developer WORLD

Tutorial

November 2010

Creating LiveView™ plug-ins

for Android™ phones

Version 1.0

Sony Ericsson

Preface

Document history

Date	Version
November 2010	Version 1.0

Purpose of this document

The purpose of this document is to illustrate how to create plug-ins for the LiveView™ micro display on Android version 2.1 (Eclair).

The document is divided into three parts. The first part describes the LiveView™ micro display, LiveWare™ application, LiveView™ manager and LiveView™ plug-ins. The second part is the actual tutorial, describing how to create plug-ins for the LiveView™ micro display. The third part includes some LiveView™ best practices.

This Tutorial is published by:

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

www.sonyericsson.com/

November 2010

Sony Ericsson Developer World

At www.sonyericsson.com/developer, developers find the latest technical documentation and development tools such as phone White papers, Developers guidelines for different technologies, Getting started tutorials, SDKs (Software Development Kits) and tool plug-ins. The Web site also features news articles, go-to-market advice, technical support and much more. For more information about these professional services, go to the Sony Ericsson Developer World Web site.

Document conventions

Terminology

AIDL	Android Interface Definition Language An IDL language used to generate code that enables two processes on an Android-powered device to talk using interprocess communication (IPC).
Plug-in	An plug-in is a set of software components that adds specific capabilities to a larger software application. If supported, plug-ins enable customising the functionality of an application.

Typographical conventions

Code is written in Courier font: `<java>..</java>`.

Trademarks and acknowledgements

Licensing

Contents

Introduction

You can create LiveView™ plug-ins which work as plug-ins of already exicting applications or as standalone applications, both with the purpose to feed information to the LiveView™ micro display. The easiest way to get started with a new plug-in, is to use the plug-in template project that is provided with the Sony Ericsson LiveView™ SDK. You can create either announce plug-ins or sandbox plug-ins which you can run on an Android phone paired with a LiveView™.

For more information on how to create plug-ins, see “How to create plug-ins” on page 15. For more information about a the different plug-ins, see “Plug-in types” on page 9. For information about an announce plug-in project, see “Announce plug-in example project” on page 25. For information about a sandbox plug-in project, see “Sandbox plug-in example project” on page 28.

LiveView™ micro display

LiveView™ micro display is an accessory that mirrors and displays the events that happen in your phone, so you never miss what is going on. On your LiveView™ screen, you can view plug-ins and notifications such as text messages, multimedia messages, incoming calls, calendar event reminders, updates from friends on Facebook™, and tweets. LiveView™ uses a Bluetooth™ connection to communicate with your phone.

Overview

The LiveView™ micro display has two hardware keys and a touch area. You can use either a clip or a wrist strap to wear the LiveView™ micro display.

1	Power on/off, Pairing mode, Display on/off, Notification LED
2	Back, Select, Media player, Display on
3	Next
4	Down
5	Previous
6	Up

Note! *If you create plug-ins, you can customise number 2, 3, 4, 5 and 6 (See illustration above).*

Notification LED

The Notification LED (Light-emitting diode) provides information on the status and notifications of LiveView™.

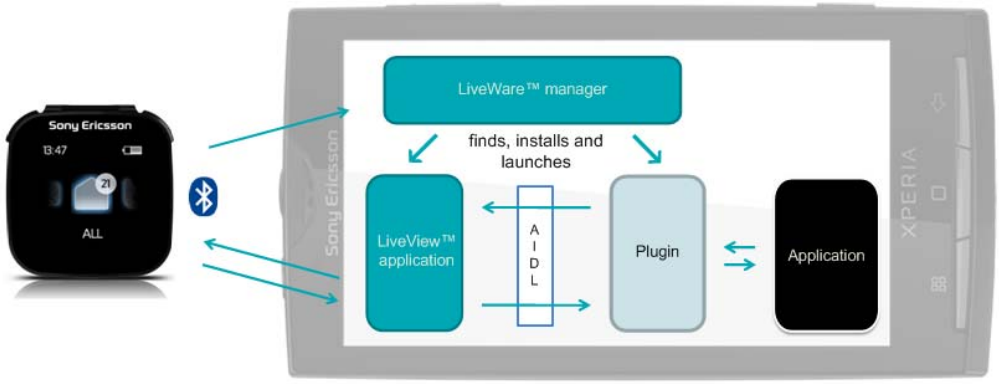
Flashing blue	Incoming call
Flashing red	Low battery
Flashing green	A new notification is available
Flashing blue and green alternately	Pairing is underway
Red	The battery is charging, and the battery level is between low and full
Green	The battery is fully charged

LiveView™ User guide

For more information about the LiveView™ micro display, a User guide is available at www.sonyericsson.com/support.

Architecture

All plug-ins communicates with the LiveView™ device via the use of a defined interface to the LiveView™ service installed on the phone. The LiveView™ service communicates with the LiveView™ device via Bluetooth.



LiveView™ plug-ins

A plug-in can be implemented as a part of an application, and would, when launched on the phone, send information to the LiveView™ device. An example could be an extended screen for the phone camera application, showing a snapshot of the just taken photo. On the other hand, a plug-in can be a standalone application which whole purpose would be to feed information to the LiveView™ device, for example a plug-in that fetches the latest information from a specific blog. LiveView™ plug-ins can be created either as announce plug-ins, or as sandbox plug-ins. For more information about the different plug-ins, see “Plug-in types” on page 9.

LiveView™ application

The LiveView™ application consists of a GUI and a service. The GUI shows the user which plug-ins that are installed, and presents the plug-in preferences, for example enabling or disabling the plug-ins as well as setting custom preferences defined by the implemented plug-in. For more information about the LiveView™ application GUI, see “LiveView™ application GUI” on page 21.

The LiveView™ service is the communication hub, talking to the LiveView™ device as well as handling the communication with the plug-ins via broadcasts, intents and direct RPC communication. For more information see “Plug-in components” on page 11.

There are two interfaces to consider for the LiveView™ service, the *IPluginService* and the *IPluginServiceCallback*. The *IPluginService* interface defines the methods to call that will interact with the LiveView™ service and consequently the LiveView™ device. The *IPluginServiceCallback* interface defines the methods that the LiveView™ service

will use to communicate with the plug-in. For more information about *IPluginService* and *IPluginServiceCallback*, see “IPluginService” on page 22 and “IPluginServiceCallback” on page 23.

LiveWare™ manager

LiveWare™ manager is a framework for Android phones that manages applications and related settings for smart accessories, for example LiveView™. LiveWare™ manager requires Android version 2.0 or later. LiveWare™ manager needs to be installed on your phone before using your LiveView™. LiveWare™ manager is required in order to:

- Define what application should be started when a specific accessory such as Live View™ is connected to the phone

- Download and install applications that are defined for a specific accessory from Android Market

- Download and install plug-ins for applications from Android Market

In order for the LiveWare™ manager to find the plug-in, you need to define certain information in the Android project manifest description. For more information, see “Step 7 - Publishing to Android Market” on page 19.

Plug-in types

Announce plug-in Sandbox plug-in

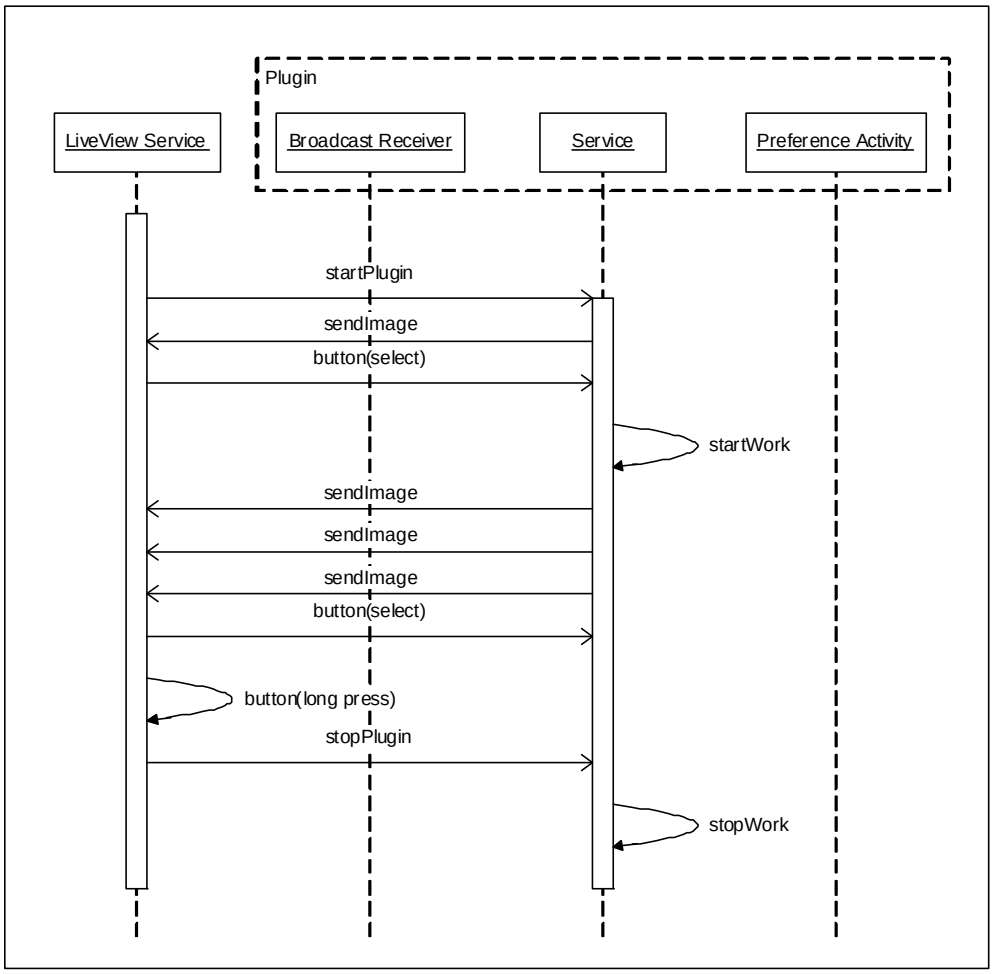
Announce plug-in

Announce plug-ins can send announcements to the LiveView™ device. This is done in the same way as the predefined announce features, like Facebook and Twitter. An announce plug-in example is available in section “Announce plug-in example project” on page 25.

Sandbox plug-in

The sandbox plug-in can take complete control of the LiveView™ device by sending images to it and control its ability to vibrate and display different colors on the LED. All user activities are propagated to the plug-in, so that it can take appropriate actions. For example, when the user presses buttons on the LiveView™ device, events are sent to the sandbox plug-in. The plug-in can then handle the events, for example by sending a specific image to the LiveView™ device.

The diagram below shows an example of how a sandbox plug-in works. The LiveView™ service starts the plug-in when the user has entered the plug-in on the LiveView™ device, then button press events are sent to it. The plug-in responds by starting a worker that sends images. All button events are sent to the plug-in. A long-press on the right button of the LiveView™ device indicates to the LiveView™ application that the user has left the sandbox mode. A *stopPlugin* call is made to the sandbox plug-in, and any work is stopped.

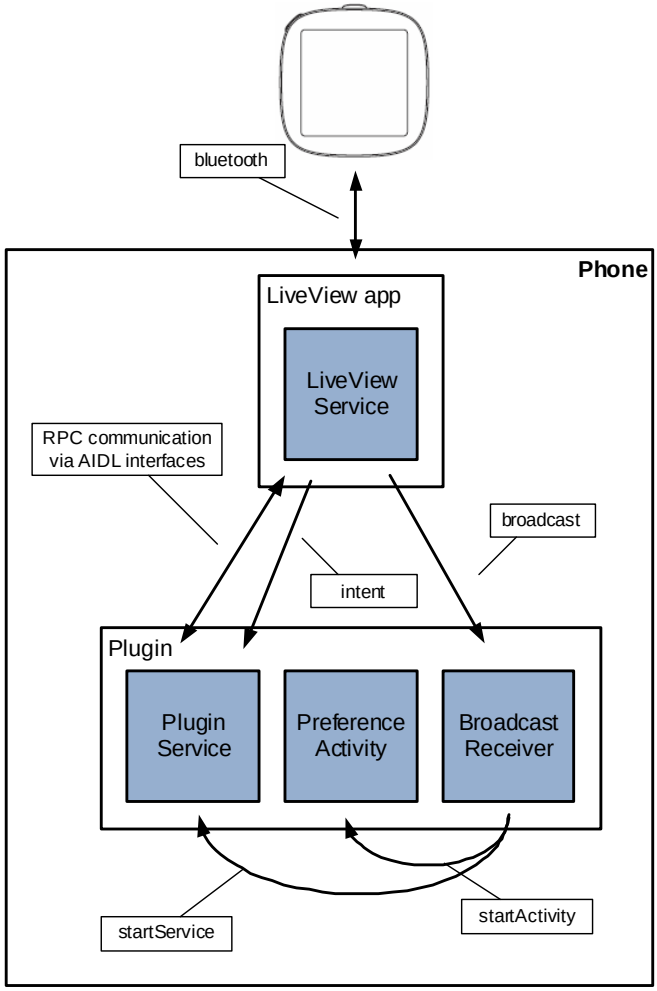


For more information about a sandbox plug-in see “Sandbox plug-in example project” on page 28.

Plug-in components

A plug-in needs to define three components to be able to function correctly with the LiveView™ micro display. It is the *Service*, *Preference Activity* and the *Broadcast Receiver*. For more information about the components, from what is already mentioned in this section, see “Step 2 - Implement components” on page 15.

The architecture overview for the LiveView™ application and plug-ins is shown in the illustration below.

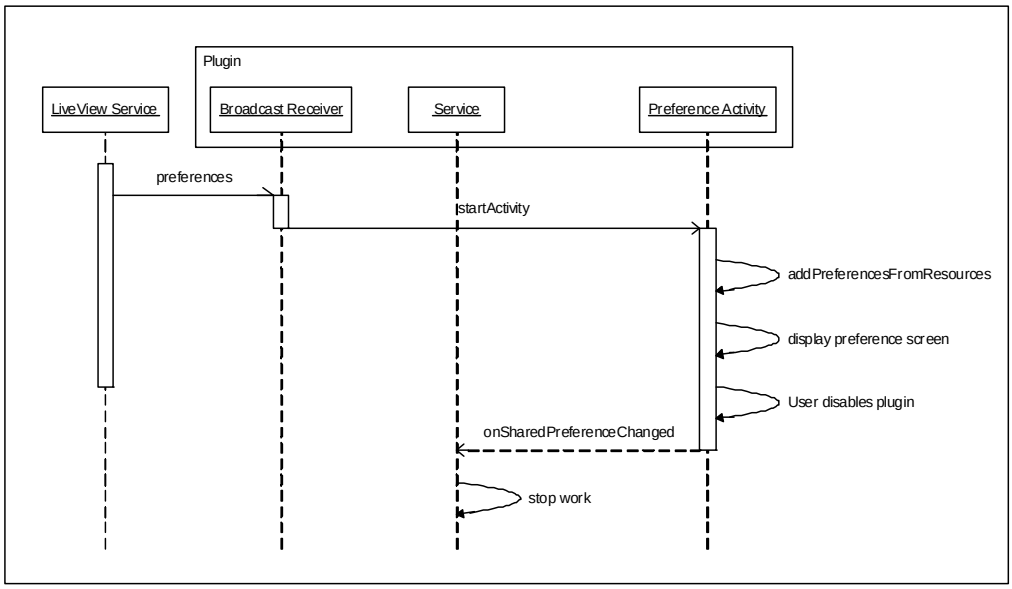


Service

The Service is the worker of the plug-in. It should handle communication with the LiveView™ service as well as define the total lifecycle span of the plug-in.

Preference Activity

The Preference Activity defines the settings that the user will be able to update. Settings could include if the plug-in is enabled, updated sequences for announcements, and account handling. All plug-ins are responsible for their own preferences which are launched as a new activity from the LiveView™ application.



The diagram above shows an example of how the user disables plug-in via preferences. The user is shown the preferences for the plug-in and the user chooses to disable the plug-in. In this sequence, the service implements a shared preference listener and is therefore notified of the change. The plug-in service stops.

Broadcast Receiver

The primary functionality of the Broadcast Receiver is to catch the LiveView™ application plug-in broadcast and use that to determine which, if any, of the two entities (install plug-in or launch its Preferences Activity) to launch. The broadcast has at least one additional data field in the key *CMD* which will equal either *start* or *preferences*.

- *start* - the Broadcast Receiver should start its service which in turn should notify the LiveView™ application that it is installed on the phone. The *start* command is sent by the LiveView™ application when the user has requested that plug-ins registers themselves.

- *preferences* - it should get the second additional data in the key *pluginName*. If that value corresponds to the plug-ins own name it should launch its Preference Activity. The *preferences* command is sent when the user has requested its preferences from the LiveView™ configuration screen.

Registration of a plug-in

The registration of a plug-in is triggered by a request from the LiveView™ application, using a pre-defined broadcasted intent which all plug-ins need to respond to. For more information about broadcast intent, see “Broadcast intent” on page 30.

The registration can be called upon several times for the same plug-in, but will only result in one actual instance being registered. When registering the plug-in, the LiveView™ service is provided with the following:

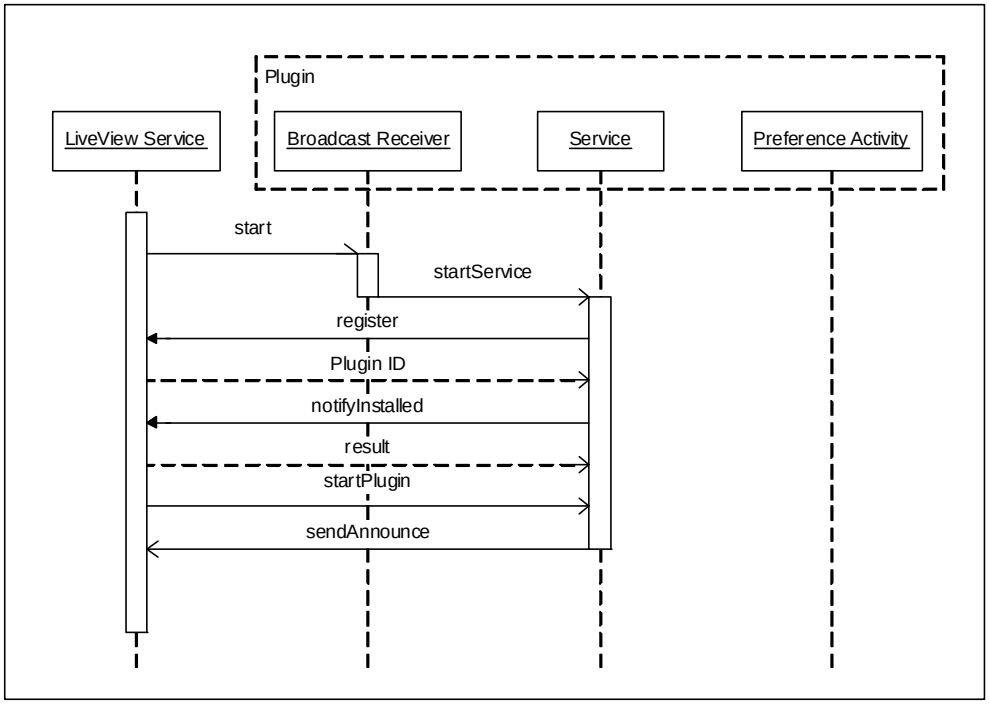
- An instance of the callback interface

- A path to the image it wants to have displayed in the menu

- A name of the plug-in (also for the menu)

- If the plug-in is an announce plug-in or a sandbox plug-in

- The plug-in package name (this is used for un-installation purposes)



The diagram above shows an example of how the user register an announce plug-in. The LiveView™ service sends a broadcast message containing a *start* command to all plug-ins. The plug-in Broadcast Receiver is invoked and starts the plug-in Service. The Service registers itself to the LiveView™ service and also notifies the specific Service start-intent for the LiveView™ service to use for specific starts and stops. The plug-in will receive the *startPlugin* call on registration, and if the LiveView™ service is restarted. The plug-in service starts its dedicated work.

Finding the installed plug-in

When a plug-in has been found by the LiveView™ application, the LiveView™ device will present a new plug-in icon in the scrolling icon list. By tapping this icon you will find the new installed plug-in.

Plug-in menu

Plug-in sandbox

Depending on what type of plug-in you have installed, the LiveView™ device will act according to its type:

- Announce - show all announcements sent by the plug-in

- Sandbox - enter sandbox mode and start plug-in

When in sandbox mode, you will leave the plug-in by pressing and holding down the action button.

Publishing to Android Market

There are some requirements for publishing your plug-in to Android Market. For more information, see “Step 7 - Publishing to Android Market” on page 19.

How to create plug-ins

Step 1 - Download developer tools

The easiest way to get started with a new plug-in, is to use the plug-in template project that is provided with the Sony Ericsson LiveView™ SDK. You can create either announce plug-ins or sandbox plug-ins which you can run on an Android phone paired with a LiveView™ device. For more information about the different plug-ins, see “Plug-in types” on page 9. Below is a table of the developer tools you need.

Tool	Information
Eclipse version 3.4 or 3.5	http://www.eclipse.org/downloads
Android SDK	http://developer.android.com/sdk/index.html
ADT plug-in for Eclipse	http://developer.android.com/sdk/eclipse-adt.html

Step 2 - Implement components

In order for the plug-in to function with the LiveView™ device, you need to implement a *Service*, a *Preference Activity* and a *Broadcast Receiver*. For more information about the three components, see “Plug-in components” on page 11.

Note! *If you decide to use the plug-in template project, the above information is already implemented for you, but has to be configured.*

Service

Your service needs to extend the *android.app.Service* class and implement four of its methods. The following four methods of the *android.app.Service* class have to be extended:

`onCreate`
This method does initializations like moving icons to the phone.

`onStart`
This method connects to the LiveView™ service and other start activities.

`onBind`
This method needs to be implemented in order to bind the service to the LiveView™ service, but does not need to contain any logic.

`onDestroy`
This method unregisters the plug-in and un-binds it from the LiveView™ service.

Besides this, the Service needs to define instances of *ServiceConnection*, to bind with the LiveView™ service, and define an *OnSharedPreferenceChangeListener*, to register in the *PreferenceManager*. The *OnSharedPreferenceChangeListener* will tell the service when the user changes any settings of the plug-in, which it then can respond to.

Preference Activity

You need to define a class that extends the *android.preference.PreferenceActivity* class. The class needs to implement the method *onCreate* and should include a call to the method *addPreferencesFromResource*, which adds the defined preference resources to the shared preferences of the current user session. Your resources need to be defined in a preference resource, that is *xml/preferences.xml*. The xml should at least contain a checkbox, for the user to check/uncheck in order to enable or disable the plug-in.

Broadcast Receiver

The LiveView™ service will try to communicate with your plug-in by sending broadcast messages with the intent

```
com.sonyericsson.extras.liveview.LAUNCH_PLUGIN
```

You need to define a class that extends the *android.content.BroadcastReceiver* class and implements the *onReceive* method.

The incoming broadcast intent will contain two extras that is *CMD* and *pluginName*. In addition to the obvious *pluginName*, the *CMD* will contain either the string *start* or *preferences* and will have the following intentions.

Start
This should start the plug-in service by for example using *context.startService(intent)*.

Preferences
This should start the preference activity by for example using *context.startActivity(intent)*.

Step 3 - Check framework classes

The framework that is the foundation of the plug-in template project consists of a number of classes that takes care of the communication with the LiveView™ application, as well as giving you access to some utility classes.

AbstractPluginService

This is the main class of the framework. It sets up the communication with the LiveView™ application. It implements:

- Bind with LiveView™ application

- *ServiceConnection.onServiceConnected* which is where the registration of the plug-in is done

- *ServiceConnection.onServiceDisconnected* which is where the connection to the LiveView™ application is terminated and the service is stopped

- *OnSharedPreferenceChangeListener.onSharedPreferenceChanged* which is when the user changes any of the settings of the plug-in via the LiveView™ application

These methods are extendable in your service via abstract method implementations. When you create a new plug-in, your service needs to extend the *AbstractPluginService*.

LiveViewAdapter

This is an abstraction of the LiveView™ application interface which encapsulates the methods you can call.

PluginConstants

This class holds common constants that are used throughout the project.

PluginPreferences

This class handles the Preference Activity call from the LiveView™ application.

PluginReceiver

This class is the Broadcast Receiver for the broadcasts sent from the LiveView™ application. It will in its turn start the preferences, *PluginPreferences*, or the service which is extending the *AbstractPluginService*. For more information about the class, see an example in “Announce plug-in example project” on page 25.

PluginUtils

This is a utility class that implements methods for communicating with the LiveView™ application, for example to send an image or send an announcement.

Step 4 - Implement Service methods

If you use the plug-in template project your Service needs to extend the *AbstractPluginService* class, as is shown in the class *TemplateService*. You can rename the *TemplateService* class and use it as a skeleton, since it already has all method stubs needed. Follow the instructions that are included as comments in the code. There you will find methods that you can extend to get the plug-in behaviour you want. You need to implement the method *isSandboxPlugin* to state what type of plug-in it is.

LiveView adapter

To communicate with the LiveView™ application, you can use the global instance *mLiveViewAdapter*.

LiveView callback interface

To be able to handle events coming from the LiveView™ device, there are a number of abstract methods that you need to implement in your service.

```
protected abstract void startPlugin();
```

This method is called when the LiveView™ application wants to start the plug-in. For a sandbox plug-in, this would be called when the user enters the plug-in on the LiveView™ device.

```
protected abstract void stopPlugin();
```

This method is called when the LiveView™ application wants to stop the plug-in. For a sandbox plug-in, this would be called when the user leaves the plug-in on the LiveView™ device by pressing and holding down the action button.

```
protected abstract void button(String buttonType, boolean
doublepress, boolean longpress);
```

Sandbox mode only. This method is called when the user presses any of the buttons on the LiveView™ device.

```
protected abstract void displayCaps(int displayWidthPx, int
displayHeigthPx);
```

This method is called by the LiveView™ application to indicate the capabilities of the LiveView™ device.

```
protected abstract void onUnregistered() throws RemoteException;
```

This method is called by the LiveView™ application when the plug-in has been kicked out by the framework, for example by uninstallation.

```
protected abstract void openInPhone(String openInPhoneAction);
```

This method is called when the user taps **open in phone** on the LiveView™ device. You could for example open an URL in the browser. For more information about this, see “Announce plug-in example project” on page 25.

All methods are forwarders for the methods of the callback interface. For more information, see “IPluginServiceCallback” on page 23.

Step 5 - Update resources for Preference Activity

In the plug-in template project, there are two plug-in resources to consider, the *xml/preferences.xml* and the *values/strings.xml.* The *xml/preferences.xml* resource should contain your plug-in preferences. A checkbox resource for handling the enabling and disabling of the plug-in is already implemented in the template project. The *values/strings.xml* resource contains resource strings. There are three defined strings that you need to fill out:

- pluginname

- intent_preferences - the intent used to start your preference activity

- intent_service - the intent used to start your plug-in service

Step 6 - Implement Broadcast Receiver

The Broadcast Receiver is already fully implemented when using the plug-in template project.

Step 7 - Publishing to Android Market

When you have created your plug-in, visit the http://developer.android.com/guide/publishing/publishing.html to find requirements for publishing to Android Market defined by Google.

Requirements on the Android Market description

The description that should be used on Android Market has to contain the following string: "Extends:<HostApplication>.". This means that your plug-in has to contain the following:

```
Extends:com.sonyericsson.extras.liveview.
```

Requirements on the manifest description

The application description should contain the String "Extends:<HostApplication>." where <HostApplication> should be the same as the PackageName of the application that has the plug-in API.

The package name of the LiveView™ application is:

```
com.sonyericsson.extras.liveview
```

Sandbox plug-in example:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
  <application android:description="Application plug-in for Sony
Ericsson notifiers.

    For: LiveView
    Extends:com.sonyericsson.extras.liveview.
  ">
</application>
</manifest>
```

Best practices

Sending images

Sending full screen images when not necessary should be avoided as it slows down both the update frequency of the screen and the battery life of the device.

If possible, use the *sendImageAsBitmap* or *sendImageAsByteArray* since that will save the framework the additional time to read and decode a file from the file system. For more information about the two methods, see examples in the “IPluginService” on page 22.

For information about image restrictions, see “Image restrictions and recommendations” on page 24.

Battery drain and Bluetooth stability

If your plug-in has a high image update ratio, be sure to make use of the callback function *screenMode* to limit the Bluetooth traffic to the LiveView™ device. This will limit the drain of the LiveView™ device and put less stress on the Bluetooth connection. For more information about the *screenMode*, see an example in the section “Sandbox plug-in example project” on page 28.

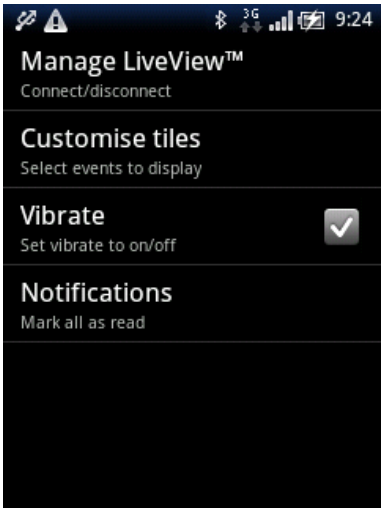
Misbehaving plug-ins

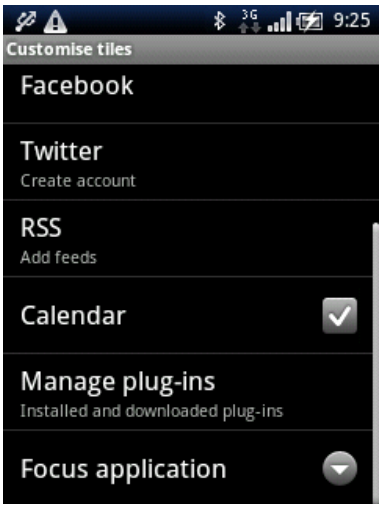
Since the plug-in API is open, a misbehaving plug-in can tear down the LiveView™ application as well. To avoid this, it is important that plug-in developers make sure not to flood the AIDL interface with too many calls.

Appendix

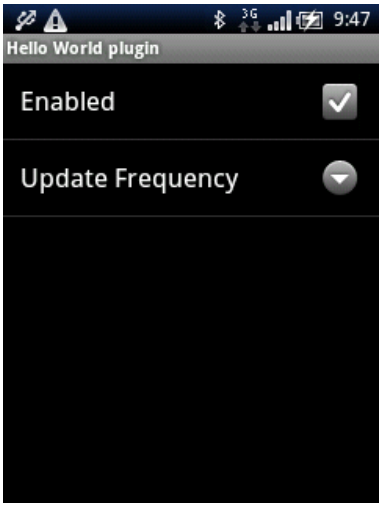
LiveView™ application GUI

The following set of images shows the LiveView™ application GUI that runs on the phone and where you will find your plug-in preferences. The first image shows the startup screen of the LiveView™ application. If you tap **Customise tiles**, you will enter the setup screen for the installed applications. If you tap **Manage plug-ins**, you will enter a screen where a list of the installed and downloaded plug-ins is presented.





In this example, only the HelloWorld plug-in is installed. The HelloWorld plug-in has two preferences, one is if the plug-in should be enabled or not, and the other is the update frequency of the "Hello World" announcements sent to the LiveView™ device.



IPluginService

```
int register(IPluginServiceCallbackV1 cb, String imageMenu,
String pluginName, boolean selectableMenu, String packageName);
```

Called by plug-in to register itself. The plug-in implementation needs to define a callback stub as well as the plug-in name, to provide in the call to the LiveView™ service. *ImageMenu* is the reference to an image that will be displayed in the LiveView™ device for the specific plug-in. *SelectableMenu* indicates if the plug-in should receive information from the LiveView™ device, for example when a button is pressed, or if the plug-in just provides notifications to the LiveView™ device. The *packageName* is the package name of the plug-in. The *register* method function returns a unique id for the specific plug-in, as it has been registered in the internal LiveView™ database.

```
void unregister(int id, IPluginServiceCallbackV1 cb);
```

This method unregisters the plug-in from the LiveView™ database. The *id* is the unique plug-in id returned in the register method.

Note! *This method should basically just be used for unregistering the plug-in when for example the plug-in has been uninstalled on the Android device.*

```
void sendAnnounce(int id, String imageAnnounce, String header,
String body, long timestamp, String openInPhoneAction);
```

This method sends an announcement to the LiveView™ device. The *id* is the unique plug-in id returned in the register method. *ImageAnnounce* is the absolute path to the image on the Android device. *Header* and *body* define what message should be shown. *OpenInPhoneAction* should define the actual action taken if the LiveView™ device user wants to open the announcement in the phone. For example, opening an URL in the browser of the phone.

```
void sendImage(int id, int x, int y, String image);
```

This method sends an image stored on the phone to the LiveView™ device. The *id* is the unique plug-in id returned in the register method. *X* is the number of pixels from the left side of the screen. *Y* is the number of pixels from the top of the screen. *Image* defines the absolute path to the image on the Android device.

For more information about image restrictions and format, see “Image restrictions and recommendations” on page 24.

```
void sendImageAsBitmap(int id, int x, int y, Bitmap bitmapData);
```

This method sends a bitmap to the LiveView™ device. The *id* is the unique plug-in id returned in the register method. *X* is the number of pixels from the left side of the screen. *Y* is the number of pixels from the top of the screen. *BitmapData* is the actual bitmap data.

For more information about image restrictions and format, see “Image restrictions and recommendations” on page 24.

```
void clearDisplay(int id);
```

This method clears the LiveView™ display. The *id* is the unique plug-in id returned in the register method.

```
int notifyInstalled(String launcherIntent, String pluginName);
```

This method is used to notify the actual start intent of the plug-in service.

```
void ledControl(int id, int rgb565, int delayTime, int onTime);
```

This method controls the LED. The *id* is the unique plug-in id returned in the register method. *Rgb565* is the color to use. *DelayTime* defines if it should be a delayed action. *OnTime* defines if the color should change on a given time.

```
void vibrateControl(in int id, int delayTime, int onTime);
```

This method controls the vibrator. The *id* is the unique plug-in id returned in the register method. *DelayTime* defines if it should be a delayed action. *OnTime* defines if the device should vibrate on a given time.

```
void sendImageAsBitmapByteArray(int id, int x, int y, byte[]
bitmapByteArray);
```

This method sends a bitmap byte array to the LiveView™ device. The *id* is the unique plug-in id returned in the register method. *X* is the number of pixels from the left side of the screen. *Y* is the number of pixels from the top of the screen.*BitmapByteArray* is the byte array.

For more information about image restrictions and format, see “Image restrictions and recommendations” on page 24.

```
void screenOff(in int id);
```

This method is used to put the screen in power save mode.

```
void screenDim(in int id);
```

This method is used to put the screen in dimmed mode.

```
void screenOn(in int id);
```

This method is used to wake the screen from power save mode.

```
void screenOnAuto(in int id);
```

This method is used to give the control of the power save functionality back to the LiveView™ device.

IPluginServiceCallback

```
void startPlugin();
```

This method is used by the LiveView™ service to start the plug-in. The plug-in workers, if any, in the plug-in should be started.

```
void stopPlugin();
```

This method is used by the LiveView™ service to stop the plug-in. The plug-in workers, if any, in the plug-in should be stopped.

```
String getPluginName();
```

This method is called by the LiveView™ service to fetch the plug-in name.

```
void openInPhone(String openInPhoneAction);
```

This method is called by the LiveView™ service to indicate to the plug-in that it should handle the action that is provided in the *openInPhoneAction* parameter. The action could be to open a browser, or a local application. The *openInPhoneAction* is provided by the plug-in in the callback interface.

```
void onUnregistered();
```

The plug-in has been kicked out of the framework.

Note! *When this happens for a sandbox plug-in, the user has left the plug-in on the LiveView™ device, and the plug-in has no control over the screen any more.*

```
void displayCaps(int displayWidthPixels, int
displayHeigthPixels);
```

This method is used by the LiveView™ service to tell the plug-in the screen boundaries.

```
void button(String buttonType, boolean doublepress, boolean
longpress);
```

This method is called by the LiveView™ service to indicate that a button has been pressed on the LiveView™ device.

```
void screenMode(int mode);
```

This method is used by the LiveView™ service to notify to the sandbox plug-in that the screen mode has changed.

0 - screen is OFF, 1 - screen id ON.

Image restrictions and recommendations

Images sent to the Liveview™ device are always palette PNG with 8 bits/pixel, non-interlaced with a palette size of 256 colors. Maximum dimensions:

- Menu icon - 48 x 48 pixels

- Announce icon - 16 x 16 pixels

- Sandbox mode - 128 x 128 pixels (currently the full size of the screen)

- In general, the PNG data has an upper limit of approximately 8k packed

If you send image data in other formats than recommended, the LiveView™ application will convert the image data using its own native implementation. Sending image data with more than 256 colors will force the LiveView™ application to reduce the color data before sending the image to the display. Since the timing is important the color reduction algorithm focuses on speed and not color optimization.

When saving a bitmap file to the file system in Android (using compress), you cannot save it in the format that LiveView™ wants. Thus all images saved to the file system will be reduced by the LiveView™ application. Be sure to use the *Config.RGB565* and that the image has no more than 256 colors. Then the compression will be as fast and lossless as possible.

Announce plug-in example project

The HelloWorld plug-in project implements an announce plug-in that sends notifications to the LiveView™ device with different intervals. The default interval is 60 seconds. The notification can be seen in the LiveView™ device, and you can open a browser on your phone with the *openInPhone* functionality. By managing the HelloWorld plug-in in the LiveView™ application, you can set the interval of your choice.

The plug-in implements a *ListPreference* resource which defines the update interval of the announcements. There is also an array string resource in the res/values directory which defines the values of the different intervals.

Implementation

When the plug-in is instantiated by the LiveView™ application, the *onStart* method is called and a *Handler* is instantiated to handle the interval of the announcements.

```
@Override
public void onStart(Intent intent, int startId) {
  super.onStart(intent, startId);

  // Create handler.
  if(mHandler == null) {
    mHandler = new Handler();
  }
}
```

Then the LiveView™ application will call the *startPlugin* method to indicate to the plug-in that it should now start its work. In the HelloWorld example, the *startPlugin* method will call the mandatory method *startWork*.

```
protected void startWork() {
  // Check if plugin is enabled.
  if(!mWorkerRunning &&
mSharedPreferences.getBoolean(PluginConstants.PREFERENCES_PLUGIN
_ENABLED, false)) {
    mWorkerRunning = true;
    scheduleTimer();
  }
```

```
}
```

The method will check that the worker is not already running, and that the user has enabled the plug-in on the phone. If these conditions are true, an announcement is scheduled to be sent by using the local method *scheduleTimer*.

```
private void scheduleTimer() {
  if(mWorkerRunning) {
    mHandler.postDelayed(mAnnouncer, mUpdateInterval);
  }
}

private Runnable mAnnouncer = new Runnable() {

  @Override
  public void run() {
    try
    {
      sendAnnounce("Hello", "Hello world number " + mCounter++);
    } catch(Exception re) {
      Log.e(PluginConstants.LOG_TAG, "Failed to send image to
LiveView.", re);
    }

    scheduleTimer();
  }

};
```

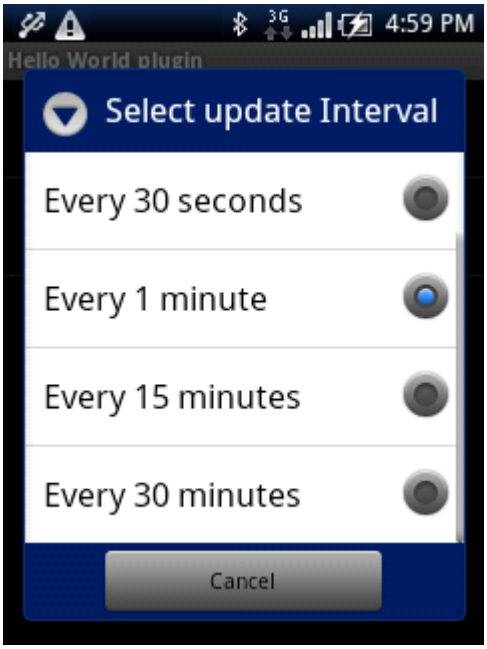
The *schedulerTimer* method makes use of the Handler instance to schedule an announcement with a specific delay in milliseconds. The *mAnnouncer* instance creates a runnable instance that executes the announcement and schedules another. To send the announcement, the *sendAnnounce* method is used.

```
mLiveViewAdapter.sendAnnounce(mPluginId, mMenuIcon, header,
body, System.currentTimeMillis(), "http://en.wikipedia.org/wiki/
Hello_world_program");
```

The *mPluginId* is used to identify the plug-in to the LiveView™ application. *mMenuIcon* is a string with the path to the menu icon of the HelloWorld project. *Header* and *body* are the texts that will be displayed on the device. The URL that is handed in to the method is the action string for the *openInPhone* action. When the user at a later stage activates the *openInPhone* action, this string will be provided, and the browser can be started, as in the HelloWorld example:

```
protected void openInPhone(String openInPhoneAction) {
  final Uri uri = Uri.parse(openInPhoneAction);
  final Intent browserIntent = new Intent();
  browserIntent.setData(uri);
  browserIntent.setClassName("com.android.browser",
"com.android.browser.BrowserActivity");
  browserIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
  startActivity(browserIntent);
}
```

Another important aspect is when the user changes the settings of the plug-in by using the LiveView™ application. The preference screen is shown and the user changes the update interval.



The update is handled by the *OnSharedPreferenceChangeListener* implemented in the *AbstractPluginService*. It will call an extension in the *HelloWorldService*, the *onSharedPreferenceChangedExtended* method.

From the *AbstractPluginService*:

```
protected OnSharedPreferenceChangeListener mPrefChangeListener =
new OnSharedPreferenceChangeListener() {
  @Override
  public void onSharedPreferenceChanged(SharedPreferences prefs,
String key) {

    ... Do standard stuff ...

    // Call custom plugin implementation
    onSharedPreferenceChangedExtended(prefs, key);
  }
};
```

The *onSharedPreferenceChangedExtended* is called from the *AbstractPluginService*. The implementation of this method in *HelloWorldService* handles the interval change.

```
protected void
onSharedPreferenceChangedExtended(SharedPreferences pref, String
key) {
  if(key.equals(UPDATE_INTERVAL)) {
    long value = Long.parseLong(pref.getString("updateInterval",
"60"));
    mUpdateInterval = value * 1000;
```

```
  }
}
```

Sandbox plug-in example project

The Sandbox plug-in project implements a sandbox plug-in. This means that some of the extended functionality has been implemented. More specifically, all button events are responded upon.

There are examples on adding a text to a bitmap, which is then stored to the phone and then transmitted to the LiveView™ device, as well as examples on sending normal images.

In the *PluginUtils* class you will find various methods for sending images to the LiveView™ device. For example, by using the *android.graphics.Canvas* you can propagate text to a specific bitmap. In the example, when you start the plug-in by pressing the right button, you will be greeted with a text saying "HELLO!". Then all button events will be reacted upon:

- up - Android image pointing up

- down - Android image pointing down

- right - Android image rotating right

- left - Android image rotating left

- right select button - starting/stopping a clock

All events are using the *sendImage* method to send images to the LiveView™ device.

Implementation

First of all, some things need to be initialized. This is done in the *onStart* in the same manner as for the HelloWorld plug-in. For more information, see “Announce plug-in example project” on page 25. A handler is needed to queue image sending, and in the example below, a bitmap is initialized to be used during the lifecycle of the plug-in.

```
mRotateBitmap =
BitmapFactory.decodeStream(this.getResources().openRawResource(R
.drawable.icon_small));
```

The method *isSandboxPlugin* needs to return true for the framework to know how to treat the plug-in.

```
protected boolean isSandboxPlugin() {
  return true;
}
```

The *SandboxPluginService* uses one Handler, *mHandler*, and two runnables to send images to the LiveView™ device, *mTimer* and *mRotator*.

```
private Runnable mTimer = new Runnable() {
```

```
  @Override
  public void run() {
    Date currentDate = new Date(System.currentTimeMillis());
    Format timeFormatter = new SimpleDateFormat("HH:mm:ss");
    PluginUtils.sendTextBitmap(mLiveViewAdapter, mPluginId,
timeFormatter.format(currentDate));

    if(mWorkerRunning) {
      scheduleTimer();
    }
  }
};
```

The method *sendTextBitmap* from *PluginUtils* is used to send a string within a bitmap to the device. In this case, the text is the current time in HH:mm:ss format.

```
private Runnable mRotator = new Runnable() {
  @Override
  public void run() {
    try
    {
      PluginUtils.rotateAndSend(mLiveViewAdapter, mPluginId,
mRotateBitmap, updateDegrees());
    } catch(Exception re) {
      Log.e(PluginConstants.LOG_TAG, "Failed to send image to
LiveView.", re);
    }

    rotate();
  }
};
```

In this case, the *rotateAndSend* utility function is used. The function will take the bitmap and rotate it to a certain degree, provided by *updateDegrees*.

The image sending is controlled via the button presses done by the user, and is implemented by the mandatory callback method button.

```
protected void button(String buttonType, boolean doublepress,
boolean longpress) {
  if(buttonType.equalsIgnoreCase(PluginConstants.BUTTON_UP)) {
    if(longpress) {
      mLiveViewAdapter.ledControl(mPluginId, 50, 50, 50);
    } else {
      rotate(0);
    }
  } else
if(buttonType.equalsIgnoreCase(PluginConstants.BUTTON_DOWN)) {
    if(longpress) {
      mLiveViewAdapter.vibrateControl(mPluginId, 50, 50);
    } else {
      rotate(180);
    }
  } else
if(buttonType.equalsIgnoreCase(PluginConstants.BUTTON_RIGHT)) {
```

```
    toggleRotate(true);
  } else
if(buttonType.equalsIgnoreCase(PluginConstants.BUTTON_LEFT)) {
    toggleRotate(false);
  } else
if(buttonType.equalsIgnoreCase(PluginConstants.BUTTON_SELECT)) {
    toggleTimer();
  }
}
```

The button types are defined in constants in the *PluginConstants* class. For each type of button that has been pressed, different actions are taken, for example rotating the Android image or start a clock. In the example, if the user presses and holds down the up and down buttons, the led will change color, or the device will vibrate.

In the Sandbox plug-in example, the mandatory callback methods *startPlugin* and *stopPlugin* will in their turn call *startWork* and *stopWork*. *StopWork* will clean up the handler queue so that the work does not continue.

```
protected void stopWork() {
  mWorkerRunning = false;
  mHandler.removeCallbacks(mTimer);
  mHandler.removeCallbacks(mRotator);
}
```

The method *screenMode* of the callback interface is used by the LiveView™ application to indicate to the plug-in that the screen has changed status, that is either gone off or on. This is a good way to limit the traffic to the LiveView™ device and thus limit the drain of the battery.

```
protected void screenMode(int mode) {
  if(mode == PluginConstants.LIVE_SCREEN_MODE_ON) {
    startUpdates();
  } else {
    stopUpdates();
  }
}

private void stopUpdates() {
  // stop workers
}

private void startUpdates() {
  // restart workers
}
```

Intents

Broadcast intent

The LiveView™ service will use the following broadcast intent to communicate with the installed plug-ins:

```
com.sonyericsson.extras.liveview.LAUNCH_PLUGIN
```

Manifest example:

```
<receiver android:name="<MYRECEIVER>">
  <intent-filter>
    <action
android:name="com.sonyericsson.extras.liveview.LAUNCH_PLUGIN"/>
  </intent-filter>
</receiver>
```

Bind intent

The following intent is used for binding plug-in to LiveView™ service:

```
com.sonyericsson.extras.liveview.PLUGIN_SERVICE_V1
```

Example:

```
bindService(new
Intent("com.sonyericsson.extras.liveview.PLUGIN_SERVICE_V1"),
<ServiceConnection to use>, 0);
```