10 applications are possible.

While registering a resident application with SDK Context the application is given a screen slot. If preferred screen slot is not available, the SDK gives choice of free available slots. If there are no available slots, then function launches a UI to free a screen slot by removing previously installed applications.

### *HOT Smart Watch Gust Apps*

These guest apps do not have a permanent screen space in the watch. These apps use a Message box type pop up screens to display information and get input from the user. The app constructs and downloads this screen to be displayed at requested time on the watch. The app can also send a full screen data to be displayed temporarily on the watch. Since this type of app does not reserve a screen space, it is not accessible to the user via screen browsing.

- Screen space not reserved. Uses notification type pop up screen to display its screen
- Screen usage and display is triggered by the android application.
- There is no limit on the number of applications, but limited by available watch memory

## 3.1 Device Management Functions

### *HOTSmartWatch Class*

This is the root class implementing HOT Smart Watch SDK connection and interaction with SDK Context.

The functions in this class allow an app to establish itself in the SDK context, obtain a (virtual) connection with a smart watch and execute SDK functions. The current version of SDK allows

multiple host apps to connect to the watch simultaneously and use its services. An application has to register itself in the SDK context when it is executing for the first time.

### *Boolean RegisterApp(Context context, String applicationName, AppType type,int preferedScreePosition )*

**Description**

This function registers a Smart Watch application with the SDK context. SDK keeps track of all Smart Watch Apps installed in the Mobile.

**Parameters**

| | |
|---|---|
| Context | Android application context. Andoid application context can be obtained by `getApplicationContext` |
| applicationName | Unique Name of the application. SDK Keeps a database of applications installed. The name is maximum 12 characters alphanumeric string. This name has to be registered and reserved from HOT Smart Watch website. This is the name displayed in the applications title bar on the watch screen. |
| type | Application type. (Ref App Type listed below) |
| preferedScreenPos | **Analog Watch Face**<br>Five screen positions are reserved for analog watch-faces on the first column. Out of these reserved positions 3-5 is available for customs analog watch faces.<br>**Digital Watch Face**<br>Five positions are reserved for the digital watch-faces in the second column. Out of these 5 locations 3-5 is available for Custom watch faces.<br>**Android App**<br>Ten slots are dedicated for custom apps .Out of these ten slots, first five are dedicated for Native apps. The remaining positions (6-10) are available for Phone apps.<br><br>If the requested "preferred position" is not available, an option is given to the user to select alternate position with a pop-up screen |

**HOT Smart Watch App Types**

**(See SDK context description given above)**

| | |
|---|---|
| APP_ANALOG_WATCHFACE | Analog Watch Face application. Andoid |

| | |
|---|---|
| | application context can be obtained by `getApplicationContext` |
| APP_DIGITAL_WATCHFACE | Digital Watch Face application |
| APP_GUEST | GUEST type HOT Smart Watch android application .They will not reserve a space in the screen. But uses a popup screen. Useful for android service type apps to notify user with some info. |
| APP_RESIDENT | RESIDENT Type HOT Smart Watch application. This application reserves a screen space. When user enters this screen, corresponding application in the phone is notified. |
| APP_CANVAS_GUEST | Similar to APP_GUEST, but uses SDKs Canvas for easier UI creation |
| APP_CANVAS_RESIDENT | Similar to APP_RESIDENT, but uses SDKs Canvas for easier UI creation |

### *Screen Organisation Matrix*

The below matrix shows the organisation of the watch Screens. The screen flow is divided in to a 5*5 matrix. At any given time, only one of the matrix screens is an active watch screen. User will scroll horizontal and vertical to reach the target screen. The notification popup screen comes into focus temporarily on top of the current active screen.

| Analog Watch Faces (AWF) | Digital Watch Faces (DWF) | Info Screens (IS) | Custom Apps (CA) | Custom Apps (CA) |
|---|---|---|---|---|
| 1 (Reserved for Default Watch face) | 1 (Reserved for Default Watch face) | 1 (Reserved for OS Use) | 1 (Reserved for OS Use) | 6 |
| 2 (Reserved for Default Watch face) | 2 (Reserved for Default Watch face) | 2 (Reserved for OS Use) | 2 | 7 |
| 3 | 3 | 3 (Reserved for OS Use) | 3 | 8 |
| 4 | 4 | 4 (Reserved for OS Use) | 4 | 9 |
| 5 | 5 | 5 (Reserved for OS Use) | 5 | 10 |

**Returns**

Successful execution of the function returns true.

**Throws**

NotPairedException

AuthFailedException

UnknownException

### *HOTDeviceContext OpenDevice(Context context, String applicationName, OnConStatusChangeListener Listener )*

**Description**

This function opens a virtual connection with the watch. If the watch is already paired with the phone, it will return an instance of HOTDeviceContext class. (see definition of HOTDeviceContext class). The app has to register first using RegisterApp before calling this function.

**Parameters**

| | |
|---|---|
| Context | Android application context. Andoid application context can be obtained by `getApplicationContext` |
| applicationName | Unique Name of the application initializing the connection. SDK Keeps a database of application installed and connected to watch. |
| Listener | Callback Event listner to obtain the the status of Bluetooth connection |

**Callback Event Listener Details**

```
enum BluetoothStatus{
	Connected,
	Disconnected,
	Unknown,
};
public interface OnConStatusChangeListener
{
	public abstract void OnStatusChnage(BluetoothStatus status);
}
```

**Returns**

Successful execution of the function returns a valid HOTDeviceContext class instance representing an established connection with SDK Context. This is equivalent to a virtual connection with the watch.

**Throws**

NotPairedException

AuthFailedException

UnknownException

### *Boolean* CloseDevice*(HOTDeviceContext deviceContext)*

**Description**

This function closes the virtual connection with the watch and SDK Context.

**Parameters**

| | |
|---|---|
| deviceContext | Device context object of the currently opened device. |

**Return**

Return true if a valid device context is supplied and closed it properly.

### *Boolean TurnOnFastRefresh(int Duration,OnRefreshModeChangeListner listner)*

**Description**

This function switches watch to fast refresh mode for specified period (in seconds). The watch will return to normal mode after the specified period. If the application wants to keep the watch in fast refresh mode for more than the specified period (or more than the allowed max time period of 3 minutes), then the application has to reissue another TurnOnFashRefresh command. RefreshModeChange listener event (which is called when fast refresh stops) can be used to put the device back to fast refresh mode.

**Parameters**

| | |
|---|---|
| Duration | Duration in seconds. Maximum is 180 (3 minutes) |

**Callback Event Listner Details**

```
enum RefreshMode
{
Normal,
FastRefrshMode,
MediumRefreshMode
};
public interface OnRefreshModeChangeListner
{
   void OnRefreshModeChange(RefreshMode mode);
}
```

**Return**

Return true if function call successful.

Note: In normal mode the watch refreshes once a minute. In the fast refresh mode the watch increase the refresh rate to 3 times a second. The fast refresh mode drains battery at much faster rate and should be used sparingly.

### *Boolean GetBluetoothConnectionStatus()*

**Description**

Function retrieves the current status of Bluetooth connection. This function is useful to check the Bluetooth status if call-back was not registered.

**Return**

Return true if Bluetooth connected.

## 3.3 Communication Channel Management

The native apps can be standalone native apps or native apps that work with companion phone app. The functions in this section are used to setup a communication channel between phone app and the corresponding companion native app in the watch. These functions are used only when a companion native app needs to communicate with the phone app. This channel is designed as a raw data channel to allow the application to define own protocols and exchange any type of data.

### *HOTComChanel class*

This class implements communication channel functions.

### *Boolean CreateComChanel(String appName[10], int chanelID, CommunicationChanelListner listner)*

**Description**

This function creates a communication channel between phone application and watch application. Communication end points are identified by Application Name and chanelID. SDK supports multiple channels of communication between connected apps which is identified by channelID.

**Parameters**

| | |
|---|---|
| **appName** | A name used to identify the application end point. This field along with chanelID will make a unique end point |
| **chanelID** | A unique chanel ID used to identify the end point. This field along with appName will make a unique end point. A maximum of 4 channels are supported. |

| | |
|---|---|
| **listener** | Listner defines the callback function which is called when there is data arriving from watch app. |

**Callback Event Listner Details**

```
public interface ComChannelListners

{

	void OnDataRecv(int chanelID,byte [] buffer);

	void OnWriteComplete(int channelID,int status);

}

Applications using communication channel can implement this interface to receive
notifications on data reception and data send completion.
```

**Return**

Return true if function call successful.

### *Int WriteToChanel( int chanID, ByteArray buffer, WriteCompleteListner listner )*

**Description**

This function sends a packet of data to the registered end point.

**Parameters**

| | |
|---|---|
| **chanelID** | A unique channel ID used to identify the end point. |
| **buffer** | Data buffer to send |
| **len** | Length of the buffer. Max size is limited 10240bytes (10 blocks of 512 bytes) |
| ***listner*** | The below function defined as the listener is called when buffered data is completely transferred or an error occurred |

**Callback Event Listner Details**

**Return**

Function returns following values

| | |
|---|---|
| 1 | Function call successful |
| -1 | Chanel not opened |
| -2 | Previous data is not transmitted completely. Data not copied to the transmission buffer. |

## 3.4 Watchface Management Functions

Watchface Management functions provide powerful features to define watchfaces. Both Digital and Analog watchfaces are supported by the SDK. SDK functions generate Meta data which defines the watchface. This data is then downloaded to the watch and interpreted by the watch-OS to display the watchface. While creating a watchface user has to select a screen location for the watchface (This is done while calling the RegisterApp function). Out of the supported 5 analog and 5 digital watchfaces on the watch, 3 analog and 3 digital watchfaces are available for SDK apps.

Note: Watchface type is selected during the RegisterApp call.

### *Watch-face Functions*

#### *HOTWatchFace Class*

**Public Constructor**

##### *Public HOTWatchFace(HOTDeviceContext deviceContext)*

**Description**

Initialise watchface interface

**Parameters**

| | |
|---|---|
| **deviceContext** | Device context object retuned by the OpenDevice function. |

##### *Boolean ClearScreen()*

**Description**

Sets background to white

Return

Return true if the function call successful.

### *Boolean SetBackgroundImage(Bitmap bmp)*

**Description**

Set screen background with the given bitmap

**Parameters**

| | |
|---|---|
| **bitmap** | Provide a Monocrome bitmap which match the size of watch face (144x1568 pixels, 1 bit per pixel) . |

**Return**

Return true if function call is successful.

### *Boolean LoadHourNeedle(Point needleFixPt, Point needlePivotPt,Bitmap needleImg)*

**Description**

This function provides the needed information create hour needle. User has to provide 12o clock position of the needle. The SDK creates other positions of the needle. This function is used only for analog watchfaces.

| | |
|---|---|
| needleFixPt | Location where needle is positioned in the watchface screen image (144x168). This is the position where the needle top right is positioned. User should ensure the needle will fit within the screen in all rotated positions. |
| needlePivotPt | Needle rotated around this point. This is the pivot point in the needle image from top-left corner of the image. |
| needleImage | This is the image of the needle in its 12o clock position. Needle Image to use. Image has to be smaller than ½ screen height (168/2) and smaller than 24 pixels wide. |

**Return**

Return true if function call is successful.

### *Boolean LoadMinutesNeedle(Point needleFixPt, Point needlePivotPt,Bitmap needleImg)*

**Description**

This function provides the needed information create minute needle. User has to provide 12o clock position of the needle. The SDK creates other positions of the needle. The SDK creates other positions of the needle. This function is used only for analog watchfaces.

| | |
|---|---|
| needleFixPt | Location where needle is positioned in the watchface screen image (144x168). This is the position where the needle top right is positioned. User should ensure the needle will fit within the screen in all rotated positions. |
| needlePivotPt | Needle rotated around this point. This is the pivot point in the needle image from top-left corner of the image. |
| needleImage | This is the image of the needle in its 12o clock position. Needle Image to use. Image has to be smaller than ½ screen height (168/2) and smaller than 24 pixels wide. |

**Return**

Return true if function call is successful.

### *Boolean DisplayTimeParts(String timePart, Point location, Size size, NumberStyle style)*

***Description***

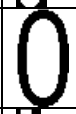
While creating Analog or digital watchfaces, it is possible to display parts of the Time as text fields. This function is used to place these units of the time (HH, MM, etc) on the screen.

| | |
|---|---|
| timePart | This string represents the part of the date or time to be displayed at the specified location<br>HH:- Hour in 24 hour format<br>hh:- Hour in 12 hour format<br>MM:-Minutes<br>SS:- Seconds. Seconds are normally off. They are switched on for about 90 seconds only when user makes a long touch in the watch-face<br>AMPM:-AM /PM part of the time |
| location | The screen co-ordinate where location is displays |
| Size | Size of the window to display time part. Size must be good enough to hold the content. |

| | |
|---|---|
| | Otherwise content may clipped |
| style | HOT Smart Watch OS support many different number styles to select from. See below table. Future versions of SDK will support custom style |

### Number Styles

The below number styles are currently supported on the SDK. The future version will support custom styles.

| | Code | Size pixels (WxH) | Font | Image |
|---|---|---|---|---|
| 0-9 digits | WFNS_SMALL_ARIALREVERSE | 8 x 14 | Arial Reverse |  |
| 0-9 digits | WFNS_MEDIUM_ARIAL | 16 x 24 | Arial |  |
| 0-9 digits | WFNS_SMALL_ARIAL | 8 x 14 | Arial |  |
| 0-9 digits | WFNS_ BIG_ARIALREVERSE | 20 x 35 | Arial Reverse |  |
| 0-9 digits | WFNS_LARGE_ARIAL | 34 x 64 | Arial |  |
| 0-9 digits | WFNS_ MEDIUM_ARIALREVERSE | 16 x 16 | Digital Reverse |  |
| 0-9 digits | WFNS_ MEDIUM_DIGITAL | 24 x24 | Digital |  |
| 0-9 digits | WFNS_LARGE_SEVENSEG | 32 x 48 | 7Segment |  |
| 0-9 digits | WFNS_BIG_SEVENSEG | 24 x32 | 7 segment |  |

### *Boolean DisplayDateParts(String datePart,Point location,Size size, DateStyle style)*

### *Description*

While creating Analog or digital watchfaces, it is possible to display parts of the Date as text fields. This function is used to place these units of the date (DD MM etc) on the screen.

.**Parameters**

| | |
|---|---|
| datePart | This string represents the date part to be displayed at the specified location<br>dd:- Day of week (SUN, MON etc)<br>DD:- Day of month<br>MM:-Month<br>YY: Year short form<br>YYYY: Year, long form |
| Location | The screen co-ordinate where to display the item. |
| Size | Size of the window to display time part. Size |

| | |
|---|---|
| | must be good enough to hold the content. Otherwise content may clipped |
| Style | HOT Smart Watch OS support many different number styles to select from. The time table above can be used for digits and weekday table below can be used for weekday and month |

**Date Part Styles**

| | Code | Size pixels (WxH) | Font | Image |
|---|---|---|---|---|
| **Weekday** | | | | |
| 3 digit Sun to Sat | WFDS_SMALL_ARIAL | 42 x 16 | Arial |  |
| 3 digit Sun to Sat | WFDS_MEDIUM_DIGITAL | 48 x 18 | Digital |  |
| 3 digit Sun to Sat | WFDS_LARGE_DIGITAL | 72 x 25 | Digital |  |
| Weekday full | WFDS_WEEKFULL_DIGITAL | 144 x 18 | Digital |  |
| Weekday Full | WFDS_WEEKFULL_7SEG | 144 x 20 | 7 segment |  |
| **Month** | | | | |
| 3 digit Jan to Dec | WFDS_MONTH_ARIAL | 42 x 16 | Arial |  |
| 3 digit Jan to Dec | WFDS_MONTH_7SEG | 64 x 32 | 7 segment |  |

### *Boolean UploadWatchFace()*

**Description**

This function uploads the meta-data generated to the watch. All the watchface functions are buffered and sent after this function is called.

**Return**

Return true if function call is successful.

## 3.5 Message Display Functions

These set of function allow host app to display a Text/Message box in the watch screen and accept simple Yes/No/Cancel input from the user.

### *HOTMesgDisplay Class*

This class implements the functionality required for notification message display

**Public Constructor**

### *Boolean HOTMesgDisplay(HOTDeviceContext deviceContext)*

**Description**

Initialise display system.

**Parameters**

| | |
|---|---|
| **deviceContext** | Device context object retuned by the OpenDevice function. |

### *Boolean DisplayAlert(String Caption,String Message, Boolean Vibrate)*

**Description**

This function displays an Alert box with text. The top line displays caption with bold text. Text may be clipped if it is longer than 12-14 characters. The next 3 lines in the watch face are used to display the message. The text may remain in the watchface until user touches it or cancels using gesture. A new event from any other source can cause text disappear from the watch face.

**Parameters**

| | |
|---|---|
| **Caption** | Alert caption text. Max length is limited to 14 characters |
| **Message** | Alert message maximum text length is limited to 28 characters. |
| **Vibrate** | Provides a short vibration to grab user attention |

**Return**

Return true if function call is successful.

### *Int DisplayMessageBox(String Caption,String Message, int Options,Boolean Vibrate,int TimeOut)*

**Description**

This function displays a message box in the watch screen. The top line displays caption with bold text. Text may be clipped, if its length is bigger than 12-14 characters. The next 2 lines in the watch face are used to display the message text. Fourth line displays buttons to accept user response. The text may remain in the watch face until user touches it or cancels using gesture. A new event from any other source can cause text disappear from the watch face. Message box is sticky and needs a user response to remove it before time out period. If the user is not responding within the timeout period, then Message box is discarded and timeout is sent back to the calling app.

**Parameters**

| | |
|---|---|
| **Caption** | Alert caption text. Max length is limited to 14 characters |
| **Message** | Alert message maximum text length is limited to 28 characters. |
| **Options** | Gives the user response options.<br>YES_NO<br>OK_CANCEL |
| **Vibrate** | Provides a short vibration to grab user attention |

**Return**

Function returns an integer value which indicates successful execution of the function.

| | | |
|---|---|---|
| MB_FAIL | -1 | MessageBox Failed .Either displayed or not displayed. May be device disconnected |
| MB_TIMEOUT | -2 | MessageBox may be displayed to user. But no response received from the watch after initial acknowledgment. May be user not responded or device disconnected |
| MB_OK | 1 | User response OK |
| MB_CANCEL | 2 | User response CANCEL |
| MB_YES | 3 | User response YES |
| MB_NO | 4 | User response NO |

## 3.6 Graphics Functions

### *HOTSmartWatchGraphics Class*

This class implements the graphics functions. The communication between watch and phone is optimised for energy efficiency. The graphics function involves a lot of data exchange between phone and watch. To minimize the data transfer and optimize the display refresh, the graphics functions are buffered until ***UpdateScreen*** function is called.

**Public Constructor**

### *HOTSmartWatchGraphics(HOTDeviceContext deviceContext)*

**Description**

Initialise graphics system.

**Parameters**

| | |
|---|---|
| **deviceContext** | Device context object retuned by the OpenDevice function. |

### *Boolean ClearScreen(Color color)*

**Description**

This function Clears the screen and is typically called before starting a drawing**.**

**Note: This function ( and all other graphics functions) will not have any immediate effect on the watch screen, unless otherwise UpdateScreen function is called.**

**Return**

Return true if function call is successful.

### *Boolean SetScreenBitmap(Bitmap backGround,BackGroundBMPOption option)*

**Description**

Sets a screen background bitmap based on the selected option . Screen real estate available to the application's use is limited 144x150pixels. Pixel size is 1 bit.

**Parameters**

| | |
|---|---|
| **background** | Bitmap to load in to screen buffer |
| **option** | See below given table for options |

Bitmap display options

| | |
|---|---|
| BM_TOPLEFT | Align bitmap to top,left and overwrite the existing content at the selected location |
| BM_TOPRIGHT | Align bitmap to top,right and overwrite the existing content at the selected location |
| BM_CENTER | Align bitmap to center |
| BM_FILL | Stretch/Shrink tmap to fill the screen |

**Return**

Return true if function call is successful.

### *Boolean DrawPixel(Point pt, Color color)*

**Description**

Draws a pixel at the absolute position given at pt parameter. Top-Left corner is represented as 0,0**. Note: While the watchface apps have access to 144x168 pixels of the watch display, other apps have access to only 144x150pixels. The top 18 rows are used to display window title information.**

**Parameters**

| | |
|---|---|
| **pt** | Contains the X,Y co-ordinate. |
| **color** | Color. Currently supported only BLACK and WHITE |

**Return**

Return true if function call is successful.

### *Boolean DrawLine(Point ptStart, Point ptEnd, Color color)*

**Description**

Draws a line from ptStart to ptEnd in absolute coordinates. Top-Left corner is represented as 0,0

Note: While the watchface apps have access to 144x168 pixels of the watch display, other apps have access to only 144x150pixels. The top 18 rows are used to display window title information.

**Parameters**

| | |
|---|---|
| **ptStart** | Line start point |
| **ptEnd** | Line End Point |
| **color** | Color. Currently supported only BLACK and WHITE |

**Return**

Return true if function call is successful.

### *Boolean DrawRect(Rect rect, Color color, short colFill)*

**Description**

Draws a rectangle with the given co-ordinates. Top-Left corner is represented as 0, 0.

Note: While the watchface apps have access to 144x168 pixels of the watch display, other apps have access to only 144x150pixels. The top 18 rows are used to display window title information.

**Parameters**

| | |
|---|---|
| **rect** | Co-ordinates to draw rectangle |
| line Color | Color .Currently supported only BLACK and WHITE |
| **colFill** | Fill Color. If 0, filling is not done |

**Return**

Return true if function call is successful.

### *Boolean DrawCircle(Point center, int radius, Color color)*

**Description**

Draws a circle with the given co-ordinates. Top-Left corner is represented as 0, 0.

| |
|---|
| Note: While the watchface apps have access to 144x168 pixels of the watch display, other apps have access to only 144x150pixels. The top 18 rows are used to display window title information. |

**Parameters**

| | |
|---|---|
| **center** | Centrer of circle |
| **radius** | Radius in pixels |
| **color** | Color. Currently supported only BLACK and WHITE |

**Return**

Return true if function call is successful.

### *Boolean DrawBitmap(Point topLeftPoint, Bitmap bmp, BitmapOptions option)*

**Description**

Draws a bitmap with given image bitmap. If the bitmap is larger than screen area (144x150), the bitmap is clipped.

Note: While the watchface apps have access to 144x168 pixels of the watch display, other apps have access to only 144x150pixels. The top 18 rows are used to display window title information

**Parameters**

| | |
|---|---|
| **point** | Top point to position bitmap |
| **bmp** | Bitmap to place. Bitmap placed with its original size |
| **option** | Bitmap drawing options. See below table |

**Bitmap Options**

| | |
|---|---|
| BM_OVERWRITE | Bitmap overwrite the background image in the screen. |
| BM_OR | Bitmap placed on the background image using OR operation. Any black (foreground) area will appear in the final image and mixed with background image |
| BM_AND | Bitmap placed on the background image using AND operation. Black pixels will appear only if both background and foreground images have it in black. |
| BM_XOR | Bitmap placed on the background image using XOR operation. Black pixels will appear only if either background or foreground images have it in black. |
| BM_XNOR | Bitmap placed on the background image using !XOR operation. |

**Return**

Return true if function executes successfully.

### *Boolean DrawBitmap(Rect rect,Bitmap bmp,BitmapOptions option)*

**Description**

Draws a bitmap using a bitmap image and stretching its contents to a rectangle**.** Original bitmap is compressed or enlarged to fit in the given rectangle.

Note: While the watchface apps have access to 144x168 pixels of the watch display, other apps have access to only 144x150pixels. The top 18 rows are used to display window title information.

**Parameters**

| | |
|---|---|
| **rect** | Rect size to stretch built the bitmap |

| | |
|---|---|
| **bmp** | Bitmap to place. |
| **option** | Bitmap drawing options. See below table |

**Bitmap Options**

| | |
|---|---|
| BM_OVERWRITE | Bitmap overwrite the background image in the screen. |
| BM_OR | Bitmap placed on the background image using OR operation. Any black (foreground) area will appear in the final image and mixed with background image |
| BM_AND | Bitmap placed on the background image using AND operation. Black pixels will appear only if both background and foreground images have it in black. |
| BM_XOR | Bitmap placed on the background image using XOR operation. Black pixels will appear only if either background or foreground images have it in black. |
| BM_XNOR | Bitmap placed on the background image using !XOR operation. |

**Return**

Return true if function executes successfully.

### *Boolean DrawText(String message,Point position,Size windowSize,FontSizeType fontSizeType, Align align,Style style)*

**Description**

This function displays a text box on the watch screen. This function is used to display text in the portion of a screen.

Note: While the watchface apps have access to 144x168 pixels of the watch display, other apps have access to only 144x150pixels. The top 18 rows are used to display window title information.

**Parameters**

| | |
|---|---|
| **message** | Text to display in the text box. |
| **position** | Top Left X,Y location of the string |
| **windowSize** | Size of the display string window |
| **fontSizeType** | Font size and type. See table below to find |

| | |
|---|---|
| | the possible options |
| **align** | Text alignment in the display. |
| **style** | Text Display Style. See table below to find the possible option |

**FontSizeType**

The below enumerations are used to select the font size currently supported by watch OS.

| | |
|---|---|
| FONT_SMALL_NORMAL | 9 Point Font Verdana |
| FONT_SMALL_TALL | 12 Point Amplitude |
| FONT_MEDIUM_NORMAL | 13 Point Font Verdana |
| FONT_MEDIUM_TALL | 18 Point Amplitude |
| FONT_LARGE_NORMAL | 14 Point Font Verdana |
| FONT_LARGE_TALL | 20 Point Amplitude |

**Text Alignment**

| | |
|---|---|
| TXT_ALIGN_LEFT | Align text to left |
| TXT_ALIGN_CENTER | Align text to the center. |
| TXT_ALIGN_RIGHT | Align text to right. |

**Style**

| | |
|---|---|
| TEXT_SINGLE_LINE | Single Line of Text. Max size is limited to 14 |
| TEXT_MULTI_LINE | Multiple line text. Max Size is limited to 48 characters. |
| TEXT_MULTI_LINE_SCROLL | Multiple line text with up-down scroll. Max Size is limited to 128 characters. |

**Return**

Return true if function executes successfully.

### *Boolean UpdateScreen(DispStatusListner listner,DisplayOption option)*

**Description**

This is the function that flushes buffered commands and updates the screen.

Outputs current screen buffer to the selected screen location in the watch**.** While calling other display related functions, SDK just prepares the screen meta-data in a temporary buffer. This meta-data is send to watch after calling *UpdateScreen* function. This is done to improve the communication efficiency between Watch and Phone**. Clear***Screen* function clears the SDK screen buffer.

Note: While installing an application, it is assigned a screen location. Refer Display Organisation section in SDK Capability Documentation**.**

**Callback Event Listener Details**

```
public interface DispStatusListner

{

      void OnDisplayUpdate(int status);

}

SDK context may not transfer display data to the watch until user selects the app
in the watch and brings the app in to the focus. Apps can use this call back to
get  notifications when screen data is transferred to watch and presented to
user.
```

**Parameters**

| | |
|---|---|
| **listener** | This call-back called after the display is successfully updated on the watch. There may be few seconds delay in the watch updated. |
| **Option** | This option allows the app to bring it's screen to focus (if it is not the current screen). If user does not take any action after bringing the screen to focus, the watch will roll back to previous screen. |

**Display Options**

| | |
|---|---|
| DISP_UPDATEONLY | This will update the app's screen. If the app is |

| | |
|---|---|
| | not currently in focus on the watch, no update happens on the watch display. |
| DISP_UPDATENDISP | This will bring the app's screen into focus and update the screen with its graphics. |

**Return**

Return true if function executes successfully.

## 3.7 Vibration Motor

### *HOTSmartWatchUtility class*

This class implements the utility functions. It includes Vibration, LED and Backlight functions.

**Public Constructor**

### *HOTSmartWatchUtility(HOTDeviceContext context)*

**Description**

Initialise utility functions.

**Parameters**

| | |
|---|---|
| **deviceContext** | Device context object retuned by the OpenDevice function. |

### *Boolean VibrateOn(VibrationType type, int Duration)*

**Description**

The watch vibration motor is used for alerting the user. The watch uses 5 patterns of alerts are used for different indications and not shared for SDK. SDK provides 3 user assignable patterns to alert the user.

**Parameters**

| | |
|---|---|
| **type** | VibrationType. See below for pre-defined types |
| **Duration** | Duration specified in milliseconds |

**Vibration Types**

| | |
|---|---|
| VIBE_INCOMINGCALL | Medium pattern (used for notifying incoming call). Not available for SDK Use |

| | |
|---|---|
| VIBE_MESGALERT | Medium pattern (used for notifying message alerts)<br><br>Not available for SDK Use |
| VIBE_PHONELOST | Long spin (used for notifying Phone lost )<br><br>Not available for SDK Use |
| VIBE_BTCONNECT | Small spin (used for notifying BT connect )<br><br>Not available for SDK Use |
| VIBE_TOUCH | Very Small spin (used for single tactile touch )<br><br>Not available for SDK Use |
| VIBE_USER_SHORT | Short Pattern reserved for user application. Available for SDK use. |
| VIBE_USER_MEDIUM | Medium Pattern reserved for user application.<br><br>Available for SDK use. |
| VIBE_USER_LONG | Long Pattern reserved for user application.<br><br>Available for SDK use. |

**Return**

Return true if function executes successfully.

## 3.8 Back-light and LED Flash Light Functions

The watch comes with LED backlight for viewing in dark as the e-paper display is not visible in dark. The curve model also has additional flash light. SDK functions allow user to control Back Light and Flash Light. LED Light Functions are available in both Phone SDK and Native Watch SDK.

### *HOTSmartWatchUtility class*

This class implements the LED and flash light functions along with Vibration functions. See construction definition in the section 3.7

### *Boolean LEDFlash(int onSeconds,Brightness bright)*

**Description**

Function switch on the LED flash light for the specified time period and with specified brightness

**Parameters**

| | |
|---|---|
| **OnSeconds** | Duration specified in seconds (Max 10 seconds) |
| **Bright** | Led brightness. See below table |

| | |
|---|---|
| BRIGHT_DIM | ¼ th of Full brightness |
| BRIGHT_HALF | 1/2$^{th}$ of Full Brightness |
| BRIGHT_FULL | Full Brightness |

**Return**

Return true if function executes successfully.

### *Boolean LEDBackLight(int onSeconds, Brightness bright)*

**Description**

This function turns on the display backlight for the specified time period and with specified brightness

**Parameters**

| | |
|---|---|
| **OnSeconds** | Duration specified in seconds (Max 20 seconds) |
| **Bright** | Led brightness. See below table |

**Brightness Options**

| | |
|---|---|
| BRIGHT_DIM | ¼ th of Full brightness |
| BRIGHT_HALF | 1/2$^{th}$ of Full Brightness |
| BRIGHT_FULL | Full Brightness |

**Return**

Return true if function executes successfully.

## 3.9 Accelerometer & Gyroscope

HOT Smart Watch is equipped with a gyroscope and accelerometer for gesture detection and fitness functions.

### *HOTAccelGyroManager class*

This class implements Accelerometer and Gyroscope functions.

**Public Constructor**

### *HOTAccelGyroManager(HOTDeviceContext context)*

**Description**

Initialise Accel and Gyro functions.

**Parameters**

| | |
|---|---|
| **deviceContext** | Device context object retuned by the OpenDevice function. |

### *Boolean InitGyroService(SensorSamplingRate samplingRateHint,int queueSize,OperationMode mode,int timeOut,SensorEventListner Listner,)*

**Description**

Initialises Gyroscope (gyro) functionality and register an event handler to receive gyro change events. During the gyro functionality the watch is either put in fast or medium refresh mode so that the sensor data is sent periodically to the phone.

The gyroscope consumes high power (many times more than the accelerometer). The gyroscope functions need to be used sparingly. Due to this higher power consumption, the gyroscope functions have a time limit after which they stop sending data. The calling app has to re-issue this command.

**Parameters**

| | |
|---|---|
| Rate | Sampling rate at which to sample the gyro data. To limit power consumption, user should avoid using higher speeds.<br>SENSOR_NORMAL<br>SENSOR_GAME<br>SENSOR_HIGHSPEED |
| queueSize | Maximum Number of samples to be stored (default size is 100). If this queue size is small, the data may be lost in normal mode where watch is sending data once in 5 seconds. |
| Mode | The mode in which the watch sends data to phone. Please see below table for options |
| timeOut | Time out period in seconds. |

| | |
|---|---|
| | After time out period gyro service stops automatically. This functionality is added to save battery. |
| Listener | Listener for sensor events. See SensorEventListner definition |

**Gyro Data Delivery Mode**

| | |
|---|---|
| SENSOR_NORMAL_MODE | The sensor data is sent to phone once in 5 seconds as watch is in medium refresh mode and wakes once in 5 seconds |
| SENSOR_REALTIME_MODE | The watch is put in fast refresh mode and sends data up to 3 times a second as the sensor data gets available. |

**Callback Event Listner Details**

```
enum SensorType
{
       Accelerometer,
       Gyroscope,
       Tap,
       Step
}
public interface SensorEventListner
{
  void OnSensorChanged(SensorType sensorType,int numSamples, float [] values);
}
```

| | |
|---|---|
| sensorType | SensorType enum value used to identify the sensor. |
| numSamples | Number of samples in the values array. Each sample contains 3 co-ordinates.<br><br>-1 :- in numSamples indicate gyro timed out |
| Values | Value[n*0] – X value<br><br>value[n*1] – Y Value<br><br>value[n*2] – Z Value<br><br>“n” represent the sample number. If the num samples is 10 then “n” falls between 0-9.<br><br>Note : Unit is radians/second |

**Return**

Return true if function executes successfully.

### *Boolean InitAccelService(SensorSamplingRate samplingRateHint,int queueSize,OperationMode mode,int timeOut,SensorEventListner Listner,)*

**Description**

Initialises Accelerometer (Accel) functionality and register an event handler to receive Accel change events. During the Accel functionality the watch is either put in fast or medium refresh mode so that the sensor data is sent periodically to the phone.

The Accel consumes power both due to accelerometer consumption and faster refresh rate for the watch. The Accel functions need to be used sparingly. Due to this higher power consumption, the Accel functions have a time limit after which they stop sending data. The calling app has to re-issue this command.

Initialises accelerometer functionality and register an event handler to receive acceleration change events. Certain accelerometer modes will work only in fast refresh mode. When device switches back to normal mode from fast refresh mode, the sdk will stop sending the data.

**Parameters**

| | |
|---|---|
| Rate | Sampling rate at which to sample the gyro data. To limit power consumption, user should avoid using higher speeds.<br>SENSOR_NORMAL<br>SENSOR_GAME<br>SENSOR_HIGHSPEED |
| queueSize | Maximum Number of samples to be stored (default size is 100). If this queue size is small, the data may be lost in normal mode where watch is sending data once in 5 seconds. |
| mode | The mode in which the watch sends data to phone. Please see below table for options |
| timeOut | Time out period in seconds.<br>After time out period gyro service stops automatically. This functionality is added to save battery |
| listener | Listener for sensor events. See SensorEventListner definition |

**Sensor Data Delivery Mode**

| | |
|---|---|
| SENSOR_NORMAL_MODE | The sensor data is sent to phone once in 5 seconds as watch is in medium refresh mode and wakes once in 5 seconds |
| SENSOR_REALTIME_MODE | The watch is put in fast refresh mode and sends data up to 3 times a second as the sensor data gets available. |

```
enum SensorType
{
       Accelerometer,
       Gyroscope,
       Tap,
       Step
}
public interface SensorEventListner
{
  void OnSensorChanged(SensorType sensorType,int numSamples, float [] values);
}
```

| | |
|---|---|
| sensorType | SensorType enum value used to identify the sensor. |
| numSamples | Number of samples in the values array. Each sample contains 3 co-ordinates.<br><br>-1 :- in numSamples indicate accelerometer timed out |
| Values | Value[n*0] – X value<br><br>value[n*1] – Y Value<br><br>value[n*2] – Z Value<br><br>“n” represent the sample number. If the num samples is 10 then “n” falls between 0-9.<br><br>Note : Unit is radians/second |

**Return**

Return true if function executes successfully.

### *Boolean InitTapService(TapType type, TapEventListner listner,)*

**Description**

Initialises the tap detection from the accelerometer and register an event handler to receive tap events. If user taps the watch, it will cause watch to wake up and send a message to phone to notify the tap event.

**Parameters**

| | |
|---|---|
| Type | There are 2 predefined tap events which device is capable. This parameter array carries the types of tap application want to subscribe. See below table for possible values |
| listener | Listener for sensor events. See TapEventListner definition |

**Sensor Data Delivery Mode**

| | |
|---|---|
| TAP_STRAP_TAP | Single tap on metal to strap area facing the user |
| TAP_SWITCH_TAP | Single tap on the metal near power switch of the watch |

**Callback Event Listner Details**

```
enum TapEventType
{
       SrapTap,
       SwitchTap
}
public interface TapEventListner
{
  void OnTapEvent(TapEventType tapEventType);
}
```

| | |
|---|---|
| tapEventType | Type of tap event detected. See TapEventType definition above |

**Return**

Return true if function executes successfully.

### *Boolean InitStepCounter(StepListner Listner,int interval)*

**Description**

Initialise pedometer for counting the steps. The pedometer will consume additional power on the watch when it is on. This function should be used only when step count is required. The listener is called once a minute to report the total steps walked and time it took to walk these steps.

**Parameters**

| | |
|---|---|
| Listner | Listner defines a function with following interface, |
| interval | Steps count notification interval in seconds |

**Callback Event Listener Details**

```
public interface StepCountListner

{

  void OnStepCount(int stepCount, int timeInSecons);

}
```

| | |
|---|---|
| stepCount | Accumulated steps |
| timeInSeconds | Total time walked . In a minute if the user walked for 30 seconds and stayed idle for 30 seconds , then this value shows 30 seconds |

```
Note: Steps counter is reset by the hardware. When registered for the step
counter that first delivered values are the current counter values.
```

**Return**

Return true if function executes successfully.

### *Boolean CloseGyroscope()*

**Description**

Application not interested in gyro events anymore and instruct watch to stop sending it

**Return**

Return true if function executes successfully.

### *Boolean CloseAccelerometer()*

**Description**

Application not interested in accelerometer events anymore and instruct watch to stop sending it

**Return**

Return true if function executes successfully.

### *Boolean StopTapdetection()*

**Description**

Application not interested in tap events anymore and instruct watch to stop sending it

**Return**

Return true if function executes successfully.

### *Boolean StopStepCounter()*

**Description**

Application stops the step counting.

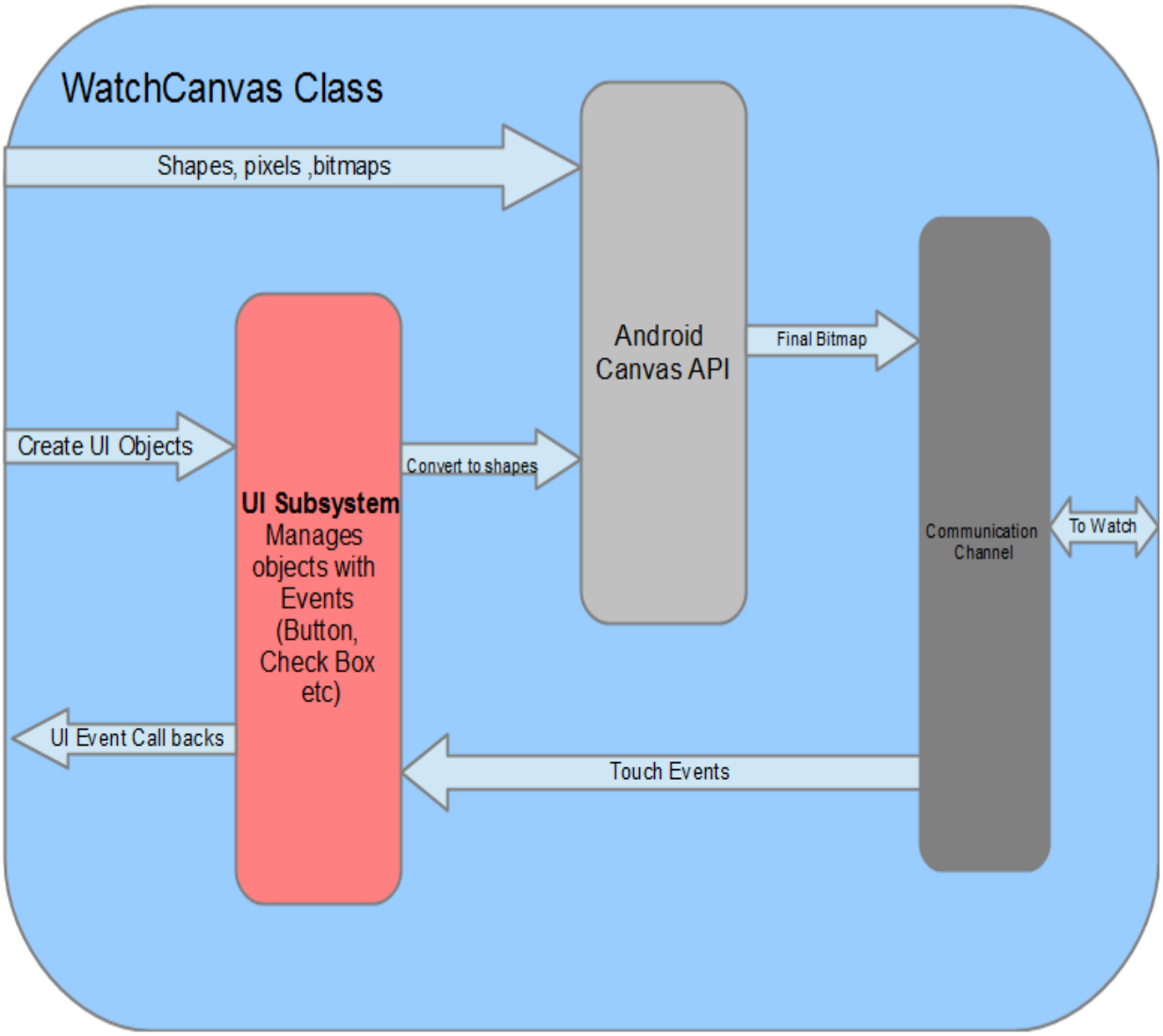
**Return**

Return true if function executes successfully.

## 4. Watch Canvas SDK

The Canvas SDK provides easier UI object generation (buttons, sliders etc.) using a canvas provided by the SDK. This functionality is built around Android Canvas 2D Drawing API. Android 2D Canvas API allows applications to manipulate a bitmap and draw different shapes and graphics in to it. Watch Canvas SDK wraps this functionality and provides a rich event driven programming environment. With the help of Android Canvas API it generates a bitmap at smart phone application and sends it to the Watch and renders it at watch face. Watch user who is seeing a familiar form type display at the watch screen and responds to it by clicking on the UI elements such as button clicks etc. The Host android application (running on the phone) which subscribed to the UI click events will receive a notification from the watch with details such as click co-ordinates etc. Watch Canvas SDK interprets this and identifies whether the click happens on button area. If so fires the corresponding button click handler in the WatchCanvas class.

The Watch Canvas functionality is very similar to the working of a HTML pages. In case of HTML, every UI event causes a post back to server and the event is handled at the server side. Watch canvas works very similar to this where watch-face does the role of browser and Phone app assumes the role of server.




## 4.1 Highlights

- **UI Elements supported by the watch canvas sub system**
  - **Graphics objects supported by Android Canvas API functions which includes** http://developer.android.com/reference/android/graphics/Canvas.html
    - **Pixel functions**
    - **Graphics shapes**
  - **Buttons**
  - **Radio Buttons**
  - **Selection Boxes**

- **Input boxes**
- **Labels (captions)**
- **Fonts and Char size selectable**
- **Image control**
- **Progress bar**

- **Partial rendering of watch screen for fast UI update.**
- **Eclipse plugin for UI design (Phase 2)**
  - **Android like UI definition language**

Note: First version of the SDK release may carry a subset of above mentioned functions.

## 4.2 HOTWatchCanvas Class

This class inherits all functionality provided by the Android Canvas class. These base class functions can accessed at the below URL

http://developer.android.com/reference/android/graphics/Canvas.html

In addition to base class functionality this class is extended to make it a container of other UI elements like buttons, selection boxes etc,etc. All these UI elements are defined as inner classes to WatchCanvas class. All these UI elements are derived from a common base class named UIObject.

**Public Constructor**

### *HOTWatchCanvas(ScreenType screenType,HOTDeviceContext context)*

**Description**

This is the default constructor of the Canvas. This function internally creates a blank bitmap equivalent to the size of watch face.

**Parameters**

| | |
|---|---|
| deviceContext | An instance of deviceContext which represents the connection with a HOT Smart Watch |
| screenType | Type of screen indication screen capabilities<br>SCRN_LOWRESMONO =144*150 , 1 Bit pixels |

### *HOTWatchCanvas(ScreenType screenType ,Bitmap background,HOTDeviceContext context)*

**Description**

This is the overloaded constructor of the Canvas. This function uses the bitmap supplied as the background. It stretches the bitmap to match the screen size.

**Parameters**

| | |
|---|---|
| background | Bitmap to use as the background for the canvas screen |
| deviceContext | An instance of HOTSmartWatch class which represents the connection with a HOT Watch |
| screenType | Type of screen indication screen capabilities<br>SCRN_LOWRESMONO =144*150 , 1 Bit pixels |

### *Void ClearCanvas(Color filColor)*

**Description**

Function clears the canvas and fills with the given color

**Parameters**

| | |
|---|---|
| FilColor | Clears the screen. Supported only BLACK and WHITE |

**Return**

none

### *boolean DisplayCanvas(DisplayStatusLisner listner,DisplayOption option )*

**Description**

This function sends the complete bitmap to the watch and displays it on the watch.

**Parameters**

| | |
|---|---|
| Listener | Call back function to receive the status of display success. |
| option | This option allows the app to bring it's screen to focus (if it is not the current screen). If user does not take any action after bringing the screen to focus, the watch will roll back to previous screen. |

**Callback Event Listner Details**

```
public interface DispStatusListner

{

      void OnDisplayUpdate(int status);

}
```

SDK context may not transfer display data to the watch until user selects the app in the watch and brings the app in to the focus.Apps can use this call back to get  notifications when screen data transferred to watch and presented to user.

| | |
|---|---|
| status | 1-Data Delivered<br>0-Data Delivery Failed<br>-1-Screen Displayed but no user response within time out period. |

**Return**

Return true if function executes successfully.

### *boolean UpdateCanvas(DisplayStatusLisner listner,DisplayOption option )*

**Description**

This function sends only changed portion of the screen from the last display command or update command. It may send multiple blocks of data (bitmaps) to the watch with co-ordinates to place them in certain portions of the screen. This will make the screen refresh more efficient by updating only the changed portion such as pressed button state.

**Parameters**

| | |
|---|---|
| listner | Call back function to receive the status of display success. |
| option | This option allows the app to bring it's screen to focus (if it is not the current screen). If user does not take any action after bringing the screen to focus, the watch will roll back to previous screen. |

**Callback Event Listener Details**

```
public interface DispStatusListner

{

      void OnDisplayUpdate(int status);
```

```
}

SDK context may not transfer display data to the watch until user selects the app
in the watch and brings the app in to the focus. Apps can use this call back to
get  notifications when screen data transferred to watch and presented to user.

status1-Data Delivered

0-Data Delivery Failed

-1-Screen Displayed but no user response
within time out period.
```

**Return**

Return true if function executes successfully.

### *boolean AddObject(UIObject uiElement)*

**Description**

This function is used to add UIObjects to canvas surface. This is not updated to watch till the DisplayCanvas or UpdateCanvas is called

**Parameters**

| | |
|---|---|
| uiElement | Objects like Button, CheckBox, ListBox etc |

Note: uiElement base class is explained in the next section

**Return**

Return true if function executes successfully.

**Helper Function**

The below functions are helper functions provided in canvas class for ease of use. They are repeating functions which can be called from other classes.

### *HOTDeviceContext getConnectedWatch()*

**Description**

Function retrieves the read only instance of connected watch

**Return**

A HOTDeviceContext object instance

### *int getScreenNumber()*

**Description**

Retrieves the current active screen number

**Return**

Returns screen number

### *Bitmap getScreenBitmap()*

**Description**

Retrieves the screen bitmap

**Return**

Returns screen bitmap is successful.

### *byte [] getScreenRawdata()*

**Description**

Retrieves the screen data as a byte array. This is the same data send to the watch.

**Return**

Returns screen raw data.

### *Void setScreenRawdata(byte [] screenData)*

**Description**

Load byte array in to screen data buffer.

## 4.2.1 UI Objects

This section describes the UI Objects available for use in the WatchCanvas class

### *4.2.1.1 Button*


The button class provides ability to create different style buttons on the screen. The default types of buttons are bitmap buttons with or without borders. It is also possible to create buttons with captions on it. Above figure shows a screen shot from the existing watch screen. This screen is a good example of multiple bitmap buttons.

**Properties**

| | |
|---|---|
| Caption | Text to display on the button face. Max 10 character |
| Icon | Button bitmap |
| Width | Width in pixels |
| Height | Height in pixels |
| TopX | Position of button in X co-ordinates |
| TopY | Position of button in X co-ordinates |
| ButtonType | UI_TEXTBUTTON :- Text Button<br>UI_BMPBUTTON:- Bitmap button<br>UI_COMPBUTTON:- Both Text and Caption |
| Font | Text Font |

Note: Due to size and resolution limitation on the watch, careful attention is needed for the design of the buttons, fonts and other UI objects.

## Methods

### *void Create(Rect rect, String caption, Bitmap icon,int uiElementID)*

**Description**

Creates the Button with bitmap and optional caption.

**Parameters**

| | |
|---|---|
| rect | Position and size of the button |
| caption | Caption of the button |
| icon | Bitmap to display on the button |
| *int uiElementID* | User assigned ID for the button. This is passed to click listner to identify the button. |

### *void Create(Rect rect,String caption)*

**Description**

Creates a blank Button with or without caption.

**Parameters**

| | |
|---|---|
| Rect | Position and size of the button |
| Caption | Caption of the button |

| | |
|---|---|
| ***int uiElementID*** | User assigned ID for the button. This is passed to click listner to identify the button. |

### *void Create(Rect rect,String caption, Bitmap icon,int uiElementID)*

**Description**

Creates the Button with bitmap.

**Parameters**

| | |
|---|---|
| Rect | Position and size of the button |
| Icon | Bitmap to display on the button |
| ***int uiElementID*** | User assigned ID for the button. This is passed to click listner to identify the button. |

### *void Create(String caption,int uiElementID)*

**Description**

Creates the Button with caption where the size is adjusted to fit the caption

**Parameters**

| | |
|---|---|
| Caption | Caption of the button |
| ***int uiElementID*** | User assigned ID for the button. This is passed to click listner to identify the button. |

### *void setOnClickLisner(OnClickListner listner)*

**Description**

This registers a call-back for button press

**Parameters**

| | |
|---|---|
| Listener | Call back function |

**Callback Event Listener Details**

```
public interface setOnClickListner

{

  void OnClick(int uiElementID);
```

```
}
```

### 4.3.2 ListBox

List box class allows user create a list box. The user can select an option from the given list. Below given screen is good example of List Box usage.

The list box concept on the watch is different from the normal windows based GUI. The list box is a one row item with title on the left and options on the right. With each touch, the options on the right change to next item. In the example given below "Font:Med" is a list box. Font is the caption and Medium is an option item in the list box. The other option items in this list box are small and large. With each touch, the option item changes to Small->Medium->Large in sequence. The below image has 4 list boxes with caption Power, Gesture and Font.


**Properties**

| | |
|---|---|
| Caption | Text to display on the Left |
| Row | Row to display option box 1-4 |
| ItemFont | Item Font |
| CaptionFont | Caption Font |
| ItemColor | Color (Black,White) background changes to opposite |
| CaptionColor | Color (Black,White) background changes to opposite |

### *void Create(int rowToDisp,String caption, String [] items,int uiElementID)*

**Description**

Creates a list box with given parameters.

**Parameters**

| | |
|---|---|
| rowToDisp | Row number where to display list box |
| Caption | List box caption |
| Items | String array containing items |
| ***int uiElementID*** | User assigned ID for the ListBox. This is passed to click listner to identify the ListBox. |

### *Void setOnClickListner(ListBoxOnClickListner listner)*

**Description**

Call-back for the listbox with current selection on the list box

**Parameters**

| | |
|---|---|
| Listener | Call back function |

**Callback Event Listner Details**

```
public interface ListBoxOnClickListner
{
      void OnItemClick(int itemID,int uiElementID);
}
```

| | |
|---|---|
| itemID | Current Item displayed in the listbox |
| uiElementID | User assigned list item ID |

## *4.3.3 ImageBox*

This class is used to display an image

**Properties**

| | |
|---|---|
| Caption | Image Caption |

| | |
|---|---|
| Rect | Size of image window |
| TopX | X position of image |
| TopY | Y Position |
| Bitmap | Display Image |
| CaptionColor | Color (Black,White) background changes to opposite |
| CaptionFont | |

### Void Create(Rect rect,String caption, Bitmap image,int uiElementID)

**Description**

Creates the Button with given properties.

**Parameters**

| | |
|---|---|
| Rect | Position and size of the ImageView |
| Caption | Caption of the Image |
| Image | Bitmap to display on the Image View |
| ***int uiElementID*** | User assigned ID for the button. This is passed to click listner to identify the button. |

### Void setOnClickListner(OnClickListner listner)

**Description**

Attach OnClickListner

**Parameters**

| | |
|---|---|
| Listener | Call back function |

**Callback Event Listener Details**

```
public interface setOnClickListner

{

  void OnClick(int uiElementID);

}
```

# 4. Native Watch SDK

Native Watch SDK is used to develop native watch applications in C language. The Native Watch apps are installed in the watch via connected phone. The Native SDK applications can be configured to start when OS starts, or when the user enter the application screen. These applications are loaded into specific locations in the memory. OS calls the "main" function to start the application. The main function must terminate immediately after registering for the interested events. If application takes too long to come out from any of SDK notification processing, the Watch restart and the application may be tagged for bad behaviour. OS will refuse to load such application when OS restarts.

## 4.1 Initialization Functions

### *void HOTInitSDK()*

**Description**

This function initialises all the SDK functions.

**Return**

None.

### *BOOL HOTInitAppContext(int *appID,char *appTitle[12], void MessageSink(int int ,int ,int , char * ));*

**Description**

This function is used to initialise an app instance. Watch OS keeps track of instances using appID. The MessageSink is a callback function that is called for all the events (touch, gesture etc) related to the app.

**Parameters**

| | |
|---|---|
| appID | Instance ID of the application returned by the function call. |
| appTitle | Application name to display on the title bar. This must be a unique application name. |
| ***void MessageSink(int int ,int ,int , char * )*** | This is the call back function which receives all OS notifications. The sample application explains the usage and functionality of this call-back. |

**Return**

Return true if function executes successfully.

### *BOOL HOTSetAppData(int appID,void * dataStruct)*

**Description**

This function allows applications to keep a data structure pointer in applications instance data. Application can define own specific data structure and keep application state in it. Whenever there is a call-back to an application, it can retrieve this structure to remember the application state.

Since the app does not have access to global data and has access only to local variables and heap data (HOTmalloc()), it's allocated memory pointer needs to be saved for later reuse when app gets control back. The app uses this function to save these allocated memory pointer data.

**Parameters**

| | |
|---|---|
| appID | Instance ID of the application returned by the function call. |
| dataStruct | Pointer to an application defined data structure. |

**Return**

Returns true if function call is successful.

### *void * HOTGetAppData(int appID)*

**Description**

This function retrieves the application data from the OS Instance table. Application stores its instance specific data to remember application state.

**Return**

Returns a valid pointer if successful. Otherwise return zero.

### *BOOL HOTEnableBluetoothNotifications(int appInstance )*

**Description**

This function enables Bluetooth notification to deliver to the application.

**Callback Notification**

Whenever there is change in Bluetooth connection status, the status is delivered to app's MessageSink call-back function.

**Parameters of MessageSink callback**

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_BLUTOOTH_STATUS |
| param1 | SWB_CONNECTED : Bluetooth connected<br>SWB_DISCONNECTED: Bluetooth disconnected. |
| param2 | Not used |
| buffer | Not used |

**Return**

Return true if function executes successfully.

### *int HOTGetBluetoothStatus(int appID)*

**Description**

Function returns the current status of Bluetooth connection with host phone

**Return**

1- If Connected

0-If not connected.

### *void * HOTMalloc(int appID,int len);*

**Description**

This function allocate memory for application needs.

**Return**

Null – Memory allocation failed

Otherwise returns a valid pointer

### *BOOL HOTFree(int appID,void *ptr);*

**Description**

This function frees the allocated memory.

**Return**

Return true if function executes successfully.

## 4.2 Display Functions

## 4.2.1 Graphics Functions

### *BOOL HOTActivateScreen(int appID, short seconds)*

**Description**

This function forces the watch to display the screen selected by the application. Application can use this when it need user attention and to display screen forcefully. This is a temporary focus for this app. After a timeout period, the control is given back to previous screen. A maximum time that an app can request for forcibly coming to front is 15 seconds.

**Return**

Return true if function executes successfully.

### *Void HOTClearScreen()*

**Description**

This function clears the screen content and set all pixel to white.

**Return**

void

### BOOL HOTPutPixel(int appID,int x,int y,Color color)

**Description**

This function draws a pixel in the app's screen.

| | |
|---|---|
| appID | Application ID returned by InitAppContext call |
| X | x Co-ordinate |
| Y | y Co-ordinate |
| color | Color of the pixel . Currently supported BLACK and WHITE only**.** Color is an enum value defined in the SDK |

**Return**

Return true if function executes successfully.

### BOOL HOTDrawLine(int appID,int xS,int yS,int xE,int yE,Color col)

**Description**

This function draws a line in the app's screen

| | |
|---|---|
| appID | Application ID returned by InitAppContext call |
| xS | Starting point x Co-ordinate |
| yS | Starting point y Co-ordinate |
| xE | Ending point x Co-Ordinate |
| yE | Ending point y Co-ordinate |
| col | Color of the line. Currently supported BLACK and WHITE only**.** Color is an enum value defined in the SDK |

**Return**

Return true if function executes successfully.

### BOOL HOTDrawRect(DWORD appID,Rect rect,Color boderCol, Color fillCol)

**Description**

This function draws a **rectangle** in the app's screen

| | |
|---|---|
| appID | Application ID returned by InitAppContext call |
| Rect | Rectangle co-ordinates. Rect is a structure define in SDK. Rect contains the X,Y co-ordinate of the starting point , and width and |

| | |
|---|---|
| | height of the rectangle in pixels |
| borderCol | Color of the border line . Currently supported BLACK and WHITE only. Color is an enum value defined in the SDK with option for selecting BLACK and WHITE |
| fillCol | Color to fill inside the rectangle. Color is an enum value defined in the SDK with option for selecting BLACK and WHITE |

**Return**

Return true if function executes successfully.

### *BOOL HOTDrawCircle(DWORD appID,int xC,int yC, int radius, Color boderCol)*

**Description**

This function draws a **Circle** in the app's screen.

| | |
|---|---|
| appID | Application ID returned by InitAppContext call |
| radius | Radius in pixels |
| xC | X co-ordinate of the center |
| yC | Y co-ordinate of the center |
| BoderCol | Circle's border color. Color is an enum value defined in the SDK with option for selecting BLACK and WHITE |

**Return**

Return true if function executes successfully.

## 4.2.1 UI Controls

These are the display utility functions to draw string, display message box etc.

### *TextStatus HOTDispText (DWORD appID, char mesg,short xS, short yS, short width, short height, FontType type, Allign alignStyle)*

**Description**

This function displays text with various formatting options.

| | |
|---|---|
| appID | Application ID returned by InitAppContext call |

| | |
|---|---|
| Mesg | Message to display. Max chars to display is limited to 14 |
| xS | X co-ordinate of the starting position to text display box |
| yS | Y co-ordinate of Starting position of text display box |
| Width | Width of the text display area. Average width of a character is 8 pix. This can vary depending on the selected text style. Maximum size of screen is 144 pixels |
| Height | Hight of the display area. Small: 14pix, Medium: 21Pix, Large:24pix |
| fontType | Type of font selected. FontType is a enum value defined in the SDK.See table below to find the possible options |
| Align | Align type selected. Align is an enum value defined in the SDK.<br>Options are<br>TXT_LEFT<br>TXT_CENTER<br>TXT_RIGHT |
| Style | Text box style . SDK provides option to display Single Line, Multiline and Multiline with scroll options. Ref description of style given below |

**FontSizeType**

This enum value is used to select the font size and type which are currently supported by watch OS.

| | |
|---|---|
| FONT_SMALL_NORMAL | 9 Point Font Verdana |
| FONT_SMALL_TALL | 12 Point Amplitude |
| FONT_MEDIUM_NORMAL | 13 Point Font Verdana |
| FONT_MEDIUM_TALL | 18 Point Amplitude |
| FONT_LARGE_NORMAL | 14 Point Font Verdana |
| FONT_LARGE_TALL | 20 Point Amplitude |

**Style**

| | |
|---|---|
| TEXT_SINGLE_LINE | Single Line of Text . Max size is limited to 14 |
| TEXT_MULTI_LINE | Multiple line text. Max Size is limited to 64 characters. |
| TEXT_MULTI_LINE_SCROLL | Multiple line text with up-down scroll. Max Size is limited to 128 characters. |

**Return**

**Returns TextStatus**

| | |
|---|---|
| TXT_SUCCESS | Test Display success |
| TXT_OVERFLOW_CLIP | Text size exceeds the limits applicable in the selected mode and clipped. This is a warning message and text is displayed on the watch. |
| TXT_WINDOWSMALL | Selected text window is too small for the selected font . Function fails |

### *BOOL HOTMessageBox( int instID, char caption, char mesg, int userOption)*

**Description**

This function displays MessageBox screen for the app.

| | |
|---|---|
| instID | Application ID returned by InitAppContext call |
| caption | Caption text |
| Mssg<br><br>userOption | Message to display.<br><br>Options to display for the user to select from<br>MB_OKCANCEL<br>MB_YESNO |

**Callback Notification**

Message box function will cause a message box to be displayed on watch face. This will remain there until the user selects an option. An OS message will be delivered to **MessageSink** callback after user respond to it. If user response is not received in a predefined time period , then it will send a time out message.

**Parameters of MessageSink callback**

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_MESSAGEBOX |
| param1 | This will contain user selected option<br>Possible values are<br>MB_OK<br>MB_CANCEL<br>MB_YES<br>MB_NO |
| param2 | Not used |
| buffer | Not used |

**Return**

Return true if function executes successfully.

### *BOOL HOTListBox(int appID, char **options, int numItems,int uiElementID)*

**Description**

This function displays list box with selections for the list item.


The list box concept on the watch is different from the normal windows based GUI. The list box is a one row item with title on the left and options on the right. With each touch, the options on the right change to next item. In the example given below “Font:Med” is a list box. Font is the caption and Medium is an option item in the list box. The other option items in this list box are small and large. With each touch, the option item changes to Small->Medium->Large in sequence. The below image has 4 list boxes with caption Power, Gesture and Font.

| | |
|---|---|
| appID | Application ID returned by InitAppContext call |
| options | Options to display in the list |
| numItems | Number of items in the array |

**Callback Notification**

ListBox function displays a ListBox on watch face. This will remain there until user selects an option. An OS message will be delivered to **MessageSink** callback after user respond to it. If user response is not received in a predefined time period , then it will send a time out message.

**Parameters of MessageSink callback**

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_LISTBOX |
| param1 | This will contain user selected options index |
| param2 | User assigned uiElementID for the object |
| buffer | Not used |

**Return**

Return true if function executes successfully.

### *BOOL HOTButton( int appID, int buttonID, Rect buttonSize, char * caption)*

**Description**

Creates a button control on the watch screen. When user presses the button, the corresponding handler function is called.

**Parameters**

| | |
|---|---|
| appID | Application ID |
| buttonID | Unique Id assigned to the button. This value is passed to call back function. |
| buttonSize | Size and location of the button. Rect structure is defined in the SDK. |
| Caption | Caption of the button. Maximum 12 characters |

**Callback Notification**

An OS message will be delivered to **MessageSink** callback when user touchs the button. An application can display more than one button on the screen. Touched button is identified by Button ID

**Parameters of MessageSink callback**

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_BUTTONPRESS |
| param1 | Button ID |
| param2 | Not used |
| buffer | Not used |

**Return**

Return true if function executes successfully.

### *void HOTReadString(int instID,int inputType, int len)*

**Description**

This function is used to read data from the watch virtual keyboard. The watch provides a T9 keyboard similar to one found in smaller cell phones and telephones. User can enter numerical or text data from this keyboard.

**Parameters**

| | |
|---|---|
| instID | Instance ID |
| inputType | Based on this value function displays either Number or Alpha Numeric key board. |
| Len | Number of characters to read. This must be less than 50 which is the maximum available space in the keyboard buffer. |

**Call-back Notification**

Read String function remains on the screen until user inputs something and touches the text/number. The user input will be delivered to **MessageSink function.** The character pointer returned is a pointer to keyboard buffer. It can be over written by next call to ReadString function. It is the applications responsibility to preserve the data if needed. The space reserved for keyboard buffer is 50 characters.

**Parameters of MessageSink callback**

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_READSTRING |

| | |
|---|---|
| param1 | Number of characters typed by the user. Excluding “enter” key. |
| param2 | Not used |
| buffer | Contains the pointer to keybord buffer |

**Return**

Return true if function executes successfully.

## 4.2.1 Utility Functions

### *Void HOTTurnOnFastRefresh( int instID,int timeSeconds)*

**Description**

This function switches the watch to fast refresh mode. In fast refresh mode the OS is checking the requests 3 times a minute and services the calls at faster rate. In normal mode, the OS serves application only once in a minute. Touch Interrupt and timer calls happen without this delay.

**Parameters**

| | |
|---|---|
| instID | Instance ID |
| timeInSeconds | Number of seconds to keep watch in fast refresh mode. Maxim allowed is 180 seconds |

**Callback Notification**

An OS message will be delivered to **MessageSink** to indicate the change in the state of operation of Watch.

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_FASTREFRSH_CHANGE |
| param1 | 0:-Changed back to normal mode.<br>1:-Changed to fast refresh mode. |
| param2 | Not used |
| buffer | Not used |

**Return**

Return true if function executes successfully.

### BOOL *HOTVibrateOn(int Duration)*

#### Description

This function switches on vibration motor for the duration provided in milliseconds

**Parameters**

| | |
|---|---|
| Duration | Duration specified in milliseconds (max 1500) |

**Return**

Return true if function executes successfully.

### BOOL *HOTVibratePattern(VibrationType type, int Duration)*

**Description**

The watch vibration motor is used for alerting the user. About 5 patterns of alerts are used for different indications. Other than these per-assigned types there are three user assignable patterns reserved to implement application specific notification.

**Parameters**

| | |
|---|---|
| **type** | VibrationType. See below for pre-defined types |
| **Duration** | If zero, Pattern is run once, Otherwise Pattern is repeated this many times (max 5 times) |

**Alert Options**

| | |
|---|---|
| VIBE_INCOMINGCALL | Medium pattern (used for notifying incoming call). Not available for SDK Use |
| VIBE_MESGALERT | Medium pattern (used for notifying message alerts)<br>Not available for SDK Use |
| VIBE_PHONELOST | Long spin (used for notifying Phone lost )<br>Not available for SDK Use |
| VIBE_BTCONNECT | Small spin (used for notifying BT connect ) |

| | |
|---|---|
| | Not available for SDK Use |
| VIBE_TOUCH | Very Small spin (used for single tactile touch )<br><br>Not available for SDK Use |
| VIBE_USER_SHORT | Short Pattern reserved for user application. Available for SDK use. |
| VIBE_USER_MEDIUM | Medium Pattern reserved for user application.<br><br>Available for SDK use. |
| VIBE_USER_LONG | Long Pattern reserved for user application.<br><br>Available for SDK use. |

**Return**

Return true if function executes successfully.

### *BOOL HOTLEDFlash(int onSeconds, Brightness bright)*

**Description**

Function switch on the LED flash light for the specified time period.

**Parameters**

| | |
|---|---|
| **OnSeconds** | Duration specified in seconds (max 10 seconds) |
| **Bright** | Led brightness. See below table |

| | |
|---|---|
| BRIGHT_DIM | ¼ th of Full brightness |
| BRIGHT_HALF | 1/2$^{th}$ of Full Brightness |
| BRIGHT_FULL | Full Brightness |

**Return**

Return true if function executes successfully.

### *BOOL HOTLEDBackLight(int onSeconds,Brightness bright)*

**Description**

This function switches on the backlight with specified brightness

**Parameters**

| | |
|---|---|
| **OnSeconds** | Duration specified in seconds (max 20 seconds) |
| **Bright** | Led brightness. See below table |

| | |
|---|---|
| BRIGHT_DIM | ¼ th of Full brightness |
| BRIGHT_HALF | 1/2th of Full Brightness |
| BRIGHT_FULL | Full Brightness |

**Return**

Return true if function executes successfully.

## 4.2.1 OS Notifications Functions

### *BOOL HOTSetOSPingCallBack(int instID)*

**Description**

This function set up the call back function which OS will call every time it wakeup from the sleep. While in normal mode OS wakes up from sleep once in every minute. In fast refresh mode, the ping is called once a second. Depending on the OS mode, the ping function frequency changes. Application can use ping function if it needs to do some back ground functions.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |

**Callback Notification**

MessageSink function is called at the above mentioned frequency.

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_OSPING |
| param1 | Not used |
| param2 | Not used |
| buffer | Not used |

**Return**

Return true if function call successful.

### BOOL HOTSetTimerListner(int instID, int timerInterval, int repeat, int timerID)

**Description**

This function creates a low resolution timer.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |
| **timerInterval** | Timer interval in seconds |
| **repeat** | Timer called once if set to 0 or repeating if set to 1 |
| **timerID** | User assigned timer ID |

**Callback Notification**

**MessageSink** function is called at requested frequency to notify timer event.

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_TIMER |
| param1 | Not used |
| param2 | Not used |
| buffer | Not used |

**Return**

Return true if function executes successfully.

### BOOL HOTStopTimer( int instID,int timerID)

**Description**

This function stops the timer.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |
| **timerID** | User assigned timer ID |

**Return**

Return true if function executes successfully.

### BOOL HOTSubscribeFocusChangeEvent(int instID)

**Description**

Registers call back for app coming into focus. ON_FOCUS message is send to the application when user enters the application's screen. This same function is called once again when user leaves the screen.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |

**Callback Notification**

**MessageSink** function is called to notify focus change event.

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_ONFOCUS_CHANGE |
| param1 | 1- Application got focus<br>2- Application Lost focus |
| param2 | Not used |
| buffer | Not used |

**Return**

Return true if function executes successfully.

### *BOOL HOTSubscribeTouchEvents(int instID)*

**Description**

Registers a call back to receive screen touch events.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |

**Callback Notification**

**MessageSink** function is called to notify touch events

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_ONTOUCH |
| param1 | X: Co-ordinate of touch |
| param2 | Y: Co-ordinate of touch |
| buffer | Not used |

**Return**

Return true if function executes successfully.

### *BOOL HOTGetTouchPoints (int instID,int *numPoints,int *xPoints,int *yPoints)*

**Description**

This function can be used in conjunction with touch events to retrieve the touch pattern. Useful to create touch gusters.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |
| **numPoints** | Number of points available in OS buffer |
| **xPoints** | X Co-Ordinates of the points |
| **yPoints** | Y Co-Ordinates of the points |

**Return**

Return true if function executes successfully.

### *BOOL HOTUnSubscribeEvents(int instID,int mesgID)*

**Description**

Registers a call back to receive screen touch events.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |
| **mesgID** | ID of the mesg to unsubscribe. Only those message has subscribe functions can be un subscribed |

**Return**

Return true if function executes successfully.

## 4.2.1 Communication Channel Functions

The native apps can be standalone native apps or native apps that work with companion phone app. The functions in this section are used to setup a communication channel between phone app and the corresponding companion native app in the watch. These functions are used only when a companion native app needs to communicate with the phone app. This channel is designed as a raw data channel to allow the application to define own protocols and exchange any type of data.

### *BOOL HOTCreateComChanel(int instID ,char appName[8], int chanelID)*

**Description**

This function creates a communication channel between phone application and watch application. Communication end points are identified by “AppName” and “chanelID”.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |
| **appName** | A name used to identify the application end point. This field along with chanelID will make a unique end point |
| **chanelID** | A unique chanel ID used to identify the end point. This field along with appName will make a unique end point |

**Callback Notification**

When a communication channel is created, any data received from the remote end will be delivered to **MessageSink** callback with SWM_DATARECVD as the message id.

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_DATARECVD |
| param1 | Bytes received |
| param2 | |
| buffer | pointer to custom data |

**Return**

Return true if function call successful.

### *Int HOTWriteToChanel( int instID,int chanID, char * buffer, int len,void )*

**Description**

This function sends a packet of data to the registered end point.

**Parameters**

| | |
|---|---|
| **instID** | Instance ID |
| **chanelID** | A unique channel ID used to identify the end point. T |
| **buffer** | Data buffer to send |

| | |
|---|---|
| **len** | Length of the buffer. Max size is limited 200 bytes |

**Callback Notification**

MessageSink call back is called after sending the complete data.

| | |
|---|---|
| appID | Application ID |
| mesgID | SWM_DATASENT |
| param1 | Bytes send |
| param2 | Not used |
| buffer | Not used |

**Return**

Please see the table for the meaning of return values

| | |
|---|---|
| 1 | Function call successful |
| -1 | Chanel not opened |
| -2 | Previous data is not transmitted completely. Date not copied to the transmission buffer. |

# 5. Native SDK App Samples

### *Example 1: Ping Callback*

This example demonstrates a simple app which registers with OS for a ping call back. It displays a message box when ping is received.

**Header File**

App1.h

```
#include <stdio.h>

void main(int arg, char* args);

void MessageSink(int appID, int mesgID, int param1,int param2, char * buffer);
```

**Implementation File**

App.c

```
#include "app1.h"
#include "SwSdk.h"
#include <stdlib.h>
#include <string.h>

void main(int arg, char* args)
{
  int appID;

   HOTInitAppContext(&appID,"a",MessageSink);

   HOTSetOSPingCallBack(appID);
}

void MessageSink(int appID,int mesgID,int param1,int param2, char * buffer)

{
   switch(mesgID)
   {
        case SWM_OSPING:
              sprint(HOTGenString, "Os ping %d,%d\r\n", appID,mesgID)
              HOTDisplayMessageBox("PING", HOTGenString, 0);
              break;

   }
}
```

### *Example 2:*

Native Application With App Data Example

**Header File**

App2.H

```
#include <stdio.h>
#include <tchar.h>



struct MyPingCounter2
{
   int counter;
   char appName[10];
};

void main(int arg, char* args);
void MessageSink(int appID,int mesgID,int param1,int param2, char * buffer);
```

Implementation File

App.c

```
#include "app2.h"
#include "SwSdk.h"
#include <stdlib.h>
#include <string.h>

void main(int arg, char* args)
{
   int appID;

   MyPingCounter2 *appData =(MyPingCounter2 *) HOTMalloc(sizeof(MyPingCounter));

   HOTInitAppContext(&appID,"b",MessageSink);
   HOTSetOSPingCallBack(appID);
   appData[0].counter=0;
   strcpy(appData[0].appName,"App Two");
   HOTSetAppData(appID,(void *) appData);
}

void MessageSink(int appID,int mesgID,int param1,int param2, char * buffer)
{
   switch(mesgID)
   {
   case SWM_OSPING:
         MyPingCounter2 *counter;
         counter =(MyPingCounter2 *)  HOTGetAppData(appID);
         sprintf(HOTGenString, "Os ping Recv by App2: %s ,AppiD:%d MesgID: %d OSPingCounter :
```

```
%d\r\n",counter->appName ,appID,mesgID,counter->counter);
        HOTDisplayMessageBox("PING", HOTGenString, 0);

        counter->counter++;
        break;
  }
}
```

# 6. Android SDK Phone App Samples

### *Example 1: Sample Canvas App*

The Below sample app demonstrates the usage of Canvas SDK to display "Hello World" with two buttons.

```
public class MainActivity extends Activity {
  WatchCanvas canvas;
   Bitmap  bitMap;
   HOTDeviceContext hotContext;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    HOTSmartWatch hotSmartWatch = new HOTSmartWatch();
    try
    {
    hotContext = hotSmartWatch.OpenDevice(getApplicationContext(), "TestApp",
                  new HotWatchConListner());
    }
    catch(HOTNotRegistered exp)
    {

      hotSmartWatch.RegisterApp(getApplicationContext(), "TestApp",
                  HOTSmartWatch.AppType.APP_GUEST, 0);
    }
    bitMap = Bitmap.createBitmap(144,150,Bitmap.Config.ARGB_8888);
    canvas = new WatchCanvas (ScreenType.SCRN_LOWRESMONO,bitMap,hotContext);
    WatchCanvas.Button button = canvas.new  Button();
    Rect rect = new Rect(10, 10, 50, 30);
    button.Create(rect, "OK");
    ButtonOneLisner but1Clik = new ButtonOneLisner();
    button.setOnClickListner(but1Clik);
    WatchCanvas.Button buttonCan = canvas.new  Button();
    Rect rect2 = new Rect(100, 10, 140, 30);
```

```
    buttonCan.Create(rect2, "Cancel");
    ButtonTwoLisner but2Clik = new ButtonTwoLisner();
    button.setOnClickListner(but2Clik);

    String str = "Hello World";
    Paint brush = new Paint();
    brush.setColor(Color.BLACK);
    canvas.drawText(str, 10, 50, brush);
    canvas.AddUIObject(button);
    canvas.AddUIObject(buttonCan);
    canvas.DisplayCanvas(null, DisplayOption.DISP_UPDATENDISP);
}

class ImgViewClick implements View.OnTouchListener
{
    @Override
    public boolean onTouch(View arg0, MotionEvent arg1) {
        // TODO Auto-generated method stub
        if(arg1.getAction()==MotionEvent.ACTION_DOWN)
        {
        int x= (int)arg1.getX();
        int y = (int) arg1.getY();
        canvas.IsClicked(new Point(x, y));
        }
        return false;
    }
}
class ButtonOneLisner implements WatchCanvas.OnClickListner
{
    @Override
    public void OnClick(UIObject uiElement) {
        // TODO Auto-generated method stub
        WatchCanvas.Button  buttonClked = (WatchCanvas.Button)uiElement;
        buttonClked.captionText = "Touched";
        canvas.DisplayCanvas(null, DisplayOption.DISP_UPDATENDISP);
    }
}
class ButtonTwoLisner implements WatchCanvas.OnClickListner
{
    @Override
    public void OnClick(UIObject uiElement) {
        // TODO Auto-generated method stub
        WatchCanvas.Button  buttonClked = (WatchCanvas.Button)uiElement;
```

```
        buttonClked.captionText = "Touched";
       public class MainActivity extends Activity {
 WatchCanvas canvas;
 Bitmap  bitMap;
 HOTDeviceContext hotContext;
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   HOTSmartWatch hotSmartWatch = new HOTSmartWatch();
   try
   {
   hotContext = hotSmartWatch.OpenDevice(getApplicationContext(), "TestApp", new
                        HotWatchConListner());
   }
   catch(HOTNotRegistered exp)
   {
      hotSmartWatch.RegisterApp(getApplicationContext(), "TestApp",
                        HOTSmartWatch.AppType.APP_GUEST, 0);
   }
   bitMap = Bitmap.createBitmap(144,150,Bitmap.Config.ARGB_8888);
   canvas = new WatchCanvas (ScreenType.SCRN_LOWRESMONO,bitMap,hotContext);
   WatchCanvas.Button button = canvas.new  Button();
   Rect rect = new Rect(10, 10, 50, 30);
   button.Create(rect, "OK");
   ButtonOneLisner but1Clik = new ButtonOneLisner();
   button.setOnClickListner(but1Clik);
   WatchCanvas.Button buttonCan = canvas.new  Button();
   Rect rect2 = new Rect(100, 10, 140, 30);
   buttonCan.Create(rect2, "Cancel");
   ButtonTwoLisner but2Clik = new ButtonTwoLisner();
   button.setOnClickListner(but2Clik);
   String str = "Hello World";
   Paint brush = new Paint();
   brush.setColor(Color.BLACK);
   canvas.drawText(str, 10, 50, brush);
   canvas.AddUIObject(button);
   canvas.AddUIObject(buttonCan);
   canvas.DisplayCanvas(null, DisplayOption.DISP_UPDATENDISP);
}

class ImgViewClick implements View.OnTouchListener
{
```

```
    @Override
    public boolean onTouch(View arg0, MotionEvent arg1) {
        // TODO Auto-generated method stub
        if(arg1.getAction()==MotionEvent.ACTION_DOWN)
        {
        int x= (int)arg1.getX();
        int y = (int) arg1.getY();
        canvas.IsClicked(new Point(x, y));
        }
        return false;
    }
}
class ButtonOneLisner implements WatchCanvas.OnClickListner
{
    @Override
    public void OnClick(UIObject uiElement) {
        // TODO Auto-generated method stub
        WatchCanvas.Button  buttonClked = (WatchCanvas.Button)uiElement;
        buttonClked.captionText = "Touched";
        canvas.DisplayCanvas(null, DisplayOption.DISP_UPDATENDISP);
    }
}

class ButtonTwoLisner implements WatchCanvas.OnClickListner
{
    @Override
    public void OnClick(UIObject uiElement) {
        // TODO Auto-generated method stub
        WatchCanvas.Button  buttonClked = (WatchCanvas.Button)uiElement;
        buttonClked.captionText = "Touched";
        canvas.DisplayCanvas(null, DisplayOption.DISP_UPDATENDISP);
    }
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
}

class HotWatchConListner implements HOTSmartWatch.OnConStatusChangeListener
{
    @Override
    public void OnStatusChnage(BluetoothStatus status) {
```

```
        // TODO Auto-generated method stub

        }
    }
    }

    }

    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
    class HotWatchConListner implements HOTSmartWatch.OnConStatusChangeListener
    {
        @Override
        public void OnStatusChnage(BluetoothStatus status) {
            // TODO Auto-generated method stub
        }
    }
}
```

## 7. Conclusion

The HOT Smart watch SDK is currently under development. This is a preliminary document. There can be changes from the definition given here to the final implementation.