Developer World
sonymobile.com/developer

February 2014

How to develop with the new SmartWatch 2 API features – beta SDK

Developer guidelines

SONY
make.believe

Developer World

For the latest technical news, tutorials and development tools go to sonymobile.com/developer.

About this document

This document describes how to use the SmartExtension APIs in order to develop apps for the SmartWatch 2 from Sony. To use this document to best effect you should be familiar with Java programming in an Android environment.

Document conventions

Typographical conventions

Code is written in Courier New font:

```
<action android:name="com.sonyericsson.intent.action " />
```

Entity references to applications and their components are written in italics:

TheApplication, *TheBroadcastReceiver*, *AndroidManifest.xml*

Placeholders to be replaced with real values are written in italics within angle brackets:

<*your class*>

First edition (June 2013)

Document history

Change history		
2014-02-24	Version 1.0.0	First released version

Contents

Introduction

The Smart Extensions API framework is a technology that enables interaction between multiple devices, such as a mobile device and a Sony Smart Accessory, and lets a user perform tasks like controlling a music player, use a calculator or get notifications on the accessory by the use of extensions.

BEFORE YOU START

This is a beta SDK release! This means that development is still in progress and changes to the final SDK may be introduced once it is officially released!

This document assumes you understand the basics, foundation and architecture of the Smart Extensions API framework. A more extensive overview of the Smart Extension API foundation and architecture can be found in the document "Smart Extension APIs architecture" included in this SDK.

Smart Extensions API framework

The mobile device, such as a phone or tablet running Android, acts as the logic core that sends and receives data to a Sony Smart Accessory.

The Sony Smart Accessory, which can be for instance a headset or a smart watch, is specifically designed for receiving and transmitting data to a mobile device that uses the Smart Extensions API framework.

An Android app with added functionality to support the Smart Extensions API framework is called an extension.

The Smart Accessories communicate with the Android device using Bluetooth™.

Keep in mind that an extension is actually running on the mobile device, but it can be set up to communicate and show content, through touch events (touch-based finger activity) or key events (key press activity), on the accessories.

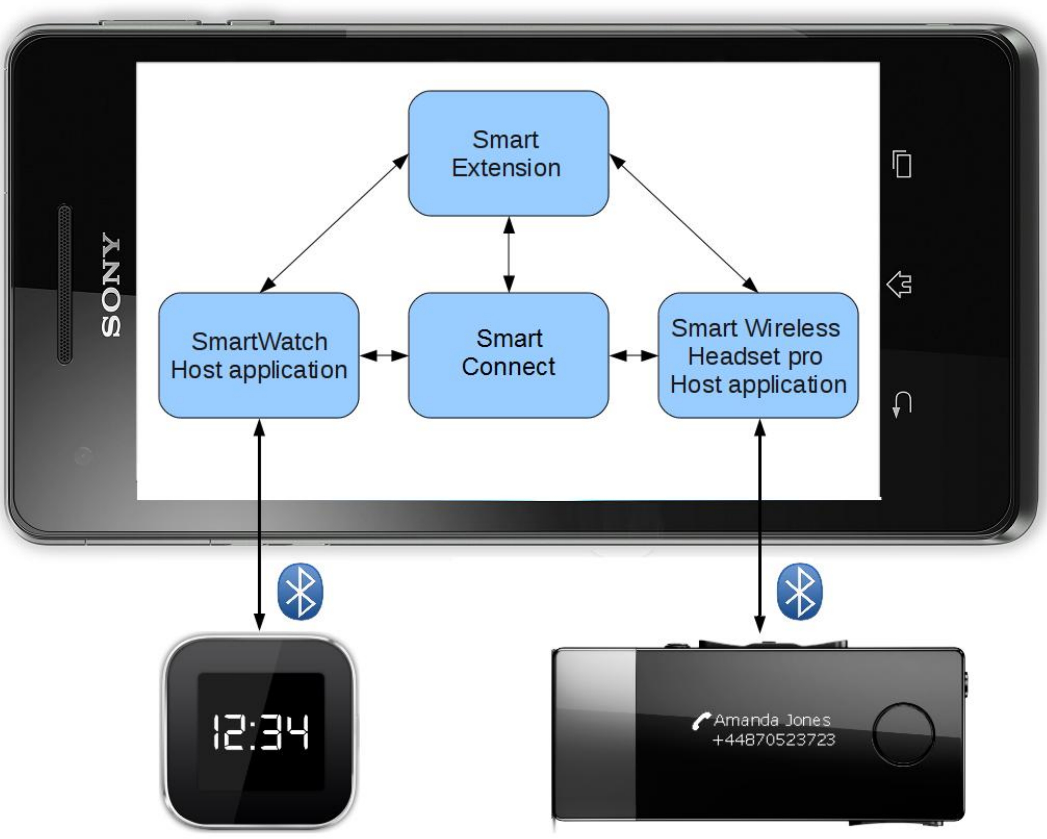
Smart Extension
SmartWatch Host application
Smart Connect
Smart Wireless Headset pro Host application
SONY
12:34
Amanda Jones
+44870523723

Figure 1. Architecture overview of the Smart Extension apps with Smart Accessories examples (original SmartWatch and Smart Wireless Headset pro).

What's new

The new features for the Smart Extensions API framework in this SDK enables the development of Android apps that make use of the following APIs

- Low power mode
- Widgets and clocks

Active Low Power mode

This API makes it possible for a control extension to run in the foreground while the backlight is off on the SmartWatch 2. The extension keeps staying visible and the screen will be able to update contents. You will also be able to interact with the screen as it listens to touch events and key presses.

Widgets and Clocks

The Widget API makes it possible to provide Widgets and Clocks that the user can place on the Accessory Watch Face, i.e. stand by screen (Figure 2).

The widgets can display different layouts depending on the Accessory state, e.g. Default, Powersave, and also when the accessory is disconnected from the mobile device.

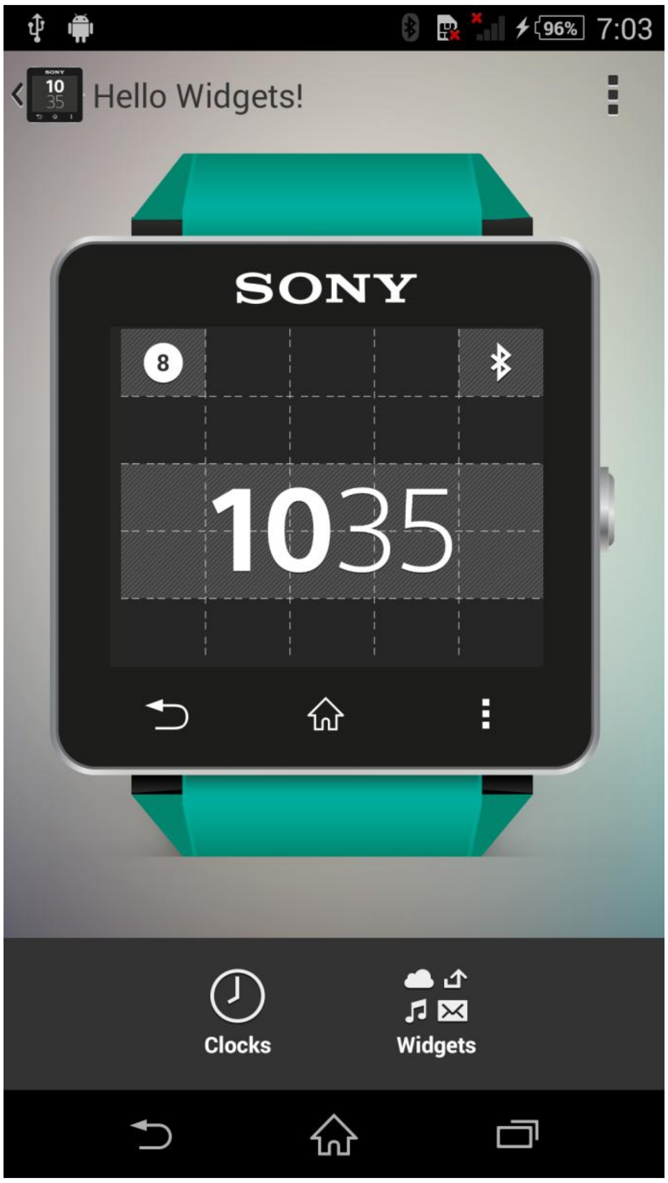

Figure 2. The Edit watch face screen

What's included in the SDK

- Documentation
- APKs for setting up your environment
- Javadocs
- API, libraries and sample code

Developers guide

This SDK includes API libraries and sample application code to help you create extensions for your SmartWatch 2 which includes the new features. These APIs are documented as API references and the sample code is discussed in the Active Low Power Mode Development and Widget and Clocks Development sections.

Components

Here is what you need to build an extension using the Smart Extension API framework including new API features

- Smart Connect app – This is located in the */apks* folder of this SDK
- SmartWatch 2 host app –This is located in the */apks* folder of this SDK
- An extension – An Android app written for the Smart Extension API framework

Get started

1. First, get set up for development:
   - In the terminal, change into the */apks* folder
   - Run:
     ```
     adb install -r SmartConnect_5.6.19.184.apk
     ```
   - Run:
     ```
     adb install -r SmartWatch2_HostApp_1.4.27.apk
     ```
   - Import the library projects located in the */libs* folder into your IDE
     - SmartExtensionAPI
     - SmartExtensionUtils
2. Develop your app
   - Check out the development overview sections and Smart Extensions API documentation
     - Active Low Power Mode Development
     - Widgets and Clocks Development
   - Import and review the samples in the */samples* folder of this SDK, see the Samples section for an overview
   - Browse the API references in the */docs/javadocs* folder of this SDK

3. Test your app by using your mobile device and a SmartWatch 2

Active Low Power Mode Development

This is a feature included as a part of the Control API.

Development

This guide assumes you have a basic understanding of the concepts of Smart Extension development and understand what Active Low Power mode is. You can read more about these topics in the "How to use the Smart Extension APIs" document and the API references in this SDK.

Required setup

Start off with importing the *ActiveLowPower* project. Specifically look at the `RegistrationInformation` class for figuring out what you need.

Registration

In order for your extension to use the low power mode, you are required to override the `supportLowPowerMode()` method in your `RegistrationInformation` class:

```
@Override
public boolean supportsLowPowerMode() {
    return true;
}
```

Adapting to low power mode

Making your extension support Active Low Power mode can be done by using the Control API in the Smart Extensions framework.

When the SmartWatch 2 is running in Low Power mode, the screen only displays content in 3 bit grayscale. We recommend that extensions provide a layout or drawable with elements containing high contrast colours when being run in Low Power mode.

When the SmartWatch 2 switches power mode, e.g. when switching the backlight on or off, while your extension is running, you will receive an `onActiveLowPowerModeChange()` callback. Typically this callback is used as a trigger for your extension to adapt to the new mode.

```
@Override
public void onActiveLowPowerModeChange(boolean lowPowerModeOn) {
    if (lowPowerModeOn) {
        // Adapt your extension layout when the Low Power mode is
        // enabled
    } else {
        // Adapt your extension layout to the Low Power mode is
        // disabled
    }
}
```

Programmatically enabling and disabling Low Power mode

You can control the Low Power mode by using:

```
ControlExtension.setScreenState()
```

To enable Low Power mode, which switches off the SmartWatch 2 backlight, use:

```
setScreenState(Control.Intents.SCREEN_STATE_OFF)
```

To disable Low Power mode, use:

```
setScreenState(Control.Intents.SCREEN_STATE_AUTO)
```

Handling timed out Low Power mode

You will receive an `onActiveLowPowerModeChange()` when the screen switches to Low Power mode through a timeout, e.g. when there has been no user interaction with the screen after a period of time. In this state, any user interaction will turn the backlight on. This automatically sets the screen to SCREEN_STATE_AUTO.

Testing your Active Low Power mode extension

After having built and installed your extension on your mobile device, you will be able to find your extension on the SmartWatch 2 app drawer. Tap your extension in the app drawer to launch your extension and test your functionality.

Widget and Clock Development

This guide assumes you have a basic understanding of the concepts of Smart Extension development and understand what Widgets are. You can read more about these topics in the "How to use the Smart Extension APIs" document and API references.

Registration

In order for your extension to use the newly introduced Widget API changes in this SDK, you are required to override the `getRequiredWidgetApiVersion()` method, in your `RegistrationInformation` class:

```
@Override
public int getRequiredWidgetApiVersion() {
    return 3;
}
```

You need to determine if the widget fits on the accessory screen. Do this by overriding `isWidgetSizeSupported()`:

```
@Override
public boolean isWidgetSizeSupported(final int width, final int
height) {
    // We usually return true here and let the below registration
    // method handle which widgets fit on the screen
    return true;
}
```

You need to return a list of Widget classes to the framework. Do this by overriding `getWidgetClasses()`:

```
@Override
protected WidgetClassList getWidgetClasses(Context context, String
hostAppPackageName, WidgetContainer widget) {
    WidgetClassList list = new WidgetClassList();
    // Register widget if it fits
    if (widget.getMaxWidth() >= StateWidget.WIDTH
        && widget.getMaxHeight() >= StateWidget.HEIGHT) {
            list.add(StateWidget.class);
        }
        return list;
    }
```

Screen refreshing

When a widget is visible on the SmartWatch 2 watch face you will receive an `onStartRefresh()` callback. You will now have the possibility to update the screen content for as long as the widget is visible. When a widget is no longer visible you will receive an `onStopRefresh()` callback.

If you need to refresh the screen on a timed interval there is a utility method, `scheduleRepeatingRefresh()`, where you provide an interval in milliseconds. When the interval limit is reached you receive an `onScheduledRefresh()` callback, where you have the possibility to update screen content.

Touch events

You can receive `onObjectClick()` callbacks when a user touches a layout object in your widget, if you are using XML based layouts and the view has been set as clickable.

You can receive `onTouch()` callbacks when a user touches your widget. This is supported in both XML based and bitmap based layouts.

Implementing Widgets

Handling accessory states

When creating a widget, you need to consider what state the SmartWatch 2 can be in. You will be able to adapt to the new state, e.g. by providing different layouts. You can provide layouts for the following accessory states:

- DEFAULT
- POWERSAVE

You are required to provide a layout for at least the DEFAULT accessory state. If you do not provide a layout for the POWERSAVE state, the DEFAULT one will be used when the SmartWatch 2 enters the POWERSAVE state.

There is a third state, DISCONNECTED. When the SmartWatch 2 enters the DISCONNECTED state, the widget will be removed from the watch face.

When the SmartWatch is in POWERSAVE and DISCONNECTED state, the screen only displays content in 3 bit grayscale. We recommend that extensions provide a layout with elements containing high contrast colours when being run in any of these modes.

Defining layouts

XML based layouts are supported by the SmartWatch 2, read the *SmartWatch products* documentation and see the *ClockWidget* sample for more details on this. It is also possible to provide bitmap based layouts, however, we do not recommend it as it will not be possible to get the same behaviour as with XML based layouts when the watch enters the POWERSAVE and DISCONNECTED state.

Implementing Clocks

The updated Widget API in this SDK includes the possibility of updating widget content when the SmartWatch 2 is disconnected from the mobile device. This specific type of widget needs to provide an additional set of layout resources, as well as explicitly defining itself as supporting the DISCONNECTED state.

Registration

Registration for Clocks is done the same way as for Widgets. Please see the Registration section for Widgets for more information.

Handling accessory states

The updated Widget API in this SDK includes the possibility of updating widget content when the SmartWatch 2 is disconnected from the mobile device.

When creating a clock widget, you can provide three layouts for the following states

- DEFAULT
- POWERSAVE
- DISCONNECTED

You are required to provide a layout for at least the DEFAULT accessory state.

When the SmartWatch 2 enters the POWERSAVE state, you have the opportunity to update your clock widget providing an alternative layout to adapt to this state. If you do not provide an alternative layout, the SmartWatch 2 will use the DEFAULT layout provided and the layour will in this case be displayed as 3 bit grayscale only.

When the SmartWatch 2 enters the DISCONNECTED state, it will use the DISCONNECTED layout provided.

Level lists

In order to start development for a clock widget, you need to define level-lists, containing the resources you want to display when time passes. You have the possibility of defining either drawables or strings in the level lists:

Defining drawables

```
/drawable-nodpi/clock_widget_minutes_msn.xml
<level-list
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/digit_0" />
    <item android:drawable="@drawable/digit_1" />
    <item android:drawable="@drawable/digit_2" />
</level-list>
```

Using strings

```
/values/string-digits.xml
<level-list
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:string="@string/digit_0" />
    <item android:string="@string/digit_1" />
    <item android:string="@string/digit_2" />
</level-list>
```

Defining layouts

You can provide a layout that makes use of newly introduced layout widgets:

- TimeLayout
- TimeView

These layout widgets are specifically made for updating content, e.g. a minute or hour indicator, when the SmartWatch 2 is disconnected from the mobile device.

Use the level-lists in a `TimeView`, which holds the list as its background. You will also have to define what type the `TimeView` will be, e.g. `hours_digit1` or `minutes_digit2`. See API references for the different available types.

Place the `TimeView` in a `TimeLayout` to place the widget according to your needs:

```
<com.sonyericsson.extras.liveware.aef.widget.TimeLayout
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="0.5"
    widgets:gravit="center"
    android:padding="5px" >

    <com.sonyericsson.extras.liveware.aef.widget.TimeView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/clock_widget_minutes_msn"
        widgets:timeType="minutes_digit2" />

    <com.sonyericsson.extras.liveware.aef.widget.TimeView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/clock_widget_minutes_lsn"
        widgets:timeType="minutes_digit1" />
</com.sonyericsson.extras.liveware.aef.widget.TimeLayout>
```

Reference the layout by overriding the `onStartRefresh()` method in a class that extends `BaseWidget`:

```
@Override
void onStartRefresh() {
    // Prepare a bundle with required resources and the resource
    // id in the layouts that shall be updated.
    ...
    bundleNoTouchText(Widget.Intents.EXTRA_LAYOUT_REFERENCE,
    R.id.widget_text_box_view_no_touch);
    bundleNoTouchText.putString(Control.Intents.EXTRA_TEXT,
    mContext.getString(R.string.powersave));
    ...
    // Connect the bundles to the different states
    showLayout(R.layout.default, R.layout.powersave,
    R.layout.disconnected);
}
```

Testing your Widget or Clock

Testing is done by using Smart Connect and the SmartWatch 2 Host App:

1. Start Smart Connect on your mobile device
2. Tap the Devices tab
3. Tap SmartWatch 2 in the list
4. Tap SmartWatch 2 – Edit settings below the SMART ACCESSORY APPLICATION title
5. Tap Edit watch faces below the WATCH FACES title
6. Tap the + sign in the upper right corner
7. Tap Widgets
8. You will be able to find your Widget or Clock in the list shown

Samples

A number of sample apps have been created and included in the */samples* folder of this SDK. These include for instance a Hello World Widget extension, a State Widget extension and ActiveLowPower extension. It is highly recommended that you import these apps into your development environment and get them running on your devices. This will ensure that your development environment is ready to create your own extensions.

A list of the samples and their main use case:

- **ActiveLowPower** – Shows how to enable and disable the Active Low Power mode
- **AdvancedWidget** – Shows how to implement a widget that presents user configured text on the accessory. Also includes implementation to handle user interaction.
- **BackwardsCompatibleWidget** – Shows how to combine a SmartWatch and SmartWatch 2 widget implementation
- **ClockWidget** – Shows how to create different types of Clocks
- **HelloWidget** – Displays a textview as a widget on the watch face
- **StateWidget** – Shows how to implement a widget that makes use of the DEFAULT, POWERSAVE and DISCONNECTED state

Known limitations

- All Widgets and Clocks will currently be located in the Widget category in the Host App

Known issues

- You have to power on the screen to make the SmartWatch 2 switch to DISCONNECTED state after having disconnected from the mobile device.
- The current software is unstable and can crash. Created watch faces may take longer time (more than 15 seconds) to appear on the watch when placing them on the watch face in the Host App.
- If you provide an incorrect number of drawable or string resources when creating a Clock, the watch might not show the Clock on the screen, or even crash.
- Some issues may occur, such as extensions not starting. In order to avoid this situation, remove the service `com.sonyericsson.extras.liveware.extension.util.TunnelService` and the intent filter connected to it from the AndroidManifest file of the extension.
- In some cases your installed widget may not show up in the widget list or the watch. If this occurs, as a last resort, follow these steps:
  1: Clear data in the Smart Connect app
  2: Force close the SmartWatch 2 app
  3: Reset the SmartWatch 2 HW by tapping Settings -> Reset SmartWatch
  4: Pair the SmartWatch 2 with your mobile device

Trademarks and acknowledgements