

Microsoft Band SDK

SDK Documentation

By using this Microsoft Band SDK, you agree to be bound by the [Terms of Use](#). Further, if accepting on behalf of a company, you represent that you're authorized to act on your company's behalf.

1	Introduction and Features	4
1.1	Multi-Platform Support	4
1.2	Getting Sensor Data	4
1.3	Creating Tiles and Sending Notifications	6
1.3.1	App Tiles	6
1.3.2	App Notifications	7
1.3.3	Haptic Notifications	7
1.3.4	Band Personalization	8
1.3.4.1	Me Tile	8
1.3.4.2	Color Themes	8
2	Getting Started	10
2.1	Android Requirements	10
2.2	iOS Requirements	10
2.3	Windows Requirements	10
2.3.1	Windows Store Applications Capabilities	10
2.3.2	Bluetooth Power Management Settings for Windows	11
3	Connecting to a Band	12
3.1	Android	12
3.2	iOS	13
3.3	Windows	13
4	Retrieving the Band Version Information	15
4.1	Android	15
4.2	iOS	15
4.3	Windows	16
5	Subscribing to Band Sensors	17
5.1	User Consent for Sensor Subscriptions	17
5.2	Subscribe to Heart Rate Sensor Stream	17
5.2.1	Android	17
5.2.2	iOS	18
5.2.3	Windows	20

5.3	Payload for Sensor Events.....	21
5.3.1	Accelerometer.....	21
5.3.2	Altimeter	21
5.3.3	AmbientLight	22
5.3.4	Barometer	22
5.3.5	Calories.....	22
5.3.6	Contact	23
5.3.6.1	enum BandContactState.....	23
5.3.7	Distance.....	23
5.3.7.1	enum MotionType	23
5.3.8	Gsr	24
5.3.9	Gyroscope.....	24
5.3.10	HeartRate	24
5.3.10.1	enum HeartRateQuality.....	24
5.3.11	Pedometer.....	25
5.3.12	RRInterval	25
5.3.13	SkinTemperature.....	25
5.3.14	UV.....	25
5.3.14.1	enum UVIndexLevel.....	25
6	Creating and Managing Tiles.....	26
6.1	Retrieving, Creating, and Removing Tiles.....	26
6.1.1	Android.....	26
6.1.2	iOS	28
6.1.3	Windows.....	29
7	Sending Notifications	32
7.1	Sending Dialogs	32
7.1.1	Android.....	32
7.1.2	iOS	32
7.1.3	Windows.....	32
7.2	Sending Messages	32
7.2.1	Android.....	32
7.2.2	iOS	33
7.2.3	Windows.....	33
7.3	Allowing Tiles to Receive Local and Remote Notifications (iOS apps only)	34
7.3.1	iOS	34
8	Customizing Tile Layouts	35
8.1	Creating a Layout for a Page	35
8.1.1	Primitive Element Types.....	35
8.1.2	Container Element Types	36
8.1.3	Example Layout Element Tree.....	36
8.1.4	Element Attributes	37
8.2	Element Rectangle and Margins	39
8.2.1	Negative Margins	39
8.3	Draw order for elements	39
8.4	Colors	40

8.5	Icons	40
8.5.1	Icons Used as FilledButton Masks	41
8.6	Barcodes	41
8.6.1	Supported character set for PDF417 barcodes	41
8.6.1.1	Microsoft Band 1	41
8.6.1.2	Microsoft Band 2	41
8.6.2	Supported character set for CODE39 barcodes.....	41
8.7	Setting the Contents of a Page	41
8.8	Simple Custom Tile Example	42
8.8.1	Android.....	42
8.8.2	iOS	44
8.8.3	Windows.....	47
9	<i>Handling Custom Tile Events.....</i>	51
9.1	Android.....	51
9.2	iOS.....	52
9.3	Windows.....	54
9.3.1	Tile Event Handling by Foreground App	54
9.3.2	Background Tile Event Handling (Universal Windows Platform)	57
9.3.2.1	Troubleshooting Background Tile Events	60
10	<i>Sending Haptics to the Band.....</i>	61
10.1	Android.....	61
10.2	iOS.....	61
10.3	Windows.....	61
11	<i>Band Personalization</i>	63
11.1	Managing the Me Tile Image.....	63
11.1.1	Android.....	63
11.1.2	iOS	63
11.1.3	Windows.....	64
11.2	Changing Theme.....	65
11.2.1	Android.....	65
11.2.2	iOS	65
11.2.3	Windows.....	66

1 INTRODUCTION AND FEATURES

The Microsoft Band SDK is designed to enable third-party application developers to harness the power of Microsoft Band. The SDK gives developers access to the sensors available on the Band, as well as the ability to create Tiles on the Band and to send notifications to these Tiles from their applications. Through the SDK, you, as an application developer, will be able to enhance and extend the experience of your application to your customers' wrists. This opens up a whole new dimension of interaction and enables new, richer scenarios for your applications that make use of the capabilities of the Band.

Applications that work with Microsoft Band make use of the Microsoft Band SDK to communicate with the Band. The application logic runs on the host OS (iPhone, Android, Windows), and remotely controls the UI of the Band as well as receives contextual information and sensor data from the Band.

The features offered by the Microsoft Band SDK are as follows:

- Multi-platform support
- Sensor data subscriptions
- Tile creation and management
- Tile notifications
- Custom layouts
- Haptic notifications
- Band theme personalization

The following subsections describe these features in more detail.

1.1 MULTI-PLATFORM SUPPORT

The SDK is also supported on a wide range of platforms. They include:

- Windows Phone 8.1 and later
- Windows 8.1 and later (Store Apps)
- iOS 7 and later
- Android 4.2 (API 17) and later

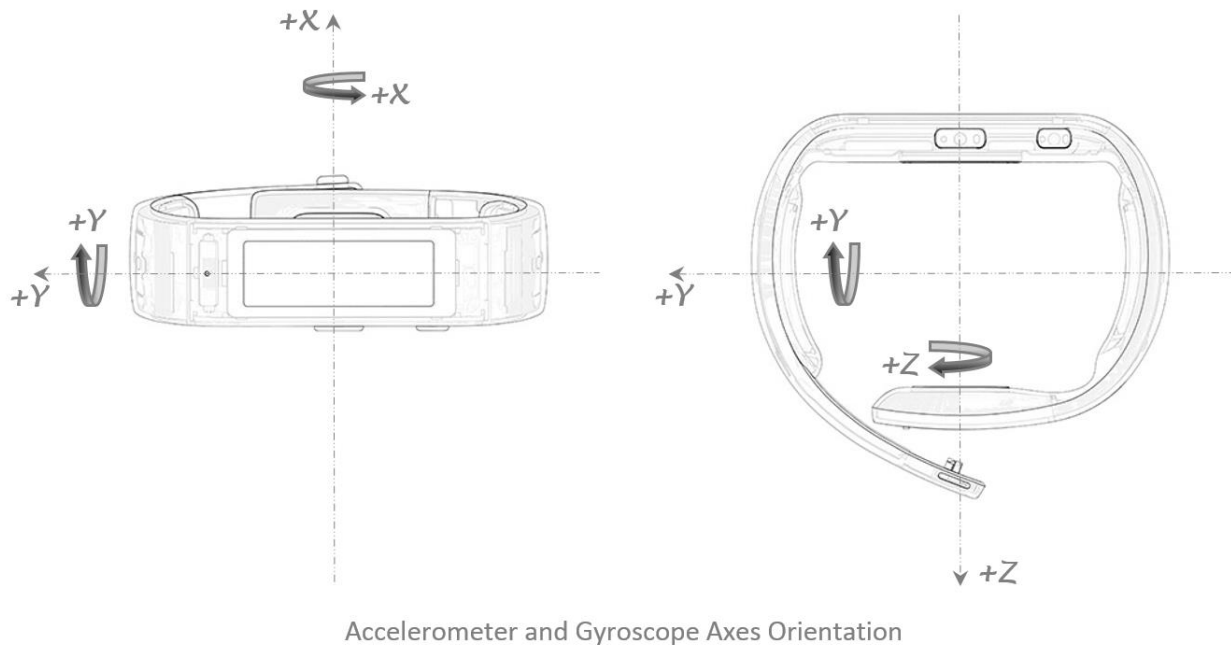
1.2 GETTING SENSOR DATA

Microsoft Band features many hardware sensors that application developers can get data from. The SDK exposes data from these sensors as streams, and applications can subscribe to these sensor streams. Here's a list of all the sensor streams that are available and the frequencies at which data can be retrieved from them. In case of multiple supported frequencies, only the highest rate is available for iOS.

Sensor Stream	Details	Frequency
Accelerometer	Provides X, Y, and Z acceleration in g units. 1 g = 9.81 meters per second squared (m/s ²).	62/31/8 Hz
Gyroscope	Provides X, Y, and Z angular velocity in degrees per second (°/sec) units.	62/31/8 Hz
Distance	Provides the total distance in centimeters, current speed in centimeters per second (cm/s), current pace in milliseconds per meter (ms/m), and the current	1 Hz

	pedometer mode (such as walking or running).	
Heart Rate	Provides the number of beats per minute; also indicates if the heart rate sensor is fully locked on to the wearer's heart rate. The data returned should be used only in resting mode. The SDK doesn't provide access to the heart rate values optimized for any other activity.	1 Hz
Pedometer	Provides the total number of steps the wearer has taken since the Band was last factory-reset. This is a lifetime counter and not a daily or a 0-based counter. To determine the absolute number of steps between two readings, you must take the difference between the returned values.	Value change
Skin Temperature	Provides the current skin temperature of the wearer in degrees Celsius.	1 Hz
UV	Provides the current ultraviolet radiation exposure intensity.	1 Hz
Band Contact	Provides the current state of the Band as being worn/not worn.	Value change
Calories	Provides the total number of calories the wearer has burned since the Band was last factory-reset. This is a lifetime counter and not a daily or a 0-based counter. To determine the absolute number of calories burned between two readings, you must take the difference between the returned values.	Value change
Galvanic Skin Response	(Microsoft Band 2 only) Provides the current skin resistance of the wearer in kohms.	0.2 Hz
RR Interval	(Microsoft Band 2 only) Provides the interval in seconds between the last two continuous heart beats. The data returned should be used only in resting mode. The SDK doesn't provide access to the RR interval values optimized for any other activity.	Value change
Ambient Light	(Microsoft Band 2 only) Provides the current light intensity (illuminance) in lux (Lumes per sq. meter).	2 Hz
Barometer	(Microsoft Band 2 only) Provides the current raw air pressure in hPa (hectopascals) and raw temperature in degrees Celsius.	1 Hz
Altimeter	(Microsoft Band 2 only) Provides current elevation data like total gain/loss, steps ascended/descended, flights ascended/descended, and elevation rate.	1 Hz

The following images show the axes for the accelerometer and gyroscope sensor relative to the Band. This is applicable to both Microsoft Band 1 and Microsoft Band 2.



Refer to [detailed section on subscribing to Band sensors](#) to learn how to create and manage heart rate subscription on the Band.

1.3 CREATING TILES AND SENDING NOTIFICATIONS

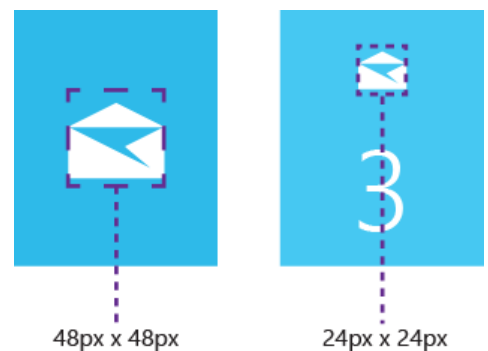
The SDK offers developers the option of creating tiles on the Band specific to their application. A developer can create one or more tiles to send notifications to the Band to let the user know something interesting has occurred. Tiles support custom icons and color themes.

1.3.1 App Tiles

Each app tile is visually represented on the Start Strip by an icon that fits within a 48px x 48px box for Microsoft Band 2 and 46px x 46px box for Microsoft Band 1.

Applications can create two types of tiles on the Band: messaging tiles and custom tiles.

Messaging tiles contain simple messages with a title and a body as their content. These tiles will also show a badge count when new content arrives. When badged, the tile uses a 24px x 24px icon to accommodate the badging system. If your app requires both sizes, you may need to redraw your icon to make sure it is visually optimized for the smaller size. Refer to the [detailed section on creating and managing tiles](#) to learn how to create your own tiles on the Band.



Custom tiles have application-defined layouts and custom content, which includes multiple icons, buttons, text blocks, and barcodes. With custom tiles, developers can define unique experiences for their applications. Developers control exactly how many pages to show inside a tile as well as the content of individual pages. They can update the contents of a page that has been created by using custom layout at any point. This is unlike messaging tiles where every new message results in the creation of a new page inside the tile. In addition, a

developer can choose to add more pages inside the tile. If the total number of pages goes past the maximum pages allowed inside the tile, the right-most page is dropped out when a new page is added.

It's also possible to register for tile events by using the SDK. This enables a developer to know when the user has entered and exited their tile. In addition, they can receive events when a user taps a button in one of their custom tiles. Refer to the [detailed section on custom layouts](#) to learn how to create and use custom layouts and how to handle event callbacks from the Band tiles.

1.3.2 App Notifications

App notifications come in two flavors:

- **Dialogs** – Dialog notifications are pop-ups that are meant to quickly display information to the user. When the user dismisses the dialog, the information it contains doesn't persist on the Band.
- **Messages** – Message notifications are sent and stored in a specific tile, and a tile can keep up to 8 messages at a time. Messages can display a dialog, too. Messages queue up inside the tile in a first-in, first-out fashion. Every new message is placed at the head of the queue (left-most position inside the tile). After the number of messages exceeds the maximum allowed for a tile, the right-most message is dropped out of the queue and replaced with the new incoming message.

Both notification types contain a title text and a body text. Refer to the [detailed section on app notifications](#) for more information on how to create and send these notifications.

1.3.3 Haptic Notifications

App notification dialogs are accompanied with a stock vibration tone on the Band. However, it's also possible to send haptic notifications to the Band without any accompanying UI notification. These haptic notifications can be used to inform the user of checkpoints or simply provide feedback for actions taken by the user. There are 9 predefined vibration tones that you can send to the Band.

Vibration Tone	Description
Notification one tone	One gentle notification tone
Notification two tone	Two gentle notification tones
Notification alarm	Three long, high-intensity tones
Notification timer	One long, high-intensity tone
One tone high	One high-intensity tone
Two tone high	Two high-intensity tones
Three tone high	Three high-intensity tones
Ramp up	One tone with ascending intensity
Ramp down	One tone with descending intensity

Note that through device settings on the Band, users have control over the intensity of haptic levels. Users can also choose to turn the haptic motor completely off. In those cases, the Band will respect the device setting and

not vibrate. Refer to the [detailed section on haptic notifications](#) to see how to create and send haptic notifications to the Band.

1.3.4 Band Personalization

In addition to creating tiles for their own applications, developers can use the SDK to further customize a user's Band by setting the background image for the Me Tile, as well as setting the default color theme for all tiles on the Band.

1.3.4.1 Me Tile












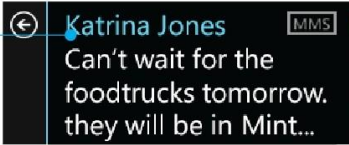











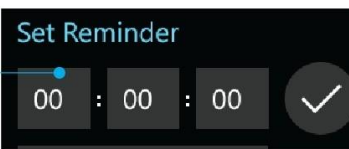
The Me Tile is the default view of the Start Strip and is the tile that contains the user's Steps, Calories, and Heart Rate. The resolution of the Me Tile background image for Microsoft Band 1 is 310x102 pixels.

Microsoft Band 2 has a different native resolution of the Me Tile image. The new resolution is 310x128 pixels. However, Microsoft Band 2 is backward-compatible with Microsoft Band 1 image sizes. If an image of 310x102 pixels is sent to Microsoft Band 2, the image will be vertically centered and the empty space will be colored in as solid black to fill up the Me Tile.



1.3.4.2 Color Themes

For each color theme, there are eight color classes that are used to represent various states of tile activity on the Band. The following graphic shows the various places where different colors show up on the Band. Band color theme applies to all tiles. The SDK lets developers override six of the eight colors of the theme. Medium and Buttons are two colors that aren't customizable via the SDK by using color themes.

Tiles	 Base	Start strip tile App Bar	 Base	
	 High Contrast	Start strip Tile Background (New content)	 High Contrast	
	 Lowlight	Down-feedback on start strip tiles Down-feedback on buttons	 Lowlight	
Theme	 Highlight	Header	 Highlight	
	 Medium	Toggle on state	 Medium	
	 Muted	Fitness achievement Marker backgrounds	 Muted	
System	 Secondary	System wide secondary text color	 Secondary	
	 Buttons	System wide grey color	 Secondary	

Refer to the [detailed section on Band personalization](#) to find out how to create custom wallpapers and color themes for the Band.

2 GETTING STARTED

For each platform, there are a few requirements before using the Microsoft Band SDK.

2.1 ANDROID REQUIREMENTS

The minimum-supported Android API version is 17. The following uses-permission tags should be added to the AndroidManifest.xml:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>

<uses-permission android:name="com.microsoft.band.service.access.BIND_BAND_SERVICE" />
```

The Android SDK relies on capabilities provided by the Microsoft Health App. To use the SDK on Android, having the Microsoft Health App installed on the phone is a requirement. Without the Microsoft Health App installed on the phone, any attempts to create a Microsoft Band client will fail with an error.

2.2 IOS REQUIREMENTS

The minimum-supported iOS version is 7.0. The recommended development environment is Xcode 6.0+. The SDK doesn't support iOS simulator. If you want to develop an application to communicate with the Band in the background, you must enable "Use Bluetooth LE accessories" in Background modes.

2.3 WINDOWS REQUIREMENTS

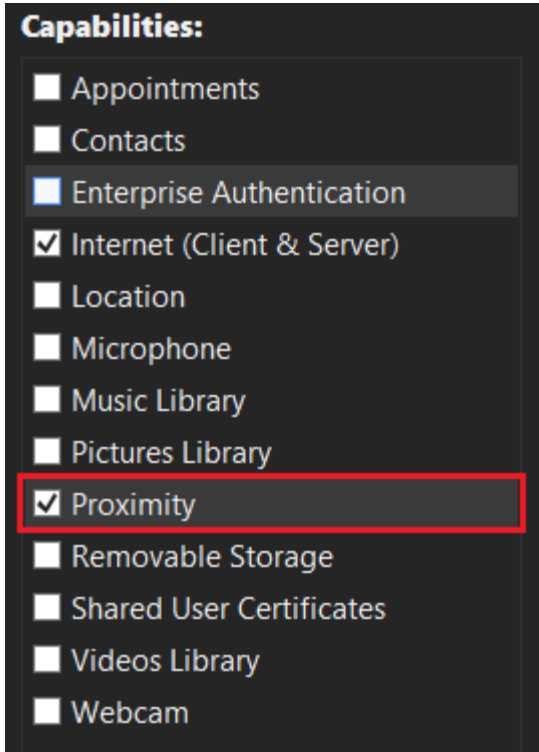
All versions of the Microsoft Band SDK for Windows platforms require Visual Studio 2013 or later. If you want to use the SDK with Windows Phone, you'll also need to install the Windows Phone SDK add-on for Visual Studio.

2.3.1 Windows Store Applications Capabilities

If you're writing a Windows RT application for Windows or Windows Phone, the application will need to declare the appropriate privileges that are needed to access Bluetooth hardware resources. To do this, manually update the Capabilities section of the application's Package.appxmanifest file to include the following:

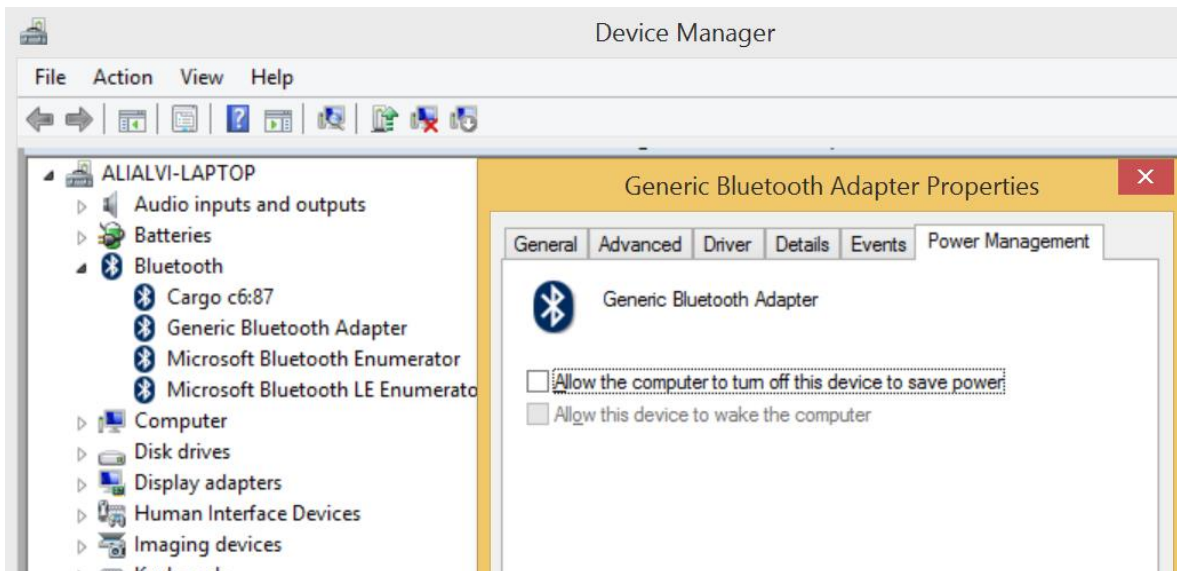
```
<m2:DeviceCapability Name="bluetooth.rfcomm">
  <m2:Device Id="any">
    <!-- Used by the Microsoft Band SDK -->
    <m2:Function Type="serviceId:A502CA9A-2BA5-413C-A4E0-13804E47B38F" />
    <!-- Used by the Microsoft Band SDK -->
    <m2:Function Type="serviceId:C742E1A2-6320-5ABC-9643-D206C677E580" />
  </m2:Device>
</m2:DeviceCapability>
```

You will also need to add the Proximity capability, but you can do that from the manifest editor UI.



2.3.2 Bluetooth Power Management Settings for Windows

One of the limitations of using the Microsoft Band SDK on Windows is that the default Power Management setting for Bluetooth devices allows Windows to turn the device off to save power. This can cause problems if power-saving is turned on in the middle of Bluetooth communication between a Band and a PC. To work around this situation, users can turn Power Management off by clearing the Allow the computer to turn off this device to save power check box in the Properties dialog box for the Bluetooth device that they're using with their PC.



3 CONNECTING TO A BAND

The first step in using the SDK is to make a connection to a Band. To make a connection, the Band and the device that your application is running on must be paired with each other. You can use the Band Client Manager to get a list of paired Bands, and establish a connection to one or more paired Bands. Operations to the Band are encapsulated in a Band Client.

3.1 ANDROID

1. Import the appropriate packages.

```
import com.microsoft.band.BandClient;
import com.microsoft.band.BandClientManager;
import com.microsoft.band.BandException;
import com.microsoft.band.BandInfo;
import com.microsoft.band.BandIOException;
import com.microsoft.band.ConnectionState;
```

2. Get a list of paired Bands.

```
BandInfo[] pairedBands =
BandClientManager.getInstance().getPairedBands();
```

3. Create a BandClient object.

```
BandClient bandClient =
BandClientManager.getInstance().create(getActivity(), pairedBands[0]);
```

4. Connect to the Band.

```
// Note: The BandPendingResult.await() method must be called from a
background thread. An exception will be thrown if called from the UI
thread.
BandPendingResult<ConnectionState> pendingResult =
bandClient.connect();
try {
    ConnectionState state = pendingResult.await();
    if(state == ConnectionState.CONNECTED) {
        // do work on success
    } else {
        // do work on failure
    }
} catch(InterruptedException ex) {
    // handle InterruptedException
} catch(BandException ex) {
    // handle BandException
}
```

3.2 iOS

1. Import the appropriate packages.

```
#import <MicrosoftBandKit_iOS/MicrosoftBandKit_iOS.h>
```

2. Set up ClientManager and its delegate.

```
[[MSBClientManager sharedManager] setDelegate:self];
```

3. Get a list of attached Clients.

```
NSArray *attachedClients = [[MSBClientManager sharedManager]
attachedClients];
```

4. Connect to the Band Client.

```
MSBClient *client = [attachedClients firstObject];
if (client)
{
    [[MSBClientManager sharedManager] connectClient:client];
}
```

5. Implement appropriate delegate methods.

```
// Note: The delegate methods of MSBClientManagerDelegate protocol are
called in the main thread.
-(void)clientManager:(MSBClientManager *)cm
clientDidConnect:(MSBClient *)client
{
    // handle connected event
}
-(void)clientManager:(MSBClientManager *)cm
clientDidDisconnect:(MSBClient *)client
{
    // handle disconnected event
}
-(void)clientManager:(MSBClientManager *)cm client:(MSBClient *)client
didFailToConnectWithError:(NSError *)error
{
    // handle failure event
}
```

3.3 WINDOWS

1. Set up using directives.

```
using Microsoft.Band;
```

2. Get a list of paired Bands.

```
IBandInfo[] pairedBands = await
BandClientManager.Instance.GetBandsAsync();
```

3. Connect to the Band to get a new BandClient object.

```
try
{
    using (IBandClient bandClient = await
BandClientManager.Instance.ConnectAsync(pairedBands[0]))
    {
        // do work after successful connect
    }
}
catch (BandException ex)
{
    // handle a Band connection exception
}
```

4 RETRIEVING THE BAND VERSION INFORMATION

Sometimes it's necessary to know what version of the Band you're communicating with. The Band Client allows you to query the Band for the current versions of both firmware and hardware. With this information, developers can create an application that offers different features based on the version of the connected Band.

Microsoft Band 1 is represented by hardware version string "19" or lower, whereas Microsoft Band 2 is represented by Hardware version string "20" or higher.

4.1 ANDROID

```
String fwVersion = null;
String hwVersion = null;
try {
    fwVersion = bandClient.getFirmwareVersion().await();
    hwVersion = bandClient.getHardwareVersion().await();

} catch (InterruptedException ex) {
    // handle InterruptedException
} catch (BandIOException ex) {
    // handle BandIOException
} catch (BandException ex) {
    // handle BandException
}
```

4.2 IOS

```
[self.client firmwareVersionWithCompletionHandler:^(NSString *version,
NSError *error){
    if (error)
    {
        // handle error
    }
    else
    {
        // handle success
    }
}];

[self.client hardwareVersionWithCompletionHandler:^(NSString *version,
NSError *error){
    if (error)
    {
        // handle error
    }
    else
    {
```

```
        // handle success
    }
}];
```

4.3 WINDOWS

```
string fwVersion;
string hwVersion;
try
{
    fwVersion = await bandClient.GetFirmwareVersionAsync();
    hwVersion = await bandClient.GetHardwareVersionAsync();

    // do work with firmware & hardware versions
}
catch(BandException ex)
{
    // handle any BandExceptions
}
```


5 SUBSCRIBING TO BAND SENSORS

The SDK provides support for Band sensors as subscriptions. The subscriptions are managed by the Band Sensor Manager on the Band Client. For each hardware sensor, the Sensor Manager allows the application developer to create a subscription. A subscription is essentially a platform-specific callback mechanism. It will deliver data at intervals specific to the sensor. Some sensors have dynamic intervals, such as the Accelerometer (on Android and Windows), that allow developers to specify at what rate they want data to be delivered. Other sensors deliver data only as their values change.

It's important to understand that subscribing to sensor data effects the battery life of the Band. The use of each sensor requires a power draw (some more than others). Developers should subscribe to sensor data only when the data is absolutely necessary for their applications.

On Windows and iOS, constant connectivity is required to maintain a subscription. If the Band loses connectivity with the phone, the subscription is stopped and it's not automatically enabled upon reconnection.

5.1 USER CONSENT FOR SENSOR SUBSCRIPTIONS

Some sensor subscriptions require user consent. The subscription permission model is as follows.

1. Permission is granted on a per-sensor basis.
2. Applications can request the permission status of a particular sensor. The status can be Granted, Declined, or Not Specified. If permission is Granted, applications can simply start the subscription.
3. Applications can request to show the permission dialog to ask the user for permission if the permission is Not Specified or Declined.
4. If the permission is Not Specified or Declined and the application requests that the subscription be enabled the subscription, the request to enable the subscription will fail.

The SDK shows the consent dialog in the context of the running application to acquire permission for the sensor subscription. The answer is remembered, so if the user has given permission once, the application wouldn't need to request it again.

Note: At this time, only heart rate and RR Interval sensor subscriptions require an explicit user consent before they can be started.

5.2 SUBSCRIBE TO HEART RATE SENSOR STREAM

5.2.1 Android

1. Update imports.

```
import com.microsoft.band.sensors.BandHeartRateEvent;
import com.microsoft.band.sensors.BandHeartRateEventListener;
import com.microsoft.band.sensors.HeartRateConsentListener;
```

2. Implement HeartRateConsentListener interface.

```
@Override
public void userAccepted(boolean consentGiven) {
    // handle user's heart rate consent decision
};
```

3. Ensure user has consented to heart rate sensor streaming.

```
// check current user heart rate consent
if(client.getSensorManager().getCurrentHeartRateConsent() !=
UserConsent.GRANTED) {
    // user hasn't consented, request consent
    // the calling class is an Activity and implements
    // HeartRateConsentListener
    bandClient.getSensorManager().requestHeartRateConsent(this,
this);
}
```

4. Create an event listener.

```
// create a heart rate event listener
BandHeartRateEventListener heartRateListener = new
BandHeartRateEventListener() {
    @Override
    public void onBandHeartRateChanged(BandHeartRateEvent event) {
        // do work on heart rate changed (i.e., update UI)
    }
};
```

5. Register the event listener.

```
try {
    // register the listener
    bandClient.getBandSensorManager().registerHeartRateEventListener(
heartRateListener);
} catch(BandIOException ex) {
    // handle BandException
}
```

6. Unregister the event listener.

```
try {
    // unregister the listener
    bandClient.getBandSensorManager().unregisterHeartRateEventListene
r(heartRateListener);
} catch(BandIOException ex) {
    // handle BandException
}
```

5.2.2 iOS

1. Verify user consent.

```
MSBUserConsent consent = [self.client.sensorManager
heartRateUserConsent];
switch (consent)
{
```

```

        case MSBUserConsentGranted:
            // user has granted access
            [self startHeartRateUpdates];
            break;
        case MSBUserConsentNotSpecified:
            // request user consent
            [self.client.sensorManager
requestHRUserConsentWithCompletion:^(BOOL userConsent, NSError *error)
{
                if (userConsent)
                {
                    // user granted access
                }
                else
                {
                    // user declined access
                }
            }];
            break;
        case MSBUserConsentDeclined:
            // user has declined access
            break;
        default:
            break;
    }
}

```

2. Start sensor updates.

```

- (void)startHeartRateUpdates
{
    // if Queue is nil, it uses default mainQueue
    NSError *subscriptionError;
    [self.client.sensorManager
startHeartRateUpdatesToQueue:nil
                        errorRef:&subscriptionError
                        withHandler:^(MSBSensorHeartRateData
*heartRateData, NSError *error)
    {
        if (error){
            // handle error
        }
    }];

    if (subscriptionError){
        // failed to subscribe
    }
}

```

3. Stop sensor updates.

```
[self.client.sensorManager
stopHeartRateUpdatesErrorRef:&subscriptionError];
if (subscriptionError){
    // failed to unsubscribe
}
```

5.2.3 Windows

1. Update using directives.

```
using Microsoft.Band.Sensors;
```

2. Ensure user has consented to heart rate sensor streaming.

```
// check current user heart rate consent
if (bandClient.SensorManager.HeartRate.GetCurrentUserConsent() !=
UserConsent.Granted)
{
    // user hasn't consented, request consent
    await
bandClient.SensorManager.HeartRate.RequestUserConsentAsync();
}
```

3. Get a list of supported intervals.

```
// get a list of available reporting intervals
IEnumerable<TimeSpan> supportedHeartBeatReportingIntervals =
bandClient.SensorManager.HeartRate.SupportedReportingIntervals;
foreach (var ri in supportedHeartBeatReportingIntervals)
{
    // do work with each reporting interval (i.e., add them to a list
in the UI)
}
```

4. Set the reporting interval (optional).

```
// set the reporting interval
bandClient.SensorManager.HeartRate.ReportingInterval =
supportedHeartBeatReportingIntervals.GetEnumerator().Current;
```

5. Subscribe to the reading changed event for the sensor.

```
// hook up to the Heartrate sensor ReadingChanged event
bandClient.SensorManager.HeartRate.ReadingChanged += (sender, args) =>
{
    // do work when the reading changes (i.e., update a UI element)
};
```

6. Start the sensor.

```
// start the Heartrate sensor
```

```

try
{
    await bandClient.SensorManager.HeartRate.StartReadingsAsync();
}
catch (BandException ex)
{
    // handle a Band connection exception
    throw ex;
}

```

7. Stop the sensor.

```

// stop the Heartrate sensor
try
{
    await bandClient.SensorManager.HeartRate.StopReadingsAsync();
}
catch (BandException ex)
{
    // handle a Band connection exception
    throw ex;
}

```

5.3 PAYLOAD FOR SENSOR EVENTS

Here are the values returned in the payloads for various sensor subscriptions exposed by the SDK. The property names and types might differ slightly between platforms. The following tables reflect the values on Windows.

5.3.1 Accelerometer

Type	Property and description
double	AccelerationX The x-axis acceleration of the Band in g units (9.81 m/s ²)
double	AccelerationY The y-axis acceleration of the Band in g units (9.81 m/s ²)
double	AccelerationZ The z-axis acceleration of the Band in g units (9.81 m/s ²)

5.3.2 Altimeter

Type	Property and description
long	FlightsAscended Number of floors ascended since the Band was last factory-reset

long	FlightsDescended Number of floors ascended since the Band was last factory-reset
float	Rate The current rate of ascend/descend in cm/s
long	SteppingGain Total elevation gained in centimeters by taking steps since the Band was last factory-reset
long	SteppingLoss Total elevation lost in centimeters by taking steps since the Band was last factory-reset
long	StepsAscended Total number of steps ascended since the Band was last factory-reset
long	StepsDescended Total number of steps descended since the Band was last factory-reset
long	TotalGain Total elevation gained in centimeters since the Band was last factory-reset
long	TotalLoss Total elevation loss in centimeters since the Band was last factory-reset

5.3.3 AmbientLight

Type	Property and description
int	Brightness Current ambient light in lumens per square meter (lux)

5.3.4 Barometer

Type	Property and description
double	AirPressure Current air pressure in hectopascals
double	Temperature Current temperature in degrees Celsius

5.3.5 Calories

Type	Property and description
long	Calories

	Number of kilocalories (kcal) burned since the Band was last factory-reset
--	--

5.3.6 Contact

Type	Property and description
BandContactState	State Current contact state of the Band

5.3.6.1 enum BandContactState

Value and description
NotWorn Value representing that the Band is currently not being worn
Unknown Value representing that the contact state of the Band is unknown
Worn Value representing that the Band is currently being worn

5.3.7 Distance

Type	Property and description
MotionType	CurrentMotion The current MotionType of the Band
float	Pace Current pace of the Band in ms/m
float	Speed Current speed of the Band in cm/s
long	TotalDistance Distance traveled in cm since the Band was last factory-reset

5.3.7.1 enum MotionType

Value and description
Unknown Value representing that the mode of the Band user's movement is currently unknown
Idle Value representing that the user of the Band is currently idle

Walking
Value representing that the user of the Band is currently walking
Jogging
Value representing that the user of the Band is currently jogging
Running
Value representing that the user of the Band is currently running

5.3.8 Gsr

Type	Property and description
int	Resistance Current skin resistance in kohms of the person wearing the Band

5.3.9 Gyroscope

Type	Property and description
double	AngularVelocityX Angular velocity around the x-axis of the Band in degrees/sec
double	AngularVelocityY Angular velocity around the y-axis of the Band in degrees/sec
double	AngularVelocityZ Angular velocity around the z-axis of the Band in degrees/sec

5.3.10 HeartRate

Type	Property and description
int	HeartRate Current heart rate as read by the Band in beats/min
HeartRateQuality	Quality Quality of the current heart rate reading

5.3.10.1 enum HeartRateQuality

Value and description
Acquiring Value representing that the Band hasn't locked on to its user's heart rate
Locked

Value representing that the Band has locked on to its user's heart rate

5.3.11 Pedometer

Type	Property and description
long	TotalSteps Total number of steps taken since the Band was last factory-reset

5.3.12 RRInterval

Type	Property and description
double	Interval Current RR interval in seconds as read by the Band

5.3.13 SkinTemperature

Type	Property and description
double	Temperature Current temperature in degrees Celsius of the person wearing the Band

5.3.14 UV

Type	Property and description
UVIndexLevel	IndexLevel Current UVIndexLevel value as calculated by the Band

5.3.14.1 enum UVIndexLevel

Value and description
None Value representing a very low UV index level
Low Value representing a low UV index level
Medium Value representing a medium UV index level
High Value representing a high UV index level
VeryHigh Value representing a very high UV index level

6 CREATING AND MANAGING TILES

Tiles offer the developer the ability to create application-specific experiences on the Microsoft Band. The Band supports up to 13 separate tiles. The Microsoft Band SDK will allow the application to create as many tiles as there is space for. To work with tiles, developers use the Band Tile Manager on the Band Client. The Tile Manager enables you to:

- Get a list of the application's tiles currently on the Band
- Get the number of available tile slots on the Band
- Create a tile
- Remove a tile

The tile can include an icon (46x46 pixels for Microsoft Band 1, 48x48 pixels for Microsoft Band 2), a small icon (24x24 pixels), and a title or name for the tile. All tiles will use the theme colors of the Band. Using the Tile Manager, the developer can override that theme.

The Band doesn't support colored icons, only white alpha-blended icons. Each platform has a helper method to convert a native bitmap object into the appropriate supported icon format. This helper method uses the alpha component of the native bitmap object pixels to construct a Band icon object. The red, green, and blue values of the native bitmap pixels aren't used.

After creating a tile, the developer can send dialogs and messages to them by using the Band Notification Manager on the Band Client. Both contain title and body strings, and the message includes a time stamp for display-ordering. Dialogs are quick pop-up windows that the user can dismiss. The information in the dialogs doesn't persist after the user has dismissed them. Messages can be sent with an option to show or not show a dialog. If chosen to be sent without the option to show a dialog, the user will see only the badge count updated on the tile. Regardless of whether sent with or without the option to show a dialog, the information of the last 8 messages persists and can be viewed by opening the tile that the message was sent to. Tiles also support badging, or showing the count of unseen messages, on the face of the tile.

6.1 RETRIEVING, CREATING, AND REMOVING TILES

6.1.1 Android

1. Update imports.

```
import java.util.List;
import java.util.UUID;
import com.microsoft.band.tiles.BandIcon;
import com.microsoft.band.tiles.BandTile;
```

2. Retrieve the list of your application's tiles already on the Band.

```
try {
    // get the current set of tiles
    List<BandTile> tiles =
bandClient.getBandTileManager().getTiles().await();
} catch (BandException e) {
    // handle BandException
} catch (InterruptedException e) {
```

```
// handle InterruptedException
}
```

3. Determine if there is space for more tiles on the Band.

```
try {
    // determine the number of available tile slots on the Band
    int tileCapacity =
bandClient.getBandTileManager().getRemainingTileCapacity().await();
} catch (BandException e) {
    // handle BandException
} catch (InterruptedException e) {
    // handle InterruptedException
}
```

4. Create a new tile.

```
// Create the small and tile icons from writable bitmaps.
// Small icons are 24x24 pixels.
Bitmap smallIconBitmap = Bitmap.createBitmap(24, 24, null);
BandIcon smallIcon = BandIcon.toBandIcon(smallIconBitmap);
// Tile icons are 46x46 pixels for Microsoft Band 1 and 48x48 pixels
// for Microsoft Band 2.
Bitmap tileIconBitmap = Bitmap.createBitmap(46, 46, null);
BandIcon tileIcon = BandIcon.toBandIcon(tileIconBitmap);

// create a new UUID for the tile
UUID tileUuid = UUID.randomUUID();

// create a new BandTile using the builder
// add optional small icon
// enable badging (the count of unread messages)
BandTile tile = new BandTile.Builder(tileUuid, "TileName", tileIcon)
    .setTileSmallIcon(smallIcon).setBadgingEnabled(true).build();

tile.IsBadgingEnabled = true;

try {
    if(bandClient.getBandTileManager().addTile(getActivity(),
tile).await()) {
        // do work if the tile was successfully created
    }
} catch (BandException e) {
    // handle BandException
} catch (InterruptedException e) {
    // handle InterruptedException
}
```

5. Remove all of the application's tiles.

```

try {
    // get the current set of tiles
    List<BandTile> tiles =
bandClient.getBandTileManager().getTiles().await();
    for(BandTile t : tiles) {
        if(bandClient.getBandTileManager().removeTile(t).await()){
            // do work if the tile was successfully removed
        }
    }
} catch (BandException e) {
    // handle BandException
} catch (InterruptedException e) {
    // handle InterruptedException
}

```

6.1.2 iOS

1. Retrieve the list of your application's tiles already on the Band.

```

[self.client.tileManager tilesWithCompletionHandler:^(NSArray *tiles,
NSError *error) {
    if (error){
        // handle error
    }
}];

```

2. Determine if there is space for more tiles on the Band.

```

[self.client.tileManager
remainingTileCapacityWithCompletionHandler:^(NSUInteger
remainingCapacity, NSError *error){
    if (error){
        // handle error
    }
}];

```

3. Create a new tile

```

// create the small and tile icons from UIImage
// small icons are 24x24 pixels
NSError *error;
MSBIcon *smallIcon = [MSBIcon iconWithUIImage:[UIImage
imageWithContentsOfFile:@"small.png"] error:&error];
if (error){
    // handle error
}
// Tile icons are 46x46 pixels for Microsoft Band 1 and 48x48 pixels
// for Microsoft Band 2.
MSBIcon *tileIcon = [MSBIcon iconWithUIImage:[UIImage
imageWithContentsOfFile:@"tile.png"] error:&error];

```

```

if (error){
    // handle error
}
// Sample code uses random tileId, but you should persist the value
for your application's tileId.
NSUUID *tileId = [NSUUID UUID];

// create a new MSBTile
MSBTile *tile = [MSBTile tileWithId:tileId name:@"sample"
tileIcon:tileIcon smallIcon:smallIcon error:&error];
if (error){
    // handle error
}
// enable badging (the count of unread messages)
tile.badgingEnabled = YES;

// add created tile to the Band.
[self.client.tileManager addTile:tile completionHandler:^(NSError
*error){
    if (error){
        // add tile failed, handle error
    }
}];
}];

```

4. Remove all the application's tiles.

```

// get the current set of tiles
__weak typeof(self) weakSelf = self;
[self.client.tileManager tilesWithCompletionHandler:^(NSArray *tiles,
NSError *error){
    for (MSBTile *aTile in tiles)
    {
        // remove this tile, can be async
        [weakSelf.client.tileManager removeTile:aTile
completionHandler:^(NSError *error){
            if (error) {
                // failed to remove this tile
            }
        }];
    }
}];
}];

```

6.1.3 Windows

1. Update using directives.

```
using Microsoft.Band.Tiles;
```

2. Retrieve the list of your application's tiles already on the Band.

```

try
{
    // get the current set of tiles
    IEnumerable<BandTile> tiles = await
bandClient.TileManager.GetTilesAsync();
}
catch (BandException ex)
{
    // handle a Band connection exception
}

```

3. Determine if there is space for more tiles on the Band.

```

try
{
    // determine the number of available tile slots on the Band
    int tileCapacity = await
bandClient.TileManager.GetRemainingTileCapacityAsync();
}
catch (BandException ex)
{
    // handle a Band connection exception
}

```

4. Create a new tile.

```

// Create the small and tile icons from writable bitmaps.
// Small icons are 24x24 pixels.
WriteableBitmap smallIconBitmap = new WriteableBitmap(24, 24);
BandIcon smallIcon = smallIconBitmap.ToBandIcon();
// Tile icons are 46x46 pixels for Microsoft Band 1, and 48x48 pixels
// for Microsoft Band 2.
WriteableBitmap tileIconBitmap = new WriteableBitmap(46, 46);
BandIcon tileIcon = tileIconBitmap.ToBandIcon();

// create a new Guid for the tile
Guid tileGuid = Guid.NewGuid();

// create a new tile with a new Guid
BandTile tile = new BandTile(tileGuid)
{
    // enable badging (the count of unread messages)
    IsBadgingEnabled = true,
    // set the name
    Name = "TileName",
    // set the icons
    SmallIcon = smallIcon,
    TileIcon = tileIcon
}

```

```
};

try
{
    // add the tile to the Band
    if (await bandClient.TileManager.AddTileAsync(tile))
    {
        // do work if the tile was successfully created
    }
}
catch (BandException ex)
{
    // handle a Band connection exception
}
```

5. Remove all the application's tiles.

```
try
{
    // get the current set of tiles
    IEnumerable<BandTile> tiles = await
bandClient.TileManager.GetTilesAsync();

    foreach (var t in tiles)
    {
        // remove the tile from the Band
        if (await bandClient.TileManager.RemoveTileAsync(t))
        {
            // do work if the tile was successfully removed
        }
    }
}
catch (BandException ex)
{
    // handle a Band connection exception
}
```

7 SENDING NOTIFICATIONS

7.1 SENDING DIALOGS

7.1.1 Android

```
try {
    // send a dialog to the Band for one of our tiles
    bandClient.getBandNotificationManager().showDialog(tileUuid,
"Dialog title", "Dialog body").await();
} catch (BandException e) {
    // handle BandException
} catch (InterruptedException e) {
    // handle InterruptedException
}
```

7.1.2 iOS

```
[self.client.notificationManager
    showDialogWithTileID:tileId
                title:@"Dialog title"
                body:@"Dialog body"
    completionHandler:^(NSError *error)
{
    if (error){
        // handle error
    }
}];
```

7.1.3 Windows

```
try
{
    // send a dialog to the Band for one of our tiles
    await bandClient.NotificationManager.ShowDialogAsync(tileGuid,
"Dialog title", "Dialog body");
}
catch (BandException ex)
{
    // handle a Band connection exception
}
```

7.2 SENDING MESSAGES

7.2.1 Android

1. Update imports.

```
import java.util.Date;
import com.microsoft.band.notification.MessageFlags;
```


2. Send a message showing it as a dialog.

```
try {
    // Send a message to the Band for one of our tiles,
    // and show it as a dialog.
    bandClient.getBandNotificationManager().sendMessage(tileUuid,
"Message title", "Message body", new Date(),
MessageFlags.SHOW_DIALOG).await();
} catch (BandException e) {
    // handle BandException
} catch (InterruptedException e) {
    // handle InterruptedException
}
```

7.2.2 iOS

1. Send a message showing it as a dialog.

```
[self.client.notificationManager
    sendMessageWithTileID:tileId
                title:@"Message title"
                body:@"Message body"
        timeStamp:[NSDate date]
        flags:MSBNotificationMessageFlagsShowDialog
    completionHandler:^(NSError *error)
{
    if (error){
        // handle error
    }
}];
```

7.2.3 Windows

1. Update using directives.

```
using Microsoft.Band.Notifications;
```

2. Send a message showing it as a dialog.

```
try
{
    // Send a message to the Band for one of our tiles,
    // and show it as a dialog.
    await bandClient.NotificationManager.SendMessageAsync(tileGuid,
"Message title", "Message body", DateTimeOffset.Now,
MessageFlags.ShowDialog);
}
catch (BandException ex)
{
    // handle a Band connection exception
}
```

7.3 ALLOWING TILES TO RECEIVE LOCAL AND REMOTE NOTIFICATIONS (iOS APPS ONLY)

7.3.1 iOS

1. Register local and remote notifications to a specific tile on the Band. This is useful if your app owns multiple tiles.

```
[self.client.notificationManager
registerNotificationWithTileID:[tile tileId]
completionHandler:^(NSError *error) {
    if (error){
        // handle error
    }
}];
```

2. Register local and remote notifications to the default tile you own.

```
[self.client.notificationManager
registerNotificationWithCompletionHandler:^(NSError *error) {
    if (error){
        // handle error
    }
}];
```

3. Unregister local and remote notifications from the Band.

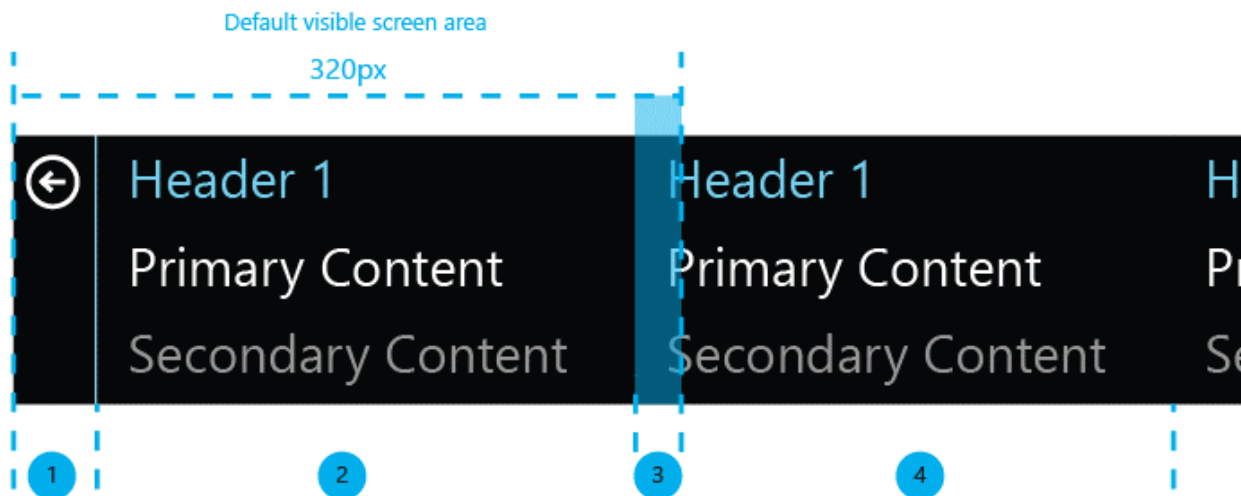
```
[self.client.notificationManager
unregisterNotificationWithCompletionHandler:^(NSError *error) {
    if (error){
        // handle error
    }
}];
```

8 CUSTOMIZING TILE LAYOUTS

The tiles that you add to Microsoft Band can each contain up to 8 pages of content. When the user taps the tile, the first page is shown. The user can swipe horizontally to navigate to other pages in the tile.

Here is the general format of Microsoft Band 1 screen shown after the user has tapped a tile:

- Left edge (section 1 below, 40 pixels wide) always shows the Back Bar, enabling the user to exit the app tile and return to the Start Strip
- Middle (section 2 below, 258 pixels wide for Microsoft Band 2, 245 pixels wide for Microsoft Band 1) shows the current page of the tile
- Right edge (section 3 below, 22 pixels wide for Microsoft Band 2, 35 pixels wide for Microsoft Band 1) shows a peek at the left-most portion of the next page to the right of the current page
- The remainder of the next page (section 4) and subsequent pages are not visible, but the user can swipe to navigate to the next page to make it visible



8.1 CREATING A LAYOUT FOR A PAGE

The visual format of the content contained within each page can be customized by creating a layout for it. A tile can have up to 5 layouts and each of the (up to) 8 pages of the tile can be configured to use any of the 5 layouts.

8.1.1 Primitive Element Types

The primitive visual element types that contain content and can be formatted with layouts are shown in the following table.

Name	Description
TextBlock	Text that's clipped if it's too wide to fit within its bounds
WrappedTextBlock	Text that wraps if it's too wide to fit within its bounds
Icon	Single color bitmap consisting of alpha values that control the opacity of each pixel
Barcode	Text encoded as a Code39 or PDF417 barcode

TextButton	White text label on black background button
FilledButton	Rectangular button filled with background color

TextButton and FilledButton elements generate events to your app when they are pressed by the user. Those events are described later in the section [Handling Custom Tile Events](#). The other element types don't generate events.

Note: A layout may contain a maximum of 10 primitive elements.

8.1.2 Container Element Types

Primitive elements are the leaf nodes of a layout. The tree of the elements begins with a PageLayout, serving as the root element of the layout. This root will have exactly one child element. It will be one of the following container elements.

Name	Description
FlowPanel	Arranges its child elements sequentially on the page, either horizontally or vertically
ScrollFlowPanel	Similar to FlowPanel, and with the ability to scroll between children that aren't currently within the visible area
FilledPanel	Rectangle filled with a background color

Container elements can have 0 or more child elements. These children can be FlowPanel, ScrollFlowPanel, or primitive elements. **A FilledPanel can be used only as the root element of a layout. A FilledPanel cannot be used as a child of a FlowPanel or ScrollFlowPanel.**

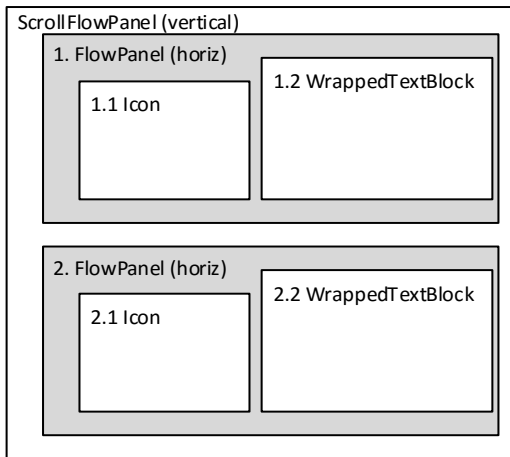
Note: The total number of elements (primitive and container) that a layout may contain is limited to about 20. The exact number will vary, depending on the type of elements in the layout.

8.1.3 Example Layout Element Tree

To display a day's weather for both day and night, you could have the following:

1. A PageLayout that has a vertical ScrollFlowPanel as its child.
2. The vertical ScrollFlowPanel has 2 horizontal FlowPanels as children.
3. Each horizontal FlowPanel has 2 children:
 - a. An Icon that will show a sun/moon, cloud, or rain image
 - b. A WrappedTextBlock with several lines of text, showing info, such as the temperature and chance of precipitation.

That will create a Page layout with elements structured like this.



8.1.4 Element Attributes

Elements have attributes to control their appearance, including the following.

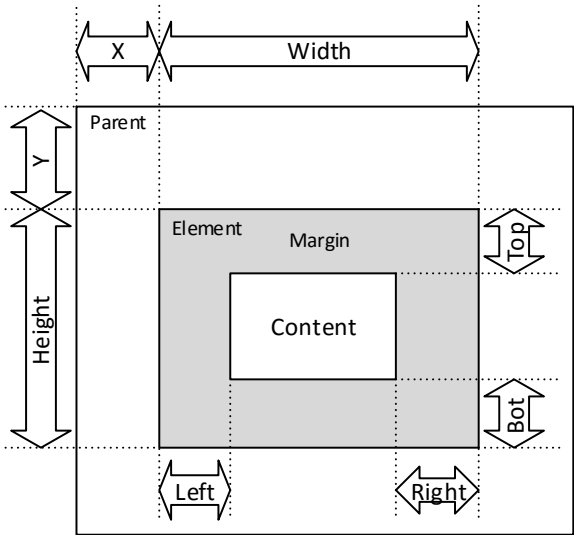
Name	Type (default in bold)	Required	Description	Applicable Element Types
Rect	PageRect 4 UInt16 (X, Y, Width, Height)	Yes	Defines the X,Y location of the element's upper-left corner, relative to its parent, and its width and height (in pixels)	All
Margins	Margins 4 Int16 (Left, Top, Right, Bottom) (0,0,0,0)	No	Defines the margin sizes (in pixels)	All
Background ColorSource	ElementColorSource (Custom)	No	Specifies the color of the FilledPanel. If set to "Custom," the color will be that of the "BackgroundColor" attribute.	FilledPanel
Background Color	BandColor (Black)	No	The RGB color of the FilledPanel	FilledPanel
ColorSource	ElementColorSource (Custom)	No	Specifies the color of the core feature of the element. If set to "Custom", the color will be that of the "Color" attribute.	TextBlock, WrappedTextBlock, Icon, ScrollFlowPanel
Color	BandColor (default varies depending on element type, see Colors)	No	The RGB color of the core feature of the element (for example, text color)	TextBlock, WrappedTextBlock, Icon, TextButton, FilledButton, ScrollFlowPanel
Horizontal Alignment	HorizontalAlignment Left , Right, Centered	No	How content should be aligned horizontally within the element	All
Vertical Alignment	VerticalAlignment	No	How content should be aligned vertically within the element	All

	Top, Bottom, Centered			
Font	TextBlockFont WrappedTextBlockFont Small, Medium TextBlock Only: Large, ExtraLargeNumbers, ExtraLargeNumbersBold	Yes	Specifies the set of glyphs to render a string Note: Not all character codes have glyphs in each font. Characters with no corresponding glyph in the font will be rendered with a default placeholder glyph.	TextBlock, WrappedTextBlock
Baseline Alignment	TextBlockBaselineAlignment Automatic, Relative	No	“Automatic”: use the TextBlock’s “Vertical Alignment” attribute to position the text “Relative”: the “Baseline” attribute will position the text baseline	TextBlock
Baseline	Int16 (o)	No	If “Baseline Alignment” is “Relative”, “Baseline” specifies the vertical offset (in pixels) from the top of the TextBlock at which to draw the string	TextBlock
AutoWidth	bool (true)	No	“true” causes the TextBlock to expand horizontally to fit the text content. “false” means the width of the TextBlock is fixed by the “Rect” attribute.	TextBlock
AutoHeight	bool (true)	No	“true” causes the WrappedTextBlock to expand vertically to fit the text content. “false” means the height of the WrappedTextBlock is fixed by the “Rect” attribute.	WrappedTextBlock
Orientation	Horizontal, Vertical	No	Defines whether child elements of the list are arranged horizontally or vertically	FlowPanel, ScrollFlowPanel
ElementId	Int16	No	Defines a unique (within the layout) identifier, used when setting the element content and determining which button was pressed in a button event. Note: o isn’t a valid ElementId. Any element with the Id of o won’t be updated with content. Therefore, ElementId should always be a positive integer	All

			between 1 and 32767.	
--	--	--	----------------------	--

The [Band Visual Guidelines](#) documentation shows you do to design page layouts for your app. That document includes details about font character sets and theme colors.

8.2 ELEMENT RECTANGLE AND MARGINS



Each element must define its “Rect” attribute that specifies the X,Y offset (in pixels) of the upper-left corner of the element relative to its parent, and the width and height (in pixels) of the element.

The X and Y coordinates of children of a FlowPanel or ScrollFlowPanel should both be set to 0. The parent will automatically position the element next to the element’s preceding sibling.

Margins can be used to create blank space along the inside edges of an element. Each of the left, top, right, and bottom margin sizes (in pixels) can be set. The content of the element won’t be drawn in the margins.

8.2.1 Negative Margins

When an element’s parent is a FlowPanel or ScrollFlowPanel, a negative margin value can be set to reposition the content of the element on top of the preceding child of the FlowPanel.

For example, suppose you want to have a button with an icon superimposed on it. One way to do that would be to construct a horizontal FlowPanel with 2 children.

- 1. FilledButton with a width=100, height=50
- 2. Icon with a width of 80, height=50, left_margin=-90, right_margin=-90

The -90 value of the left_margin of the Icon shifts it such that the Icon left edge is 90 pixels to the left of the right edge of its predecessor in its parent FlowPanel (the 100-pixel wide FilledButton). The Icon is superimposed over the FilledButton and indented 10 pixels from the left edge of the FilledButton.

8.3 DRAW ORDER FOR ELEMENTS

The drawing for container elements is done in the order they are added to the layout. The first container element added will be at the bottom of the z-order. However, the drawing for primitive elements within a container element is done in reverse order, based on when they are added in the layout. Primitive elements are drawn from

last to first, which means that the element that was added first to the container in the layout will be drawn last. In case of overlapping elements, the element at the top of the z-order would be the one that was added first to the layout, followed below by the element that was added right after it, and so on. This fact should be kept in mind when trying to overlay primitive elements; for example, drawing an Icon over FilledButton.

8.4 COLORS

Most elements have the “ColorSource” and “Color” attributes that are used to control the element’s core feature color. If the “ColorSource” is set to “Custom”, the RGB value specified in the “Color” attribute will be used. Otherwise, the “ColorSource” specifies that one of the 6 Band Theme colors, or 6 Tile Theme colors should be used. In that case, the color value will be determined dynamically and will reflect the current state of those Theme colors.

The core feature color that is controlled by the color attribute of each type of element is defined in the following table.

Element Type	Default Color	Feature Controlled
TextBlock	White	Font color
WrappedTextBlock	White	Font color
Icon	White	Render color
TextButton	Gray	Button background color when pressed
FilledButton	White	Button background color when not pressed
FilledButton	Gray	Button background color when pressed
FilledPanel	Black	Color for the rectangle background
ScrollFlowPanel	White	Scroll bar color

8.5 ICONS

When you create a tile, you provide the normal (unbadged) and small (badged) icons that will be shown on the Start Strip. When you create a custom layout for a tile, you can provide up to 8 (if using Microsoft Band 1) or 13 (if using Microsoft Band 2) additional icons that can be used within the layout.

You can create Icons from native bitmaps by using the helper method provided in the SDK. This helper method will use the alpha component of each pixel in the native bitmap to construct the Band Icon object. The red, green, and blue components of each pixel in the native bitmap are ignored. The alpha value of each pixel in the Icon is used to determine the transparency of the color rendered on the screen. Each pixel of an Icon element will use the same color as set in the Icon element’s “Color” attribute in the layout.

The page content specifies which particular icon is displayed within an Icon element. The content of an Icon element is the index value of the Icon to be displayed. The normal and small tile Icons are at index values 0 and 1, respectively, and additional icons that were provided start at index value 2. The layout can make use of any of these Icons.

8.5.1 Icons Used as FilledButton Masks

By defining an Icon bitmap that acts as a mask and then superimposing that Icon over a FilledButton (see Negative Margins), you can create the effect of the Icon image becoming visible when the button is pressed. That is, the Icon bitmap is defined to have transparent pixels for the desired image, and opaque pixels elsewhere. When the user presses the FilledButton, the FilledButton color changes but shows through only the transparent portions of the Icon bitmap.

8.6 BARCODES

The content of a Barcode element is specified as a string and BarcodeType. The supported BarcodeTypes are PDF417 and CODE39. The Band will convert the string into its graphical barcode representation that's based on the type.

8.6.1 Supported character set for PDF417 barcodes

8.6.1.1 *Microsoft Band 1*

Each character must be a digit (0-9). The maximum length of the string is 39 characters. But because of the Band screen width and resolution, the effective upper limit that can be rendered is 20 characters.

8.6.1.2 *Microsoft Band 2*

Microsoft Band 2 supports a much wider range of characters for PDF417-based barcodes. ASCII character 9 (Horizontal Tab), 10(New Line), 13(Carriage Return), and characters 32(Space) through 126(~) are supported. The maximum length of the string allowed is 39 characters, but depending on the characters used in the barcode and the rendering space available to the barcode element, the actual length that can be rendered might be less than 39.

If the Band is unable to render the entire Barcode content within the space available to the element in the layout, the available space will be blank. That is, if the complete Barcode cannot be displayed, no portion of the Barcode will be displayed.

8.6.2 Supported character set for CODE39 barcodes

On both Microsoft Band 1 and Microsoft Band 2, each character must be a valid Code 39 character: 0-9, A-Z, <space>, \$, %, +, -, /, or . (period). The Band will allow a string of up to 39 characters to be assigned, but because of the Band screen width and resolution, the effective upper limit that can be rendered is 12 characters.

Like with PDF417 barcodes, if the Band is unable to render the entire CODE39 Barcode content within the space available to the element in the layout, the available space will be blank. That is, if the complete Barcode cannot be displayed, no portion of the Barcode will be displayed.

8.7 SETTING THE CONTENTS OF A PAGE

Layouts describe only the visual formatting of the content. The content is set separately as needed to update the Band. That is, the layout is set once when the tile is created, and then content is pushed to the Band afterward as necessary.

When setting the content of a tile page, you specify the tile's Guid and the page's Guid. If the page Guid matches 1 of the 8 current pages of the tile, the content will update that existing page. If the page Guid doesn't match any of those current pages, a new page will be created to hold the content. New pages are added on the left, pushing any existing pages to the right. If the tile already has 8 pages, the oldest (right-most) page will be discarded.

The following table specifies the type of content that can be assigned to each element.

Element Type	Content Data Type	Content Description
TextBlock	TextBlockData	String to display
WrappedTextBlock	WrappedTextBlockData	String to display
Icon	IconData	Icon index (0-9 for Microsoft Band 1 and 0-14 for Microsoft Band 2) to render
Barcode	BarcodeData	String and BarcodeType to render
TextButton	TextButtonData	String to display
FilledButton	FilledButtonData	Color to display when the button is being pressed

Note: For Microsoft Band 1, at most 1 TextBlock/WrappedTextBlock can have a string content longer than 20 characters in a tile page. The limit of this single “long” string is 160 characters. Every other string on the page must be ≤ 20 characters. This restriction doesn’t apply to Microsoft Band 2.

8.8 SIMPLE CUSTOM TILE EXAMPLE

The following code samples demonstrate how to add a tile that has a single page of content to the Band. The layout of the page is formatted as a vertical ScrollFlowPanel with 2 child WrappedTextBlock elements, each of which will be used to display a text message to the user. Briefly, the steps shown in the example are as follows.

1. Create a tile.
2. Create a layout that consists of a vertical ScrollFlowPanel with children that are wrapped text blocks to display the text messages.
3. Add the layout to the tile.
4. Add the tile to the Band.
5. Set the message content of the text blocks.

8.8.1 Android

1. Update imports.

```
import com.microsoft.band.tiles.pages.ScrollFlowPanel;
import com.microsoft.band.tiles.pages.FlowPanelOrientation;
import com.microsoft.band.tiles.pages.PageData;
import com.microsoft.band.tiles.pages.PageLayout;
import com.microsoft.band.tiles.pages.PageTextBlockData;
import com.microsoft.band.tiles.pages.TextBlock;
import com.microsoft.band.tiles.pages.TextBlockFont;
import com.microsoft.band.tiles.pages.ElementColorSource;
```

2. Create the layout.

```
// Define symbolic constants for indexes to each layout that
// the tile has. The index of the first layout is 0. Because
// only 5 layouts are allowed, the max index value is 4.
enum TileLayoutIndex {
    MessagesLayout
```

```

}

// Define symbolic constants to uniquely (within the
// messages page) identify each of the elements of our layout
// that contain content that the app will set
// (that is, these Ids will be used when calling APIs
// to set the page content).
// Always start from 1 because 0 isn't a valid element ID.
enum TileMessagesPageElementId {
    Message1 = 1,
    Message2
}

// create a scrollable vertical panel that will hold 2 text messages
ScrollFlowPanel panel = new ScrollFlowPanel(new PageRect(0, 0, 245,
102));
panel.setFlowPanelOrientation(FlowPanelOrientation.VERTICAL);
panel.setHorizontalAlignment(PageHorizontalAlignment.LEFT);
panel.setVerticalAlignment(PageVerticalAlignment.TOP);

// create the first text block
WrappedTextBlock textBlock1 = new WrappedTextBlock(new PageRect(0, 0,
245, 102), WrappedTextBlockFont.MEDIUM);
textBlock1.setId(TileMessagesPageElementId.Message1.ordinal());
textBlock1.setMargins(new PageMargin(15, 0, 15, 0));
textBlock1.setColor(Color.WHITE);
textBlock1.setAutoHeightEnabled(true);
textBlock1.setHorizontalAlignment(PageHorizontalAlignment.LEFT);
textBlock1.setVerticalAlignment(PageVerticalAlignment.TOP);

// create the second text block
WrappedTextBlock textBlock2 = new WrappedTextBlock(new PageRect(0, 0,
245, 102), WrappedTextBlockFont.MEDIUM);
textBlock2.setId(TileMessagesPageElementId.Message2.ordinal());
textBlock2.setMargins(new PageMargin(15, 0, 15, 0));
textBlock2.setColorSource(ElementColorSource.BAND_BASE);
textBlock2.setAutoHeightEnabled(true);
textBlock2.setHorizontalAlignment(PageHorizontalAlignment.LEFT);
textBlock2.setVerticalAlignment(PageVerticalAlignment.TOP);

// add the text blocks to the panel
panel.addElements(textBlock1, textBlock2);

// create the page layout
PageLayout layout = new PageLayout(panel);

```

3. Create the tile and add the layout.

```
// create the tile and add the layout
BandTile tile = new BandTile.Builder(tileId, "MyTile", tileIcon)
    .setPageLayouts(layout)
    .build();
```

4. Add the tile to the Band.

```
// add the tile to the Band
try {
    if(!client.getTileManager().addTile(this, tile).await()) {
        // handle add tile failure
    }
} catch(BandException ex) {
    // handle exception
}
```

5. Set the content for the page on the Band.

```
try {
    if(client.getTileManager().setPages(tileId,
        new PageData(TilePageId.PAGE1.Uuid,
            TileLayoutIndex.MESSAGES_LAYOUT.ordinal())
            .update(new
                PageWrappedTextBlockData(TileMessagesPageElementId.MESSAGE1.ordinal(),
                    "This is the text of the first
message")))
        .update(new
            PageWrappedTextBlockData(TileMessagesPageElementId.MESSAGE1.ordinal(),
                "This is the text of the second
message")))).await()) {
        // handle set pages failure
    }
} catch(BandException ex) {
    // handle exception
}
```

8.8.2 iOS

1. Update imports.

```
#import <MicrosoftBandKit_iOS/MicrosoftBandKit_iOS.h>
```

2. Create the tile.

```
NSUUID *tileId = [NSUUID UUID];
// create a new MSBTile
NSError * error;
MSBTile *tile = [MSBTile tileWithId:tileId name:@"sample"
    tileIcon:tileIcon smallIcon:smallIcon error:&error];
if (error){
    // handle error
}
```

```
}
```

3. Create a layout.

```
// Create a scrollable vertical panel that will hold 2 text messages.
MSBPageScrollFlowPanel *panel = [[MSBPageScrollFlowPanel alloc]
initWithRect:[MSBPageRect rectWithX:0 y:0 width:245 height:102]];
panel.horizontalAlignment = MSBPageHorizontalAlignmentLeft;
panel.verticalAlignment = MSBPageVerticalAlignmentTop;

// Define symbolic constants for indexes to each layout that
// the tile has. The index of the first layout is 0. Because
// only 5 layouts are allowed, the max index value is 4.
typedef enum LayoutIndex
{
    LayoutIndexMessages = 0,
} LayoutIndex;

// Define symbolic constants to uniquely (within the
// messages page) identify each of the elements of our layout
// that contain content that the app will set
// (that is, these Ids will be used when calling APIs
// to set the page content).
typedef enum MessagesPageElementId : uint16_t
{
    MessagesPageElementId1 = 1, // Id for the 1st message text block
    MessagesPageElementId2 = 2, // Id for the 2nd message text block
} MessagesPageElementId;

// add the text block to contain the first message
MSBPageWrappedTextBlock *textBlock1 = [[MSBPageWrappedTextBlock alloc]
initWithRect:[MSBPageRect rectWithX:0 y:0 width:245 height:102]
font:MSBPageWrappedTextBlockFontMedium];
textBlock1.elementId = MessagesPageElementId1;
textBlock1.margins = [MSBPageMargins marginsWithLeft:15 top:0 right:15
bottom:0];
textBlock1.color = [MSBColor colorWithRed:0xFF green:0xFF blue:0xFF];
textBlock1.autoHeight = YES;
textBlock1.horizontalAlignment = MSBPageHorizontalAlignmentLeft;
textBlock1.verticalAlignment = MSBPageVerticalAlignmentTop;

[panel addElement:textBlock1];

// add the text block to contain the second message
MSBPageWrappedTextBlock *textBlock2 = [[MSBPageWrappedTextBlock alloc]
initWithRect:[MSBPageRect rectWithX:0 y:0 width:245 height:102]
font:MSBPageWrappedTextBlockFontMedium];
```

```

textBlock2.elementId = MessagesPageElementId2;
textBlock2.margins = [MSBPageMargins marginsWithLeft:15 top:0 right:15
bottom:0];
textBlock2.color = [MSBColor colorWithRed:0xFF green:0xFF blue:0xFF];
textBlock2.autoHeight = YES;
textBlock2.horizontalAlignment = MSBPageHorizontalAlignmentLeft;
textBlock2.verticalAlignment = MSBPageVerticalAlignmentTop;

[panel addElement:textBlock2];

// create the page layout
MSBPageLayout *layout = [[MSBPageLayout alloc] init];
layout.root = panel;

```

4. Add the layout to the tile.

```
[tile.pageLayouts addObject:layout];
```

5. Add the tile to the Band.

```

// prerequisite: bandClient has successfully connected to a Band
[self.client.tileManager addTile:tile completionHandler:^(NSError
*error){
    if (error){
        // add tile failed, handle error
    }
}];

```

6. Set the content of the page on the Band.

```

// create a new Guid for the messages page
NSUUID *messagesPageId = [NSUUID UUID];

// create the object that contains the page content to be set
MSBPageWrappedTextBlockData *textData1 = [MSBPageWrappedTextBlockData
pageWrappedTextBlockDataWithElementId:MessagesPageElementId1
text:@"This is the text of the first message" error:&error];
if (error)
{
    // handle error
}

MSBPageWrappedTextBlockData *textData2 = [MSBPageWrappedTextBlockData
pageWrappedTextBlockDataWithElementId:MessagesPageElementId2
text:@"This is the text of the second message" error:&error];
if (error)
{
    // handle error
}

```

```

MSBPageData *pageData = [MSBPageData pageDataWithId:messagesPageId
layoutIndex:LayoutIndexMessages value:@[textData1, textData2]];

[self.client.tileManager setPages:@[pageData] tileId:tileId
completionHandler: ^(NSError *error){
    if (error)
    {
        // unable to set data to the Band
    }
}];

```

8.8.3 Windows

1. Update using directives.

```
using Microsoft.Band.Tiles.Pages
```

2. Create the tile.

```

// create a new Guid for the tile
Guid tileGuid = Guid.NewGuid();

// create a new tile
BandTile tile = new BandTile(tileGuid)
{
    // set the name
    Name = "MyTile",
    // set the icons
    SmallIcon = smallIcon,
    TileIcon = tileIcon
};

// Our layout doesn't use Icons, but if it did, we would
// add our Icons to the tile by calling tile.AdditionalIcons.Add
// to add our (up to 8) additional BandIcons.

```

3. Create a layout.

```

// Create a scrollable vertical panel that will hold 2 text messages.
ScrollFlowPanel panel = new ScrollFlowPanel.
{
    Rect = new PageRect(0, 0, 245, 102),
    Orientation = FlowListOrientation.Vertical,
    ColorSource = ElementColorSource.BandBase
};

// Define symbolic constants for indexes to each layout that

```

```

// the tile has. The index of the first layout is 0. Because only
// 5 layouts are allowed, the max index value is 4.
internal enum TileLayoutIndex
{
    MessagesLayout = 0,
};

// Define symbolic constants to uniquely (in MessagesLayout)
// identify each of the elements of our layout
// that contain content that the app will set
// (that is, these Ids will be used when calling APIs
// to set the page content).
internal enum TileMessagesLayoutElementId : short
{
    Message1 = 1, // Id for the 1st message text block
    Message2 = 2, // Id for the 2nd message text block
};

// add the text block to contain the first message
panel.Elements.Add(
    new WrappedTextBlock
    {
        ElementId = (short)TileMessagesLayoutElementId.Message1,
        Rect = new PageRect(0, 0, 245, 102),
        // left, top, right, bottom margins
        Margins = new Margins(15, 0, 15, 0),
        Color = new BandColor(0xFF, 0xFF, 0xFF),
        Font = WrappedTextBlockFont.Small
    }

// add the text block to contain the second message
panel.Elements.Add(
    new WrappedTextBlock
    {
        ElementId = (short)TileMessagesLayoutElementId.Message2,
        Rect = new PageRect(0, 0, 245, 102),
        // left, top, right, bottom margins
        Margins = new Margins(15, 0, 15, 0),
        Color = new BandColor(0xFF, 0xFF, 0xFF),
        Font = WrappedTextBlockFont.Small
    }
);

// create the page layout
PageLayout layout = new PageLayout(panel);

```


4. Add the layout to the tile.

```
try
{
    // add the layout to the tile
    tile.PageLayouts.Add(layout);
}
catch(BandException ex)
{
    // handle an error adding the layout
}
```

5. Add the tile to the Band.

```
// prerequisite: bandClient has successfully connected to a Band

try
{
    // add the tile to the Band
    if (await bandClient.TileManager.AddTileAsync(tile))
    {
        // tile was successfully added
        // can proceed to set tile content with SetPagesAsync
    }
    else
    {
        // tile failed to be added, handle error
    }
}
catch(BandException ex)
{
    // handle a Band connection exception
}
```

6. Set the content of the page on the Band.

```
// create a new Guid for the messages page
Guid messagesPageGuid = Guid.NewGuid();

// create the object that contains the page content to be set
PageData pageContent = new PageData(
    messagesPageGuid,
    // specify which layout to use for this page
    TileLayoutIndex.MessagesLayout,
    new WrappedTextBlockData(
        (Int16)TileMessagesLayoutElementId.Message1,
        "This is the text of the first message"),
    new WrappedTextBlockData(
```

```
        (Int16)TileMessagesLayoutElementId.Message2,  
        "This is the text of the second message")  
);  
  
try  
{  
    // set the page content on the Band  
    if (await bandClient.TileManager.SetPagesAsync(tileGuid,  
pageContent))  
    {  
        // page content successfully set on Band  
    }  
    else  
    {  
        // unable to set content to the Band  
    }  
}  
catch (BandException ex)  
{  
    // handle a Band connection exception  
}
```

9 HANDLING CUSTOM TILE EVENTS

When the user taps a tile you've added to Microsoft Band, an event is generated to notify your application of that action. Similarly, when the user taps one of the button elements on a tile's pages, an event is sent to your app.

There are 3 types of events that can be generated for your tiles:

- Tile entered (user has tapped a tile)
- Tile exited (user has pressed the Back button to exit a tile)
- Button pushed (user has pushed a button element on a tile's page)

To receive these events for a custom tile, you must do the following.

1. Create the custom tile.
2. Implement handlers that will be called when one of these events occurs.
3. Subscribe to the event types (open, close, button push) that you're interested in.

Your handlers will be provided with the event object that will provide the GUID of the tile that triggered the event and the time at which the event was triggered. If the event is for a button push, the event object will also provide both the GUID of the page that contains the button and the button element ID that you assigned in the layout.

The following example shows how to create a tile with a single page that contains a button, and how to define and register a handler for the events generated by the Band for the tile.

9.1 ANDROID

For Android, tile events are broadcast by using intents. To receive tile events, an application needs to implement a broadcast receiver to handle intents with the following actions:

- `com.microsoft.band.action.ACTION_TILE_OPENED`
- `com.microsoft.band.action.ACTION_TILE_BUTTON_PRESSED`
- `com.microsoft.band.action.ACTION_TILE_CLOSED`

Here is an example implementation of a broadcast receiver that handles tile events.

```
public class TileEventReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction() ==
"com.microsoft.band.action.ACTION_TILE_OPENED") {
            // handle tile opened event
        }
        else if (intent.getAction() ==
"com.microsoft.band.action.ACTION_TILE_BUTTON_PRESSED") {
            // handle button pressed event
        }
        else if (intent.getAction() ==
"com.microsoft.band.action.ACTION_TILE_CLOSED") {
            // handle tile closed event
        }
    }
}
```

It's important to note these intents will be broadcast by only the Microsoft Band SDK to the application package that created the tile. This is especially important if you plan to handle broadcasts in an application service. You must make sure the service is running in the same package as the application's UI activity (if created in the UI activity) or the service is creating the tile.

A nice benefit of using intents for tile events is that the application doesn't need to be running in the foreground to receive them. You can accomplish this by adding a receiver definition in the application's manifest and adding an intent filter for the receiver, shown as follows.

```
<application
    ...
    <receiver
        android:name=".TileEventReceiver"
        android:label="@string/app_name"
        android:enabled="true"
        android:exported="true">
        <intent-filter>
            <action
                android:name="com.microsoft.band.action.ACTION_TILE_OPENED" />
            <action
                android:name="com.microsoft.band.action.ACTION_TILE_BUTTON_PRESSED" />
            <action
                android:name="com.microsoft.band.action.ACTION_TILE_CLOSED" />
        </intent-filter>
    </receiver>
    ...
</application>
```

9.2 iOS

For iOS, tile events are pushed to the application as long as the application has a connected MSBClient. The prerequisite is that the application needs to have at least one tile added to the Band. This sample code shows you how to create a tile with a Text Button element and an observe button event if a user presses the button.

1. Update using directives.

```
#import <MicrosoftBandKit_iOS/MicrosoftBandKit_iOS.h>
```

2. Implement required delegate methods.

```
// set delegate object in the desired class
self.client = [[MSBClientManager sharedManager]
    clientWithConnectionIdentifier:persistedConnectionId];
self.client.tileDelegate = self;

-(void)client:(MSBClient *)client tileDidOpen:(MSBTileEvent *)event
{
    NSLog(@"%@", event);
}
```

```

-(void)client:(MSBClient *)client buttonDidPress:(MSBTileButtonEvent
*)event
{
    NSLog(@"%@", event);
}

-(void)client:(MSBClient *)client tileDidClose:(MSBTileEvent *)event
{
    NSLog(@"%@", event);
}

```

3. Create the tile.

```

NSUUID *tileId = [NSUUID UUID];
// create a new MSBTile
NSError *error;
MSBTile *tile = [MSBTile tileWithId:tileId name:@"sample"
tileIcon:tileIcon smallIcon:smallIcon error:&error];
if (error){
    // handle error
}

```

4. Create a layout.

```

// create a scrollable vertical panel that will hold 2 text messages
MSBPageFlowPanel *panel = [[MSBPageFlowPanel alloc]
initWithRect:[MSBPageRect rectWithX:0 y:0 width:245 height:102]];
panel.horizontalAlignment = MSBPageHorizontalAlignmentLeft;
panel.verticalAlignment = MSBPageVerticalAlignmentTop;

// add the text button
MSBPageTextButton *textButton = [[MSBPageTextButton alloc]
initWithRect:[MSBPageRect rectWithX:0 y:0 width:245 height:102]];
textButton.elementId = 1;
textButton.margins = [MSBPageMargins marginsWithLeft:15 top:0 right:15
bottom:0];
textButton.pressedColor = [MSBColor colorWithRed:0xFF green:0xFF
blue:0xFF];

[panel addElement:textButton];

// create the page layout
MSBPageLayout *layout = [[MSBPageLayout alloc] init];
layout.root = panel;

```

5. Add the layout to the tile.

```

[tile.pageLayouts addObject:layout];

```

6. Add the tile to the Band.

```
// prerequisite: bandClient has successfully connected to a Band
[self.client.tileManager addTile:tile completionHandler:^(NSError
*error){
    if (error){
        // add tile failed, handle error
    }
}];
```

7. Set the content of the page on the Band.

```
// create a new Guid for the messages page
NSUUID *buttonPageId = [NSUUID UUID];

// create the object that contains the page content to be set
MSBPageTextButtonData *textButtonData = [MSBPageTextButtonData
pageTextButtonDataWithElementId:1 text:@"Sample Button" error:&error];
if (error)
{
    // handle error
}

MSBPageData *pageData = [MSBPageData pageDataWithId:buttonPageId
layoutIndex:0 value:@[textButtonData]];

[self.client.tileManager setPages:@[pageData] tileId:tileId
completionHandler: ^(NSError *error){
    if (error)
    {
        // unable to set data to the Band
    }
}];
```

9.3 WINDOWS

On Windows, the application can receive events while it has a connected BandClient. If the application is a Universal Windows Platform app on Windows 10 Mobile, then it can receive tile events in its background code even when the app is not running in the foreground.

9.3.1 Tile Event Handling by Foreground App

1. Update using directives.

```
using Microsoft.Band.Tiles
```

2. Create the tile with a single page with a layout that contains a button.

```
// create a new Guid for the tile
Guid tileGuid = Guid.NewGuid();
```

```

// create a new tile with a new Guid
BandTile tile = new BandTile(tileGuid)
{
    // set the name
    Name = "MyTile",
    // set the icons
    SmallIcon = smallIcon,
    TileIcon = tileIcon
};

// create a filled rectangle to provide the background for a button
FilledPanel panel = new FilledPanel
{
    Rect = new PageRect(0, 0, 245, 102),
    Color = ThemeColor.BandBase
};

// define the element Ids for our tile's page
Enum TilePageElementId : short
{
    Button_PushMe = 1,
}

// add a button to our layout
panel.Elements.Add(
    new TextButton
    {
        ElementId = (short)TilePageElementId.Button_PushMe,
        Rect = new PageRect(60, 25, 100, 50),
        PressedColor = new BandColor(0xFF, 0x00, 0x00),
    }
);

// create the page layout
PageLayout layout = new PageLayout(panel);

// add the layout to the tile, and add the tile to the Band
try
{
    tile.PageLayouts.Add(layout);
    if (await bandClient.TileManager.AddTileAsync(tile))
    {
        // layout and tile added successfully
    }
    else
    {

```

```

        // tile failed to be added, handle error
    }
}
catch(BandException ex)
{
    // handle an error adding the layout
}

Guid pageGuid = Guid.NewGuid();

// create the content to assign to the page
PageData pageContent = new PageData(
    pageGuid,
    0, // index of our (only) layout
    new Button(
        TilePageElementId.Button_PushMe,
        "Push Me!")
);

// set the page content to the Band
try
{
    if (await bandClient.TileManager.SetPagesAsync(tileGuid,
pageContent))
    {
        // page content successfully set on Band
    }
    else
    {
        // unable to set content to the Band
    }
}
catch(BandException ex)
{
    // handle a Band connection exception
}

```

3. Define handlers for tile open, tile close, and button push events.

```

void EventHandler_TileOpened(object sender,
BandTileEventArgs<IBandTileOpenedEvent> e)
{
    // This method is called when the user taps our Band tile.
    //
    // e.TileEvent.TileId is the tile's Guid.
    // e.TileEvent.Timestamp is the DateTimeOffset of the event.

```



```

    //
    // handle the event
}

void EventHandler_TileClosed(object sender,
BandTileEventArgs<IBandTileClosedEvent> e)
{
    // This method is called when the user exits our Band tile.
    //
    // e.TileEvent.TileId is the tile's Guid.
    // e.TileEvent.Timestamp is the DateTimeOffset of the event.
    //
    // handle the event
}

void EventHandler_TileButtonPressed(object sender,
BandTileEventArgs<IBandTileButtonPressedEvent> e)
{
    // This method is called when the user presses the
    // button in our tile's layout.
    //
    // e.TileEvent.TileId is the tile's Guid.
    // e.TileEvent.Timestamp is the DateTimeOffset of the event.
    // e.TileEvent.PageId is the Guid of our page with the button.
    // e.TileEvent.ElementId is the value assigned to the button
    //                               in our layout (i.e.,
    //                               TilePageElementId.Button_PushMe).
    //
    // handle the event
}

```

4. Register the handlers and start listening for incoming events from the Band.

```

// Subscribe to events
bandClient.TileManager.TileOpened += EventHandler_TileOpened;
bandClient.TileManager.TileClosed += EventHandler_TileClosed;
bandClient.TileManager.TileButtonPressed +=
    EventHandler_TileButtonPressed;

// Start listening for events
bandClient.TileManager.StartReadingsAsync();

```

9.3.2 Background Tile Event Handling (Universal Windows Platform)

A Universal Windows Platform application running on Windows 10 Mobile can receive Band tile events when the app is not running in the foreground. The Microsoft Health app must be installed, and the Band must be

registered with the Microsoft Health app for the Band tile events to be available to the application's background code.

The steps shown above for creating and installing the tile on the Band should be followed. After the tile is installed, the app should call `SubscribeToBackgroundTileEventsAsync`:

```
// Subscribe to background tile events
await bandClient.SubscribeToBackgroundTileEventsAsync(tileGuid);
```

The app must also provide an `AppService` to receive and handle the background tile events that will now be sent to the application (by the Microsoft Health app). This is a new feature of Windows 10 that allows app to app communication. A general introduction to creating and using `AppServices` can be found at:

<https://msdn.microsoft.com/en-us/library/windows/apps/xaml/mt187314.aspx>

The following steps show how to create an `AppService` specifically for handling Band background tile events.

1. Add the app service extension to `package.appxmanifest`. This informs the OS that your app supports the "com.microsoft.band.observer" `AppService` on which Band tile events will be sent by the Microsoft Health app, and also tells the OS the name of the class within your app that implements the interface.

```
<Applications>
  <Application ...
    <Extensions>
      <uap:Extension Category="windows.appService"
EntryPoint="MyNamespace.MyTileEventHandler">
        <uap:AppService Name="com.microsoft.band.observer"/>
      </uap:Extension>
    </Extensions>
  ...
```

2. Add a Windows Runtime Component (Windows Universal) project to the solution. This will contain the background code that implements the `AppService`.
3. In the Windows Runtime Component project, add a class to implement the `AppService`. To continue with the names ("MyNamespace", "MyTileEventHandler") that we chose to use in the manifest example in step 1, this class would contain:

```
using Windows.ApplicationModel.AppService;
using Windows.ApplicationModel.Background;
using Windows.Foundation.Collections;
using Microsoft.Band;

namespace MyNamespace
{
    public sealed class MyTileEventHandler : IBackgroundTask
    {
        private BackgroundTaskDeferral backgroundTaskDeferral;
        private AppServiceConnection appServiceConnection;

        public void Run(IBackgroundTaskInstance taskInstance)
        {
```

```

        this.backgroundTaskDeferral = taskInstance.GetDeferral();
        taskInstance.Canceled += OnTaskCanceled;

        // Add handlers for tile events
        BackgroundTileEventHandler.Instance.TileOpened +=
EventHandler_TileOpened;
        BackgroundTileEventHandler.Instance.TileClosed +=
EventHandler_TileClosed;
        BackgroundTileEventHandler.Instance.TileButtonPressed +=
EventHandler_TileButtonPressed;

        // Set up handler for incoming app service request messages
        var details = taskInstance.TriggerDetails as
AppServiceTriggerDetails;
        this.appServiceConnection = details.AppServiceConnection;
        this.appServiceConnection.RequestReceived +=OnRequestReceived;
    }

    private void OnTaskCanceled(IBackgroundTaskInstance sender,
BackgroundTaskCancellationReason reason)
    {
        if (this.backgroundTaskDeferral != null)
        {
            this.backgroundTaskDeferral.Complete();
        }
    }
}

```

4. Add the implementation of OnRequestReceived handler to the MyTileEventHandler class.

```

private async void OnRequestReceived(AppServiceConnection sender,
AppServiceRequestReceivedEventArgs args)
{
    var messageDeferral = args.GetDeferral();

    ValueSet response = new ValueSet();
    ValueSet request = args.Request.Message;

    // Decode the received message and call the appropriate handler
    BackgroundTileEventHandler.Instance.HandleTileEvent(request);

    // Send the response
    await args.Request.SendResponseAsync(response);

    messageDeferral.Complete();
}

```

5. Add the implementation of the tile event handlers to the `MyTileEventHandler` class.

```
private void EventHandler_TileOpened(object sender,
BandTileEventArgs<IBandTileOpenedEvent> e)
{
    // TODO: Handle the tile opening
}

private void EventHandler_TileClosed(object sender,
BandTileEventArgs<IBandTileClosedEvent> e)
{
    // TODO: Handle the tile closing
}

Private void EventHandler_TileButtonPressed(object sender,
BandTileEventArgs<IBandTileButtonPressedEvent> e)
{
    // TODO: Handle the button push
}
```

Notes:

- The Band 2 is required. The original Band does not support background tile events with Windows 10.
- If your app wants to connect to the Band from the background, then the app should call `GetBandsAsync(isBackground: true)` to get a connection that will not interfere with any app that is running in the foreground and currently using a Band connection.
- Dispose any connected `IBandClient` when the `TileClosed` event occurs, so that other background apps can connect if their tiles are opened.
- Band tile events will only be received in the background from the Band that is currently registered with the Microsoft Health app on the phone. Other Bands that are paired to the phone but not registered with the Microsoft Health app will not generate background tile events.
- `IBandClient.UnsubscribeFromBackgroundTileEvents(tileGuid)` can be used to remove a subscription to background tile events.
- If the background task has not completed within 30 seconds of being triggered, then the OS will terminate it.
- Only the app that installed/owns a tile may subscribe to background tile events from that tile.
- When the debugger is connected to the app, the debugger will show an exception occurring in the background code when the Band SDK retrieves the current application ID. That exception is normal and expected, and is handled by the SDK.

9.3.2.1 Troubleshooting Background Tile Events

If the app does not receive background tile events, here are some things to try:

- Check that your `AppService` class implementation is in a Windows Runtime Component, not a class library
- Check that the namespace and class name of your `AppService` class implementation exactly match that declaration in the `package.appxmanifest`

- Check that your AppService class implements the IBackgroundTask interface
- Launch the Microsoft Health app and verify that it can manage the tiles on the Band

10 SENDING HAPTICS TO THE BAND

Haptics, or vibrations, allow the developer to send tactile responses to Microsoft Band by using the Band Notification Manager on the Band Client. This provides a feedback mechanism to enhance the notification experience for an application. Currently, the Microsoft Band SDK provides a list of predefined vibration types that the application developer can choose from.

10.1 ANDROID

```
try {
    // send a vibration request of type alert alarm to the Band
    bandClient.getBandNotificationManager().vibrate(VibrationType.NOTIFICATION_ALARM).await();
} catch (InterruptedException e) {
    // handle InterruptedException
} catch (BandException e) {
    // handle BandException
}
```

10.2 iOS

```
// send a vibration request of type alert alarm to the Band
[self.client.notificationManager
    vibrateWithType:MSBNotificationVibrationTypeAlarm
    completionHandler:^(NSError *error)
{
    if (error){
        // handle error
    }
}];
```

10.3 WINDOWS

```
try
{
    // send a vibration request of type alert alarm to the Band
    await
bandClient.NotificationManager.VibrateAsync(VibrationType.NotificationAlarm);
}
catch (BandException ex)
{
    // handle a Band connection exception
}
```


11 BAND PERSONALIZATION

The Band Personalization Manager on the Band Client allows the developer to personalize the Me Tile, or default UI, on Microsoft Band as well as setting the default theme colors for all tiles.

For Me Tile personalization, the Microsoft Band SDK allows setting the background color or image. The Band uses RGB565 image format, so all bitmaps will be converted to that (either implicitly or explicitly, depending on the platform) before being sent to the Band.

11.1 MANAGING THE ME TILE IMAGE

11.1.1 Android

1. Retrieve the current Me Tile image as a bitmap.

```
try {
    Bitmap meTileBitmap =
bandClient.getPersonalizationManager().getMeTileImage().await();
} catch (InterruptedException e) {
    // handle InterruptedException
} catch (BandException e) {
    // handle BandException
}
```

2. Set the current Me Tile image from a bitmap.

```
try {
    // Create a bitmap for the Me Tile image.
    // The image must be 310x102 pixels for Microsoft Band 1
    // and 310x102 or 310x128 pixels for Microsoft Band 2.
    Bitmap meTileBitmap = Bitmap.createBitmap(310, 102, null);
    bandClient.getPersonalizationManager().setMeTileImage(meTileBitmap).await();
} catch (InterruptedException e) {
    // handle InterruptedException
} catch (BandException e) {
    // handle BandException
}
```

11.1.2 iOS

1. Retrieve the current Me Tile image as a UIImage.

```
[self.client.personalizationManager
meTileImageWithCompletionHandler:^(MSBImage *image, NSError *error){
    if (error){
        // handle error
    }
    else if (!image){
        // no Me Tile image on the Band
    }
}
```

```

    }
    else {
        self.meTileImage = [image UIImage];
    }
}];

```

2. Set the current Me Tile image from a UIImage.

```

// Create a bitmap for the Me Tile image.
// The image must be 310x102 pixels for Microsoft Band 1
// and 310x102 or 310x128 pixels for Microsoft Band 2.

MSBImage *image = [[MSBImage alloc] initWithUIImage:[UIImage
initWithContentsOfFile:@"sampleMeTile.png"]];
[self.client.personalizationManager updateMeTileImage:image
                                completionHandler:^(NSError
*error)
{
    if (error){
        // handle error
    }
}];

```

11.1.3 Windows

1. Update using directives.

```
using Microsoft.Band.Personalization;
```

2. Retrieve the current Me Tile image as a writeable bitmap.

```

try
{
    // grab the Me Tile image from the Band
    BandImage bandImage = await
bandClient.PersonalizationManager.GetMeTileImageAsync();
    // convert it to a writeable bitmap
    WriteableBitmap bandImageBitmap = bandImage.ToWriteableBitmap();

    // do work with the image
}
catch (BandException ex)
{
    // handle a Band connection exception
}

```

3. Set the current Me Tile image from a writeable bitmap.

```

// Create a bitmap for the Me Tile image.
// The image must be 310x102 pixels for Microsoft Band 1
// and 310x102 or 310x128 pixels for Microsoft Band 2.

```



```

WriteableBitmap meTileBitmap = new WriteableBitmap(310, 102);
BandImage meTileImage = meTileBitmap.ToBandImage();

try
{
    await
bandClient.PersonalizationManager.SetMeTileImageAsync(meTileImage);
}
catch (BandException ex)
{
    // handle a Band connection exception
}

```

11.2 CHANGING THEME

11.2.1 Android

1. Get the current theme from the Band.

```

try {
    // get the current theme from the Band
    BandTheme theme =
bandClient.getPersonalizationManager().getTheme().await();
} catch (InterruptedException e) {
    // handle InterruptedException
} catch (BandException e) {
    // handle BandException
}

```

2. Set the current theme on the Band.

```

try {
    // set the current theme on the Band
    bandClient.getPersonalizationManager().setTheme(new
BandTheme(0x39BF6F, 0x41CE7A, 0x35AA65, 0x989996, 0x37E27C,
0x31774F)).await();
} catch (InterruptedException e) {
    // handle InterruptedException
} catch (BandException e) {
    // handle BandException
}

```

11.2.2 iOS

1. Get the current theme from the Band.

```

[self.client.personalizationManager
themeWithCompletionHandler:^(MSBTheme *theme, NSError *error){
    if (error){

```

```

        // handle error
    }
}];

```

2. Set the current theme on the Band.

```

MSBColor *base = [MSBColor colorWithUIColor:[UIColor whiteColor]
error:nil];
MSBColor *highContrast = [MSBColor colorWithUIColor:[UIColor
yellowColor] error:nil];
MSBColor *highlight = [MSBColor colorWithUIColor:[UIColor blueColor]
error:nil];
MSBColor *lowlight = [MSBColor colorWithUIColor:[UIColor
lightGrayColor] error:nil];
MSBColor *muted = [MSBColor colorWithUIColor:[UIColor grayColor]
error:nil];
MSBColor *secondary = [MSBColor colorWithUIColor:[UIColor blackColor]
error:nil];
MSBTheme *theme = [MSBTheme themeWithBaseColor:base
                        highlightColor:highlight
                        lowlightColor:lowlight
                        secondaryTextColor:secondary
                        highContrastColor:highContrast
                        mutedColor:muted];
[self.client.personalizationManager updateTheme:theme
                                completionHandler:^(NSError *error)
{
    if (error){
        // handle error
    }
}];

```

11.2.3 Windows

1. Get the current theme from the Band.

```

try
{
    // get the current theme from the Band
    BandTheme theme = await
bandClient.PersonalizationManager.GetThemeAsync();
}
catch (BandException ex)
{
    // handle a Band connection exception
}

```

2. Set the current theme on the Band.

```

try

```

```
{  
    // create a new Band theme  
    BandTheme theme = new BandTheme()  
    {  
        Base = Colors.White.ToBandColor(),  
        HighContrast = Colors.Yellow.ToBandColor(),  
        Highlight = Colors.Blue.ToBandColor(),  
        Lowlight = Colors.LightBlue.ToBandColor(),  
        Muted = Colors.Gray.ToBandColor(),  
        SecondaryText = Colors.Black.ToBandColor()  
    };  
  
    // set the new theme on the Band  
    await bandClient.PersonalizationManager.SetThemeAsync(theme);  
}  
catch (BandException ex)  
{  
    // handle a Band connection exception  
}
```