SOMA Client J2ME Developer Guide

Learn how to ad-enable your application

CONFIDENTIAL

Date: 6/6/08

Table of Contents

Table of Contents....2

Installing SOMA Client....3

Installing Prerequisites....3

Installing SOMA Client....3

Exploring the Sample Applications....4

Integration Guide....4

Register with Smaato....5

Advertisements Libraries....5

Offline SOMA Client API....5

Online SOMA Client API....5

Selecting the Right SOMA Client API....5

Include Advertising Manually....5

Lifecycle of Ad-Enabled Applications....6

Embedding Advertising Banners in Your Application....6

Sample Application Code....6

Implementing a SomaActionCallback....6

Instantiating the SOMA library....8

Low Level Banner Retrieval....8

Low Level Banner Event Handling....9

High Level Banner Retrieval....11

Include Advertising Automatically....11

Lifecycle of Automatic Ad-Enabled Applications....11

Configuring J2ME Polish....11

Installing SOMA Client

The SOMA Client is distributed as an installer, which installs the SOMA J2ME Client as a JAR file (later also referred to as “SOMA Client” or just “library”). The installer also installs the documentation and sample code.

Installing Prerequisites

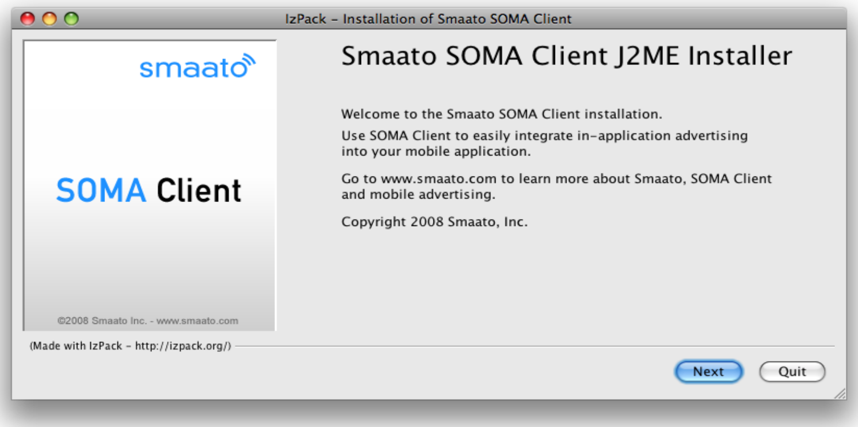
There are no prerequisites necessary other than your normal tools for mobile Java development such as the Sun Java Wireless Toolkit, which can be obtained at http://java.sun.com/javame.

However, it may be helpful for you to use the J2ME Polish framework and Ant for building the sample applications. You can download J2ME Polish at http://www.j2mepolish.org. If you are familiar with the command line, you will probably want to install Ant as a standalone tool as well. You can download Ant at http://ant.apache.org.

Installing SOMA Client

You are now ready to install the SOMA Client. Start the installation by double-clicking the *smaato-soma-${version}.jar* file or by calling `java -jar  smaato-soma-${version}.jar` from the command line, e.g. `java -jar  smaato-soma-3.0.9.jar`. In some cases the Microsoft Internet Explorer renames JAR files to ZIP files when downloading them. In such case please do rename the downloaded file to `smaato-soma-${version}.jar` before you continue.

The installer guides you through the installation process as shown in the following illustrations. In the following sections we refer the installation directory by *${smaato.home}*.


IzPack – Installation of Smaato SOMA Client

Smaato SOMA Client J2ME Installer

Welcome to the Smaato SOMA Client installation.

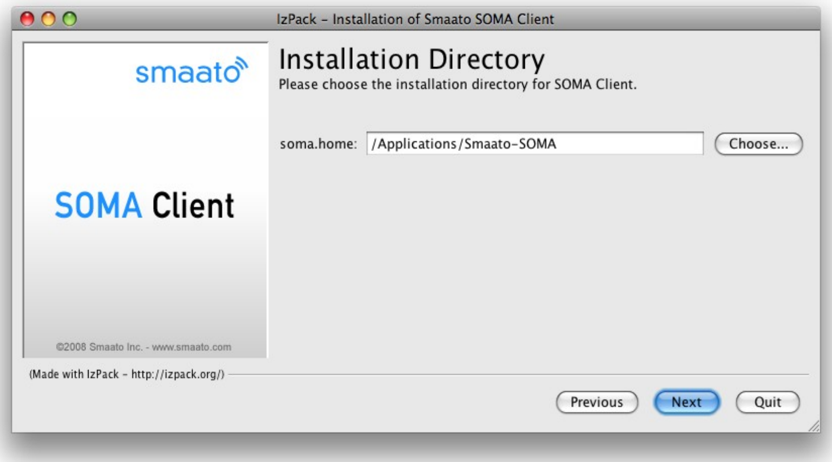
Use SOMA Client to easily integrate in-application advertising into your mobile application.

Go to www.smaato.com to learn more about Smaato, SOMA Client and mobile advertising.

Copyright 2008 Smaato, Inc.

smaato
SOMA Client
©2008 Smaato Inc. - www.smaato.com
(Made with IzPack - http://izpack.org/)
Next
Quit

Figure 1: Start Screen of the Installer


IzPack – Installation of Smaato SOMA Client

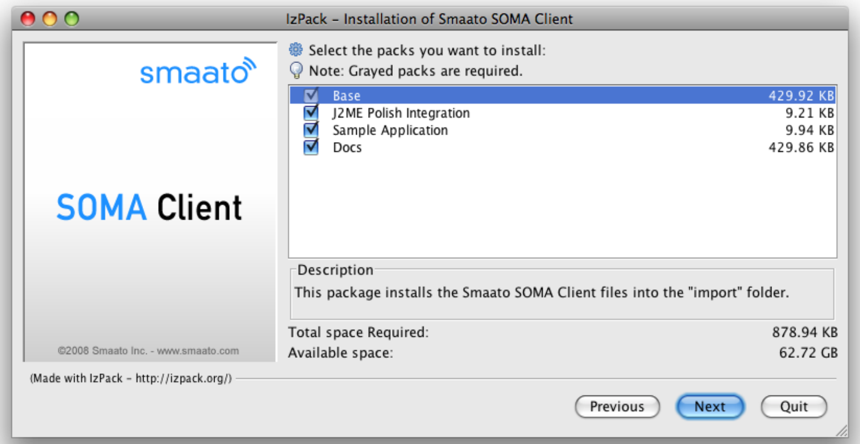
Installation Directory

Please choose the installation directory for SOMA Client.

soma.home: /Applications/Smaato-SOMA

Choose...
smaato
SOMA Client
©2008 Smaato Inc. - www.smaato.com
(Made with IzPack – http://izpack.org/)
Previous
Next
Quit

Figure 2: Selection of the Installation Folder ${smaato.home}


IzPack – Installation of Smaato SOMA Client

Select the packs you want to install:

Note: Grayed packs are required.

✓ Base	429.92 KB
✓ J2ME Polish Integration	9.21 KB
✓ Sample Application	9.94 KB
✓ Docs	429.86 KB

Description

This package installs the Smaato SOMA Client files into the "import" folder.

Total space Required: 878.94 KB

Available space: 62.72 GB

smaato
SOMA Client
©2008 Smaato Inc. - www.smaato.com
(Made with IzPack – http://izpack.org/)
Previous
Next
Quit

Figure 3: Selection Of The Packages Which You Want To Install

Exploring the Sample Applications

You will find the sample applications in *${smaato.home}/samples*. A J2ME Polish configuration is provided along with them – so you can start them directly by calling ant emulator on the command line or by calling Ant from within your IDE within each sample applications directory.

Integration Guide

In the following sections you learn how you can integrate mobile advertising in your application.

Register with Smaato

Before you can start earning money you need to register yourself and your application with Smaato. For testing purposes, however, you do not need to sign up.

For the registration, please go to http://www.smaato.com/dev_zone.php and enter the registration data. In turn you will receive a publisher-ID for your organization, an application-ID for each of your advertising-enabled applications and one or several adspace-IDs for each advertising position within your application.

You can start testing right away without signing up by using '0' (zero) for each ID.

Advertisements Libraries

The Smaato SOMATM Client provides two different libraries that you can include in your application:

1. *${smaato.home}/import/smaato-somaclient-online.jar* loads each shown banner from the network.

2. *${smaato.home}/import/smaato-somaclient-offline.jar* caches downloaded banners for reusing them.

Offline SOMA Client API

The offline API downloads banners and caches them on the device using the Recordstore Management System (RMS). Cached banners will then be cycled until there are too few remaining active banners – in that case new banners will be downloaded automatically. You can specify the minimum number of active banners by calling SomaLibrary.setNumberOfCachedBanners(int number). Banners are stored in the record store management system, so depending on the target device you may need to increase the MIDlet-Data-Size JAD attribute.

In your project you need to integrate the ${smaato.home}/import/smaato-somaclient-offline.jar file when you want to cache banners.

Online SOMA Client API

The online API is a small footprint API that loads each shown banner from the network. In contrast to the offline API you can take over the network communication mode by implementing the SomaConnector interface and calling SomaLibrary.setSomaConnector(SomaConnector connector).

In your project you need to integrate the ${smaato.home}/import/smaato-somaclient-online.jar file.

Selecting the Right SOMA Client API

Which SOMATM Client suits best your needs depends on your project:

- if you want to limit network traffic you should use the offline API.
- if you want to control the network traffic, or if you don't want to store additional data or if you need to squeeze down your application's JAR size, you will want to use the online API.

Both APIs are compatible for the most parts, so you can easily exchange them and test them during your implementation phase.

Include Advertising Manually

The library that you’ve just installed usind the SOMA Client installer provides you an API for easily integrating advertising into your applications.

Lifecycle of Ad-Enabled Applications

The lifecycle of ad-enabled applications consists of following phases:

1. Start up and initialize the SOMA library
   After starting the application the developer acquires an instance of the SOMA library:
   `SomaLibrary.getInstance( long publisherId, long applicationId, long[] adSpaceIds, Display display, SomaActionCallback callback )`

   Once you have initialized the SomaLibrary in this way, you can also access it by calling `SomaLibrary.getInstance().`
2. Retrieve and Display the Advertising Banner
   Whenever appropriate the developer requests banners by calling `getBanner( long adSpaceId )` on the SomaLibrary instance and shows it within the application, for example during application startup or when starting a new game level.
3. Let the User Interact with the Advertising
   The user can interact with banner and trigger the ad specific action like launching the native web browser, initiating a phone call or sending a text message.

Embedding Advertising Banners in Your Application

The SOMA Client J2ME supports the inclusion of advertising banners in two different user interface modes:

1. High Level Integration Mode
   This mode allows integrating banners as CustomItems into Forms. Banners are retrieved by calling `SomaLibrary.getBannerAsCustomItem( long adSpaceId )` and displayed by calling `Form.append( banner )`. This eases the development of interactions with the advertising, since this is handled by the phone's highlevel user interface API (`javax.microedition.lcdui.Form`, etc). A drawback of this mode is that CustomItems can scroll out of the visible area of a Form and thus be rendered ineffective.
2. Low Level Integration Mode
   You can also display advertising banners on lowlevel user interface components such as Canvas or GameCanvas. Since you control the complete user interface, the banners can be placed in any manner and you can and should ensure that they are always visible. A drawback of this mode is that you need to forward events to the banners manually and might need to allow the user to select the banner. The SOMA API provides a `BannerCanvas` that handles user interactions and displays the advertising banners automatically.

Sample Application Code

The SOMA Client ships with a sample application and Java Doc documentation. In the following sections you can see some crucial code snippets.

Implementing a SomaActionCallback

You need to start by implementing `SomaActionCallback`. This callback will be used for allowing the user to interact with mobile advertising, for example launching the native Internet browser for clickthrough advertising. The callback is defined as follows:

```
package com.Smaato.soma;

import javax.microedition.io.ConnectionNotFoundException;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDletStateChangeException;

/**
 * <p>A callback used by the SomaLibrary to invoke actions of banners.</p>
 *
 * <p>Copyright Smaato Inc. 2007</p>
 */
public interface SomaActionCallback {
		/**
		 * Invokes a banner action request like invoking the native browser or initiating a phone call.
		 * Each MIDlet automatically implements this interface, so typically no additional implementation is required for this
method.
		 *
		 * @param url the URL of the banner action request
		 * @return true if the application must be exited to serve this request
		 * @throws ConnectionNotFoundException when the specified request cannot be served
		 */
		public boolean platformRequest( String url )
		throws ConnectionNotFoundException;

		/**
		 * Is called just before exiting this application.
		 * Typically the library calls destroyApp() and then notifyDestroyed() for allowing a platform request.
		 * @param unconditional true when the application is going to be exited in any case
		 * @throws MIDletStateChangeException when the application could not be exited
		 */
		public void destroyApp( boolean unconditional )
		throws MIDletStateChangeException;

		/**
		 * Retrieves the display which is required for handling some interactions like showing a fullscreen advertising.
		 * @return the display
		 */
		public Display getDisplay();

}
```

As you can see some of the methods are already provided by MIDlets anyhow, so it might make sense to let your MIDlet implement the callback. Note that you do not need to implement `platformRequest()` explicitly, since it is provided by the base class `MIDlet` already.

```
package com.Smaato.soma.sample;

import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Graphics;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import com.Smaato.soma.BannerCanvas;
import com.Smaato.soma.BannerItem;
import com.Smaato.soma.SomaActionCallback;
import com.Smaato.soma.SomaLibrary;

/**
 * <p>Sample for using the SOMA library</p>
 */
public class SomaSampleMidlet
extends MIDlet
implements SomaActionCallback
{


		/* (non-Javadoc)
		 * @see javax.microedition.midlet.MIDlet#destroyApp(boolean)
		 */
		public void destroyApp(boolean unconditional) throws MIDletStateChangeException {
				if (this.somaLibrary != null) {
						this.somaLibrary.exit();
				}

		}
```

```
        /* (non-Javadoc)
         * @see com.Smaato.soma.SomaActionCallback#getDisplay()
         */
        public Display getDisplay()
        {
                return Display.getDisplay(this);
        }

        [...]
}
```

Instantiating the SOMA library

You need to provide your registration details to the SOMA Library by calling `SomaLibrary.getInstance( long publisherId, long applicationId, long[] adSpaceIds, Display display, SomaActionCallback callback )`. Typically this is done within the `startApp()` method of a MIDlet:

```
public class SomaSampleMidlet
extends MIDlet
implements SomaActionCallback
{

        private static final long MY_PUBLISHER_ID = 0;
        private static final long MY_APPLICATION_ID = 0;
        private static final long MY_SPACE_ID_STANDARD = 0;

        private SomaLibrary somaLibrary;

        public SomaSampleMidlet() {
                // do nothing
        }

        /* (non-Javadoc)
         * @see javax.microedition.midlet.MIDlet#startApp()
         */
        protected void startApp() throws MIDletStateChangeException {
                Display display = Display.getDisplay( this );
                SomaLibrary lib = SomaLibrary.getInstance(MY_PUBLISHER_ID, MY_APPLICATION_ID, new
long[]{MY_SPACE_ID_STANDARD}, display, this );
                this.somaLibrary = lib;
                BannerItem banner= lib.getBanner( MY_SPACE_ID_STANDARD );
                SampleCanvas canvas = new SampleCanvas( banner );
                display.setCurrent( canvas );
        }
        [...]
}
```

After you have instantiated and configured the SomaLibrary once, you can use `SomaLibrary.getInstance()` without any parameters for easier access.

Low Level Banner Retrieval

You can retrieve low level banners for inclusion in a `javax.microedition.lcdui.Canvas` or a `javax.microedition.lcdui.game.GameCanvas` by calling `getBanner( long adSpaceId )`:

```
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Graphics;
import com.Smaato.soma.BannerItem;
import com.Smaato.soma.SomaLibrary;

class MyCanvas extends Canvas {

        private BannerItem banner;
```

```
	public MyCanvas() {
		long adspaceId = 0;
		this.banner = SomaLibrary.getInstance().getBanner( adspaceId );
	}

	protected void paint(Graphics g)
	{
		this.banner.paint(0, 0, getWidth(), getHeight(), g);
		// now paint content...
		[...]
	}

}
```

Low Level Banner Event Handling

You have several options for letting the user interact with a banner:

1. Use commands and forward command handling to the banner
2. Forward key and pointer events to the banner
3. Combine both methods

The following implementation retrieves the interaction command from the banner and forwards command handling to it:

```
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Graphics;
import com.Smaato.soma.BannerItem;
import com.Smaato.soma.SomaLibrary;

class MyCanvas extends Canvas implements CommandListener {

	private BannerItem banner;

	public MyCanvas() {
		long adspaceId = 0;
		this.banner = SomaLibrary.getInstance().getBanner( adspaceId );
		Command cmd = this.banner.getCommand();
		addCommand( cmd );
		setCommandListener( this );
	}

	protected void paint(Graphics g)
	{
		this.banner.paint(0, 0, getWidth(), getHeight(), g);
		// now paint content...
	}

	public void commandAction(Command cmd, Displayable disp)
	{
		if (this.banner.handleCommand(cmd)) {
			// banner handled the command:
			return;
		}
		// handle the command yourself...
		[...]
	}
}
```

You can also opt to forward key and pointer events to the banner item. Note that you need to decide yourself, whether the banner is currently selected and should thus receive key events. Banner events are independent and can always be forwarded:

```
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Graphics;
import com.Smaato.soma.BannerItem;
import com.Smaato.soma.SomaLibrary;

class MyCanvas extends Canvas {

        private BannerItem banner;
        private boolean bannerIsSelected = true;

        public MyCanvas() {
                long adspaceId = 0;
                this.banner = SomaLibrary.getInstance().getBanner( adspaceId );
        }

        /* (non-Javadoc)
         * @see javax.microedition.lcdui.Canvas#keyPressed(int)
         */
        protected void keyPressed(int keyCode)
        {
                int gameAction = getGameAction( keyCode );
                if (this.banner.handleKeyPressed( keyCode, gameAction )) {
                        return;
                }
        }

        /* (non-Javadoc)
         * @see javax.microedition.lcdui.Canvas#keyReleased(int)
         */
        protected void keyReleased(int keyCode)
        {
                int gameAction = getGameAction( keyCode );
                if (this.banner.handleKeyReleased( keyCode, gameAction )) {
                        return;
                }
        }

        /* (non-Javadoc)
         * @see javax.microedition.lcdui.Canvas#keyRepeated(int)
         */
        protected void keyRepeated(int keyCode)
        {
                int gameAction = getGameAction( keyCode );
                if (this.banner.handleKeyRepeated( keyCode, gameAction )) {
                        return;
                }
        }

        /* (non-Javadoc)
         * @see javax.microedition.lcdui.Canvas#pointerPressed(int, int)
         */
        protected void pointerPressed(int x, int y)
        {
                if (this.banner.handlePointerPressed(x,y)) {
                        return;
                }
        }

        /* (non-Javadoc)
         * @see javax.microedition.lcdui.Canvas#pointerReleased(int, int)
         */
        protected void pointerReleased(int x, int y)
        {
```

```
			if (this.banner.handlePointerReleased(x,y)) {
				return;
			}
		}


		protected void paint(Graphics g)
		{
			this.banner.paint(0, 0, getWidth(), getHeight(), g);
			// now paint content...
		}
}
```

High Level Banner Retrieval

Using high level CustomItem based banners is considerably easier, but the banner will become part of your content, which might not be desired. The following example includes a banner in a Form:

```
import javax.microedition.lcdui.Form;
import com.Smaato.soma.BannerCustomItem;
import com.Smaato.soma.SomaLibrary;

class MyForm extends Form {
		private BannerCustomItem banner;

		public MyForm( String title ) {
			super(title);
			long adspaceId = 0;
			this.banner = SomaLibrary.getInstance().getBannerAsCustomItem( adspaceId );
			append( this.banner );
			// append other form items...
		}
}
```

Include Advertising Automatically

Any application that is based on J2ME Polish (http://www.j2mepolish.org) can be automatically ad-enabled. J2ME Polish consists of a device database, a build framework and client APIs like user interface components, persistence and client server communication components.

Lifecycle of Automatic Ad-Enabled Applications

J2ME Polish can embed advertising in any application screens that extend `de.enough.polish.ui.Screen`, for typical application this means all screens can be ad-enabled. J2ME Polish reduces the visible area automatically for the application and displays the banner above the application's content area. The banner is changed automatically whenever a new screen is shown.

User interactions with the advertising are handled by adding Commands to the application. These are handled transparently by J2ME Polish and allow any advertising action like launching the web browser, initiating a phone call or displaying an ad in fullscreen mode.

Configuring J2ME Polish

You need to configure J2ME Polish by changing your *build.xml* script a little bit.

In the beginning of your *build.xml* script you need to import Smaato specific Ant properties with the following statement:

```
<property name="smaato.home" location="C:\Program Files\thirdparty\smaato"
/>
<property file="${smaato.home}/integration/j2mepolish/soma.properties" />
```

You can then set following additional properties either as Ant properties or in the `<variables>`

section of the `<j2mepolish>` task:

Property/Variable	***Required***	***Values***	***Explanation***
`soma.enableAdvertisements`	yes	`true` `false`	Enables the automatic inclusion of advertisements in your application when set to `true`.
`soma.publisherId`	no	your publisher-ID	The ID of your organization that you have received after registering with smaato. When not set the trial mode will be set automatically.
`soma.applicationId`	no	your application-ID	The ID of the application that you have received after registering with smaato. When not set the trial mode will be set automatically.
`soma.adspaceId`	no	your adspace-ID	The ID of the ad-space that you have received after registering with smaato or '0' (zero) for the default adspace.
`soma.networkMode`	no	`online` `offline`	Selects the library that is used for retrieving advertisements. The `online` mode loads a new banner whenever a new banner is shown. The `offline` mode caches banners on the device. See discussion of the different client apis above. Defaults to `offline`.
`soma.banner.backgroundcolor`	no	Integer in RRGGBB format	The background color for banners – default to white (`0xffffff`).

The following example shows a typical configuration for ad-enabling a J2ME Polish application.

```
<variable name="smaato.enableAdvertisements"   value="true" />
<variable name="soma.publisherId"              value="0" />
<variable name="soma.applicationId"            value="0" />
<variable name="soma.adspaceId"                value="0" />
<variable name="soma.banner.backgroundcolor"   value="0xff0000" />
```

Figure 4 demonstrates the “blackbox” sample application that uses the automatic J2ME Polish integration.

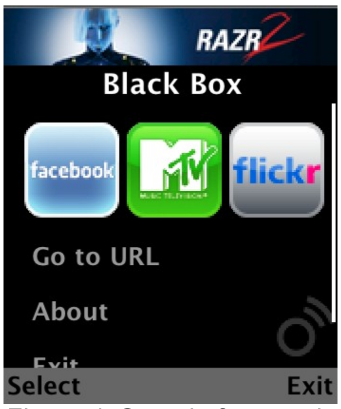


Figure 4: Sample for an ad-enabled application