

October 2009

# Creating the Translator application demo

with Sony Ericsson Flash Lite™ UI components



# Preface

## About this tutorial

---

This tutorial demonstrates how Sony Ericsson UI components can be used to create real interactive mobile applications.

## Sony Ericsson Developer World

---

At [www.sonyericsson.com/developer](http://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 plugins. The Web site also features news articles, go-to-market advice, moderated discussion forums offering free technical support and a Wiki community sharing expertise and code examples.

For more information about these professional services, go to the Sony Ericsson Developer World Web site.

This Tutorial is published by:

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

[www.sonyericsson.com/](http://www.sonyericsson.com/)

© Sony Ericsson Mobile Communications AB,  
2009. All rights reserved. You are hereby granted  
a license to download and/or print a copy of this  
document.  
Any rights not expressly granted herein are  
reserved.

First edition (October 2009)  
Publication number: 1233-4364.1

This document is published by Sony Ericsson  
Mobile Communications AB, without any  
warranty\*. Improvements and changes to this  
text necessitated by typographical errors,  
inaccuracies of current information or  
improvements to programs and/or equipment,  
may be made by Sony Ericsson Mobile  
Communications AB at any time and without  
notice. Such changes will, however, be  
incorporated into new editions of this document.  
Printed versions are to be regarded as temporary  
reference copies only.

\*All implied warranties, including without  
limitation the implied warranties of  
merchantability or fitness for a particular  
purpose, are excluded. In no event shall  
Sony Ericsson or its licensors be liable for  
incidental or consequential damages of any  
nature, including but not limited to lost profits or  
commercial loss, arising out of the use of the  
information in this document.

# Typographical conventions

---

In this document code examples are written in Courier font:

```
softKeys_sks._MSK = "Select" ;
```

Names of labels, buttons, layers and menus are written in italics, for example, *Select File - Install extension*.

Names of edited values, file names and input text are within quotes, for example, "MXP files.zip".

# Trademarks and acknowledgements

---

Adobe, Adobe Flash Lite and Adobe Flash are either trademarks or registered trademarks of Adobe Systems Incorporated in United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc, in the U.S. and other countries.

Google is a trademark or a registered trademark of Google, Inc.

Other product and company names mentioned herein may be the trademarks of their respective owners.

# Document history

---

---

## Change history

2009-10-12	Doc. no. 1233-4364.1	First version published on Developer World
------------	----------------------	--

# Contents

<b>Introduction .....</b>	<b>5</b>
Skill level .....	5
Requirements .....	5
Content of the zip file .....	7
UI components used in this tutorial .....	7
<b>Tutorial .....</b>	<b>8</b>
Creating the application .....	8
Frame 1 – Translator application .....	9
Testing .....	16

# Introduction

This tutorial shows how to use Sony Ericsson Flash Lite™ 2.0 UI Components in developing real mobile applications.

The Translator application is an application based on the famous web application service called Google™ Translate. The demo translates a limited number of words from one language to another by accessing the actual web service from Google.

**Note:** Support for Project Capuchin is required for the application to run on the phone. A list of phones supporting Project Capuchin can be found at: <http://developer.sonyericsson.com/device/searchDevice.do?defaultSearch=true&attributes=e3b7531a-468c-40c5-9e54-8b1ce192ebb3>.

## Skill level

---

**Intermediate** – Good skills in Action Script 2.0 are needed.

## Requirements

---

The applications and tools required to execute this tutorial are listed below.

- **Adobe™ Flash™ CS3/CS4**  
A multimedia authoring application used to create web applications, games, movies, and content for embedded devices. It features support for vector and raster graphics and ActionScript.  
<http://www.adobe.com/products/flash/>
- **Adobe Extension Manager**  
Adobe tool devoted to install new extensions for Adobe Flash CS3/CS4.  
[http://www.adobe.com/exchange/em\\_download/](http://www.adobe.com/exchange/em_download/)
- **Java runtime environment (JRE)**  
The latest version of JRE and installation instructions are available at  
<http://www.java.com/en/download/manual.jsp>
- **Sony Ericsson UI Components**  
The Sony Ericsson UI components are available at  
[http://developer.sonyericsson.com/site/global/docstools/flashlite/p\\_flashlite.jsp](http://developer.sonyericsson.com/site/global/docstools/flashlite/p_flashlite.jsp).  
Install them using the Adobe Extension Manager and read the help documentation for further information on application architecture.

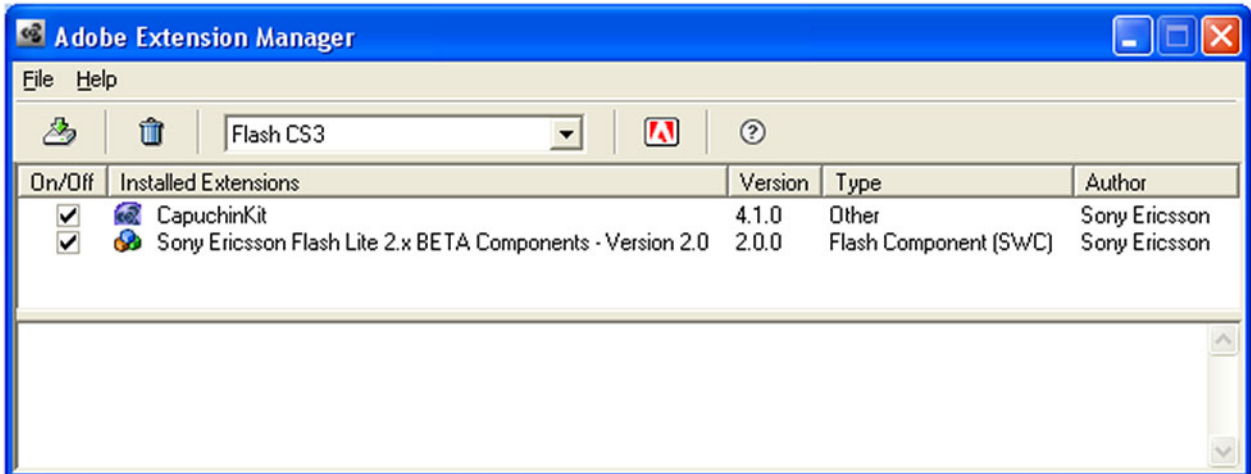
- **Project Capuchin Kit MXP file**

Project Capuchin Kit MXP component for Microsoft® Windows® or OS X.

The Project Capuchin kit extends Adobe Flash CS3 and CS4 to support creating and publishing Flash content as a Project Capuchin application. This file can be downloaded from

[http://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p\\_projectcapuchin.jsp](http://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p_projectcapuchin.jsp).

After successfully installing all MXP packages, they should be listed in the Adobe Extension Manager as in the image below:



# Content of the zip file

The "TranslatorApplication.zip" file contains the folder structure and content described below:



- "Installation Files" folder:
  - "TranslatorApplication.jar" and "TranslatorApplication.jad": The installable application (MIDlet) for your Capuchin enabled Sony Ericsson phone.
- "Source File" folder:
  - "TranslatorApplication fla": The Flash source of the Translator application. This tutorial describes how the Flash file was implemented, using the Sony Ericsson UI components beta version.
- "Tutorial" folder:
  - "tutorial\_flash\_ui\_translator\_appl\_r1a.pdf": This document.

# UI components used in this tutorial

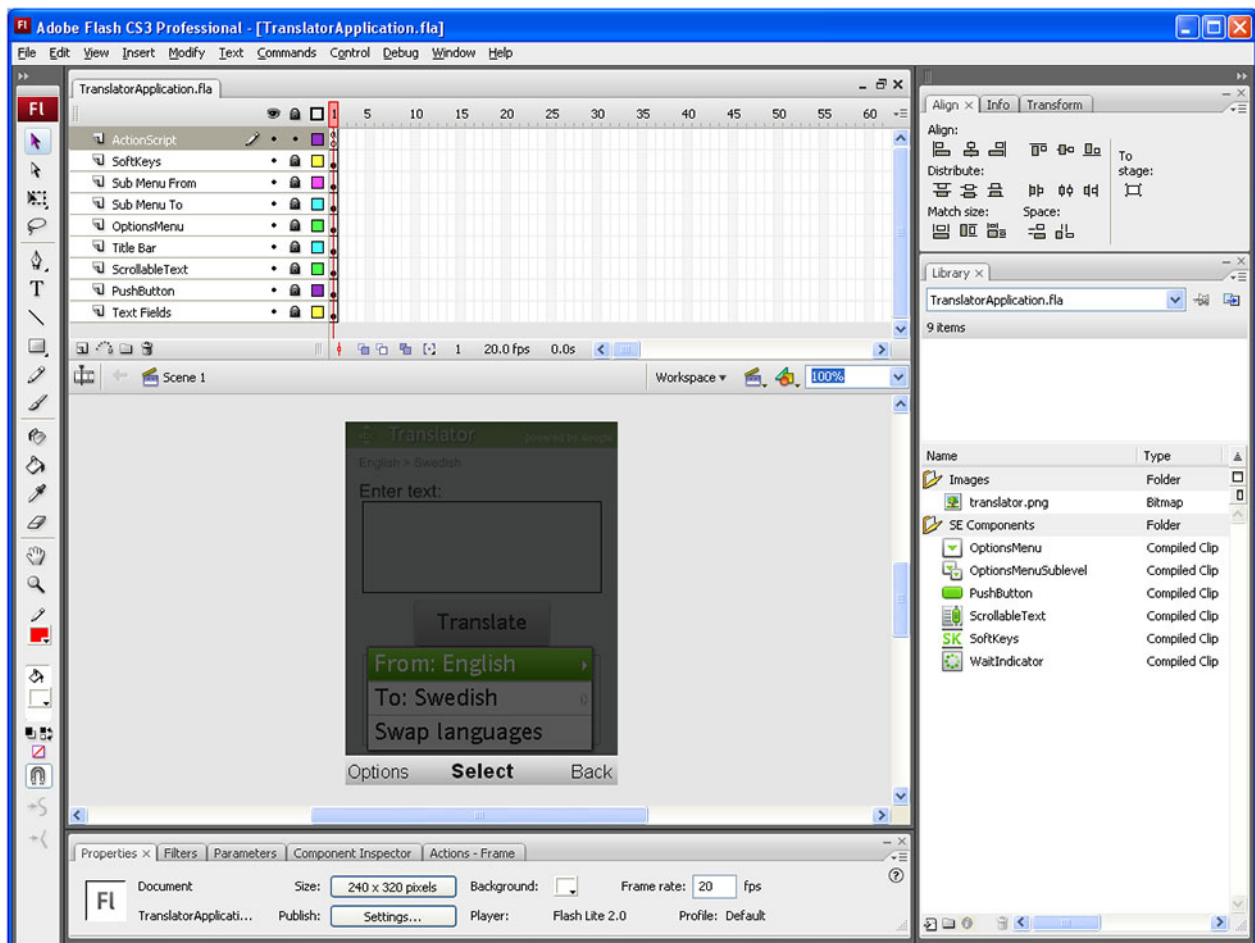
The following Sony Ericsson Flash Lite UI components are used in this tutorial:

- Options Menu: Used to provide the list of possible settings.
- Options Menu Sublevel: Two instances are used: the lists of "From" and "To" possible languages.
- Push Button: Used to send the translation request to the server.
- Scrollable Text: Used to hold the translated text allowing translation of long texts.
- Soft Keys: Used to provide the list of possible actions of each object of the application, such as "Select", "Translate", "Options", and so on.
- Wait Indicator: Used to give visual feedback that the result is being loaded.

# Tutorial

## Creating the application

Open the "TranslatorApplication.fla" file. In the *Library* (*Window > Library*), you find one image and the six Sony Ericsson Flash Lite components used: Options Menu, Options Menu Sublevel, Push Button, Scrollable Text, Soft Keys and Wait Indicator.



The entire application is structured in a single frame, with objects placed on the stage and all interactions defined in the ActionScript code.

The application has the following layers: *ActionScript*, *SoftKeys*, *Sub Menu From*, *Sub Menu To*, *OptionsMenu*, *Title Bar*, *ScrollableText*, *PushButton* and *Text Fields*.

## Frame 1 – Translator application

---

The application was designed to be very simple and fast to use, simulating the same layout as found in Google Translate. Sony Ericsson components were used to simplify the code of the application even more. Each one of these components has a specific purpose:

- **Options Menu:** Used to provide a quick settings menu where the languages of the translation can be easily customised. It opens when the left soft key labeled "Options" is pressed and is dismissed when selecting a language (see the next topic *Options Menu Sublevel* for more details) or the "Swap languages" option. It also opens when right soft key labeled "Back" is pressed.
- **Options Menu Sublevel:** Used to create a list of languages available to the application. Two instances are used – one to select the "from" language of the text to be translated and another to select the "to" language. Each sublevel can be accessed by pressing the *Right* navigation key or the middle soft key (MSK) when the item on the Options Menu is highlighted. To dismiss, select a language on the sublevel, press the right soft key labeled "Back" or press the Left navigation key.
- **Push Button:** Sends the original text to the translation server. A warning message is returned if there is no text to translate.
- **Scrollable Text:** Shows the result of the translation. It can be focused only when the translation has been made.
- **Soft Keys:** Show the available interactions with the application or with the focused object.
- **Wait Indicator:** Displayed when a translation request is in progress. When the request finishes, the component is removed from the screen.

The Soft Keys component (*softKeys\_sks* instance) is always visible and placed above all objects (inside the top-most assets layer called *SoftKeys*).

There are three Soft Keys: left (LSK), middle (MSK) and right soft key (RSK). Their labels can be customised either using the *Parameters* panel (*Window > Properties > Parameters*) or using code.

The instance placed on stage already has its labels customised via the *Parameters* panel with the following values:

Parameter	Value
LSK content (text or path)	"Options"
MSK content (text or path)	"Select"
RSK content (text or path)	"Back"

Depending on which object is highlighted, the Soft Key labels may vary. This modification during runtime is possible using ActionScript code.

The code below shows the customisation of the left and middle soft keys when the Options Menu is opened:

```
softKeys_sks._LSK = "";
softKeys_sks._MSK = "Select";
```

The following table shows all the values that each soft key can assume during the application:

	LSK label	MSK label	RSK label
When the application starts	"Options"	"Select"	"Back"
"Enter text" input text field is focused	"Options"	"Edit"	"Back"
"Translate" push button is focused	"Options"	"Select"	"Back"
The result container (Scrollable Text) is focused	"Options"	"Clear"	"Back"
Options Menu opens	""	"Select"	"Back"
"From" Options Menu Sublevel opens	""	"Select"	"Back"
"To" Options Menu Sublevel opens	""	"Select"	"Back"

The Soft Keys component dispatches two events when any soft key is selected: `onSoftKeyDown` (when a Soft Key is pressed) and `onSoftKeyUp` (when it is released). This application uses the `onSoftKeyUp` event with the following structure to respond to user interaction:

```
softKeys_sks.onSoftKeyUp = function(keyPressed:String):Void {
    switch(keyPressed) {
        case "MSK":
            // Middle soft key actions.
            break;

        case "LSK":
            // Left soft key actions.
            break;

        case "RSK":
            // Right soft key actions.
            break;
    }
}
```

The Push Button component (*translate\_btn* instance) is responsible for triggering the text translation when it is pressed. Its label is customised using the *Parameters* panel (*Window > Properties > Parameters*) to change the value of the *Text* property to "Translate".

The following 3 events are dispatched by the Push Button when they occur: *onSelect* (when the button is pressed), *onFocusIn* (when it gains focus) and *onFocusOut* (when it loses focus). Both *onSelect* and *onFocusIn* are used in this application.

The *onSelect* event is used to send the request to the HTTP server (the code inside the event handler is omitted to simplify its understanding):

```
translate_btn.onSelect = function(sender:MovieClip, itemLabel:String) {
    // Executed actions when the Translate button is pressed.
}
```

The `onFocusIn` event is used to control the middle soft key label:

```
translate_btn.onFocusIn = function(sender:MovieClip) {
    softKeys_sks._MSK = "Select";
}
```

The Scrollable Text component (`resultDisplay_sct` instance) is used to display the translation result once the "Translate" Push Button is pressed. The component allows the user to scroll the text result in case it is a long text that does not fit the defined area. Once highlighted, the Scrollable Text loses focus only if *Left* or *Right* navigation keys are pressed, since *Up* and *Down* keys are used by the component to scroll its content. Another option is to press the middle soft key labeled "Clear", which is responsible for resetting the application.

The `Text` and the `Font size` properties are modified using the `Parameters` panel (`Window > Properties > Parameters`):

Parameters	Values
Text	"Translated text will be shown here."
Font size	"16"

The `Text` property is also changed using code according to the translation requested. In the example below, the property receives the result of the translation when it is successful:

```
resultDisplay_sct._text = unescape(resultText);
```

The following table shows all possible values of the `Text` property:

Parameters	Values
When the application starts	"Translated text will be shown here."
If an empty string is sent to translation	"No text to translate."
When the translation request finishes	<i>The translated text.</i>
When the MSK labeled "Clear" is pressed	"Translated text will be shown here."

The application uses the `onFocusIn` (dispatched when the object gains focus) and `onFocusOut` (dispatched when the object loses focus) events of the Scrollable Text component.

The `onFocusIn` event is used to help with the navigation management and to customise the middle soft key label. The `tabEnabled` property of the `fromText_txt` text field is modified here to avoid interactions with this object while the user scrolls the translated text:

```
resultDisplay_sct.onFocusIn = function(sender:MovieClip) {
    fromText_txt.tabEnabled = false;
    softKeys_sks._MSK = "Clear";
}
```

The *onFocusOut* event turns the *tabEnabled* property of the *fromText\_txt* text field on again, allowing user interaction with this object:

```
resultDisplay_sct.onFocusOut = function(sender:MovieClip) {
    fromText_txt.tabEnabled = true;
}
```

Scrollable Text is only activated when a translation result is ready to be shown. To avoid interactions with the component and to give a disabled look and feel, the *tabEnabled* property is set to *false* and colours are customised as follows:

```
resultDisplay_sct._backgroundColor = 0xE6E6E6;
resultDisplay_sct._textColor = 0x999999;
resultDisplay_sct.tabEnabled = isEnabled; // isEnabled = false
```

When a translation is ready to be shown, the component is activated as described below:

```
resultDisplay_sct._backgroundColor = 0xFFFFFFFF;
resultDisplay_sct._textColor = 0x282828;
resultDisplay_sct.tabEnabled = isEnabled; // isEnabled = true
```

The Options Menu component (*menu\_opt* instance) is used to create a Settings menu, where important information of the application can be easily found and changed.

There are three items added to the *Text* property of the component in the *Parameters* panel (*Window > Properties > Parameters*). These are the default values of the application:

	Value
0	"From: English"
1	"To: Swedish"
2	"Swap languages"

Both "From" and "To" languages are items that contain sublevels, which are provided by another component: the Options Menu Sublevel, explained later on this tutorial. For now, it is important to know that there are two instances of the sub level placed on stage: *submenuFrom\_opt* (opened by the "From" option) and *submenuTo\_opt* (opened by the "To" option).

The "Swap languages" item is responsible for changing the "From" and "To" languages. The following schema exemplifies how it works:

	Original values	Values after selecting "Swap languages"
0	"From: English"	"From: Swedish"
1	"To: Swedish"	"To: English"
2	"Swap languages"	"Swap languages"

The application stores the selected languages in two numerical variables: *fromIndex* and *toIndex*. The "Swap languages" simply invert these values and refreshes the Options Menu (modifying its *Text* property) and the text field that displays the selected languages (placed on a top-left position on stage).

Selection of the "Swap languages" option is detected using the *onSelect* event of the Options Menu, dispatched when the middle soft key is pressed with the component opened. The following piece of code shows the *onSelect* event handler implementing the "Swap languages" functionality:

```

    menu_opt.onSelect = function(sender:MovieClip, itemLabel:String,
itemPosition:Number) {
        if (itemPosition == 2) { // Ensures that the selected option is "Swap
languages".
            var index:Number = fromIndex;
            fromIndex = toIndex;
            toIndex = index;

            // Modifies the selected option on both "From" and "To"
sublevels.
            submenuFrom_opt._highlightPosition = fromIndex;
            submenuTo_opt._highlightPosition = toIndex;

            updateTopMenu(); // Updates the application to use the new
settings.
        }
    }

    function updateTopMenu():Void {
        menu_opt._text = ["From: " + langStr[fromIndex], "To: " +
langStr[toIndex], "Swap languages"];
        languageDisplay_txt.text = langStr[fromIndex] + " > " +
langStr[toIndex];
    }

```

The initial state of the Options Menu is hidden, and it is only opened when the user presses the left soft key labeled "Options". The code below shows the component:

```

menu_opt.show();

```

When the transition of the component finishes, an *onOpen* event is dispatched. The *onClose* event is dispatched when the transition out finishes. These two events are mapped in the application to handle the Soft Keys labels and also to add and remove the "Enter text" input text field of the navigation flow (using the *tabEnabled* property). The code for these two functions is as follows:

```

menu_opt.onOpen = function(sender:MovieClip) {
    softKeys_sks._LSK = "";
    softKeys_sks._MSK = "Select";
    fromText_txt.tabEnabled = false;
}

menu_opt.onClose = function(sender:MovieClip) {
    softKeys_sks._LSK = "Options";
    isOpened = false; // Sets a flag to indicate that the Options Menu is
closed.
    fromText_txt.tabEnabled = true;
    Selection.setFocus(translate_btn);
}

```

Now let us focus on the Options Menu Sublevel. First of all, to make the two instances accessible using the Options Menu, they must be assigned as sublevels in it. An arrow pointing to the right is used to demonstrate that the item on the Options Menu has a sub level. Select the *menu\_opt* instance of the Options Menu and open the *Parameters* panel (*Window > Properties > Parameters*). Fill the *Sub-level instance* property as follows:

	Value
0	"submenuFrom_opt"
1	"submenuTo_opt"

These values (*submenuFrom\_opt* and *submenuTo\_opt*) are the instance names of the Options Menu Sublevel components placed on stage.

Both *submenuFrom\_opt* and *submenuTo\_opt* are filled with a list of several languages available to the translation. To simplify the code, two auxiliary arrays were created to hold the language name (*langStr*) and its identifier (*langCode*). The identifier is used by the Google API to determine the desired *From* and *To* languages of the translation.

The *fromIndex* and *toIndex* variables are references to positions insides these arrays. For instance, if *fromIndex* is set to 9, it references the "English" language in the *langStr* array and also the "en" value in the *langCode* array.

The "From" sublevel is created by the following code and its highlight is set to the initially selected language:

```
submenuFrom_opt._text = langStr;
submenuFrom_opt._highlightPosition = fromIndex;
```

The "To" sublevel is created based on the same idea:

```
submenuTo_opt._text = langStr;
submenuTo_opt._highlightPosition = toIndex;
```

When executing the application, the user opens the sublevels by opening the Options Menu and pressing either middle soft key or Right navigation key with the highlight positioned on the first or the second item.

If the user presses the middle soft key with *submenuFrom\_opt* open, its *onSelect* event is dispatched. This event handler is used to change the value of the *fromIndex* variable and to update the top menu as implemented below:

```
submenuFrom_opt.onSelect = function(sender:MovieClip, itemLabel:String,
itemPosition:Number) {
    fromIndex = itemPosition;
    updateTopMenu();
}
```

If *submenuTo\_opt* is opened and the user presses the middle soft key, the *onSelect* event of this instance is dispatched. This event handler is used to change the value of the *toIndex* variable and to update the top menu as follows:

```
submenuTo_opt.onSelect = function(sender:MovieClip, itemLabel:String,
itemPosition:Number) {
    toIndex = itemPosition;
    updateTopMenu();
}
```

The Wait Indicator component is placed in the application *Library* (*Window > Library*) only to be attached or removed using ActionScript when needed.

When the "Translate" button is pressed, a circle shaped Wait Indicator is created and correctly positioned by the code below:

```
if (!waitIndicator_wti) {

_root.attachMovie("WaitIndicator", "waitIndicator_wti", this.getNextHighestDepth
());
    waitIndicator_wti._shape = "circle";
    waitIndicator_wti._x = 102;
    waitIndicator_wti._y = 228;
}
```

When the server request has finished, the component is removed:

```
waitIndicator_wti.removeMovieClip();
```

# Testing

To publish the Translator application as a jar file, select *Commands > Create Capuchin Application* from the menu. A Swf2Jar window opens. (The User guide for the Swf2Jar tool is available at [https://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p\\_projectcapuchin.jsp](https://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p_projectcapuchin.jsp).)

The following image shows the required input text fields in **bold**, see the Swf2Jar User guide. The Security tab is optional and not necessary for this tutorial to work properly. Once the required fields are filled, click the *Create* button.

