developer WORLD

Developer guidelines:

SmartWatch specifications

January 2012

Sony Ericsson
make.believe

Document history

Version		
January 2012	First released version	Version 1

Sony Ericsson Developer World

For the latest Sony Ericsson technical documentation and development tools, go to http://developer.sonyericsson.com.

This document is published by:

Sony Ericsson Mobile Communications AB,SE-221 88 Lund, Sweden

www.sonyericsson.com

First released version (January 2012)

Table of contents

1. Before you start

1.1 About this document

This document contains hardware and software specifications for the SmartWatch accessory, describing its capabilities and characteristics. It is intended for SmartWatch extension developers.

To make the best of this document, you should be familiar with the concepts described in the *Smart Extension API specification* document. We also recommended you to look at the Sample Extensions described in the *Tutorial Sample Extensions* document.

User interface guidelines can be found in *Extensions Application UI guidelines.*

These documents can also be downloaded from the Sony Ericsson Developer World site, http://developer.sonyericsson.com

1.2 Typography

Code is written in Courier font:

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

References in text to applications and their components are written in italics: *SampleControlSmartWatch.java*, *AndroidManifest.xml*

References to Intents defined in the *Smart Extension API documentation* are written in upper case: EXTENSION_REGISTER_REQUEST_INTENT

2. SmartWatch

SmartWatch is a touch screen accessory which uses Bluetooth to communicate with the phone. It is equipped with:

- Vibrator
- Accelerometer
- Button (not accessible to applications)

Figure 1 SmartWatch

These features are not accessible when developing Smart Extensions for SmartWatch:

- LED
- Keypad
- Text to speech

The SmartWatch has two user interface levels with extensions, the application level and the widget level.

The application level (see Figure 1) shows all extensions that uses the Control or Notification API. We do not recommend that you use both the notification and the control API in an extension, because then it will not be possible to specify what happens when a user taps the application icon.

The widget level shows extensions that use the Widget API. For extensions that use the Notification API, but not the Widget API, the host application automatically generates a widget. Widgets for notification extensions are disabled by default, but can easily be enabled in the host application settings.

2.1 Display

The SmartWatch display is 128 pixels wide, 128 pixels high and has 65536 colours. The touch screen is limited to 9 distinct touch areas in a 3 x 3 matrix.

A control extension is in control of the entire display. The images are sent as RGB565 (no alpha) to the accessory, so it is recommended to use Bitmap.Config.RGB_565 for the bitmaps drawn by the extension. Images from other sources should not have a transparent background.

When showing a widget, the accessory renders the status bar and then the area controlled by the extension is 128 pixels wide and 110 pixels high. To be able to cache widget images, the host application reduces the colour depth of widget images to 1024 colours.

This extension sends its images to the host application which converts the images to a format supported by the accessory. Images sent to the host application should be in .png format.

For a control extension, the SmartWatch supports both tap touch (through *CONTROL_TOUCH_EVENT_INTENT* or `ControlExtension.onTouch()`) and swipe motions (through *CONTROL_SWIPE_EVENT_INTENT* or `ControlExtension.onSwipe()`).

For a widget extension, only tap touch is available (through WIDGET_ONTOUCH_INTENT or `WidgetExtension.onTouch()`). A too short swipe motion might be interpreted by the accessory as a tap event. To avoid that a widget reacts on those events, we recommend that a widget only reacts to touch events that are inside the 80 by 80 pixels square in the display centre. You achieve this by checking that a touch point is inside the rectangle defined in *SmartWatchConst.ACTIVE_WIDGET_TOUCH_AREA*.

For a code example, please see the *Sample Widget* in the *Tutorial Sample Extensions*. If the widget has a dedicated button outside the active touch area like the *Music Player – Smart Extras™,* where the play pause state can be toggled by tapping the play pause icon, taps on that button should of course be handled even though they are outside the active touch area.

2.2 Battery

SmartWatch is a high performance smart accessory, and it is therefore important to use the battery power wisely. The host application and the SDK include some functionality to preserve of the battery.

It is also important that the extension follow these recommendations:

- Don’t use too much white colour in your images.
- Control Extension: Don’t send full screen images when not necessary (send partial screen-updates instead).
- Control Extension: Avoid using *Control.SCREEN_STATE_ON.*
- Control Extension: When using sensor; use SensorInterruptMode.SENSOR_INTERRUPT_ENABLED to only receive sensor data when the sensor has changed.

2.3 Accelerometer

The accelerometer in SmartWatch is available to the control extension that is currently started.

If the control is paused (for example due to that the screen timeout elapses, the power button is pressed or another control extension is started) the sensor is also stopped.

If the control extension wants to continue receiving sensor information it should use SCREEN_STATE_ON, SCREEN_STATE_OFF or SCREEN_STATE_DIM.

2.4 Background process

All extensions are run in a service. Android treats services as background processes. Android has a limitation on the number of background processes that can be run simultaneously. When this limit is exceeded, the Android system stops the least recent used process. This might happen even if the extension is currently showing on the display, since the Android system is not aware of this.

Normally this is not a problem for extensions that are only shown for a short while and that the user is directly interacting with. The helper classes in *SmartExtensionUtils* also contain some functionality to help an extension recover if it was stopped when the user was interacting with an event.

For extensions running for a longer time and which should not be interrupted, you can use `Service.startForeground()` to make the system consider it to be something the user is actively aware of. The side effect is that a notification is shown in the status bar as long as the service is in foreground state.

2.5 Market description

Users can search for compatible extensions via the host application. Your market description should end with "`\n\nLiveWare™ extension for SmartWatch`" to ensure that your extensions are found.

Trademarks and acknowledgements