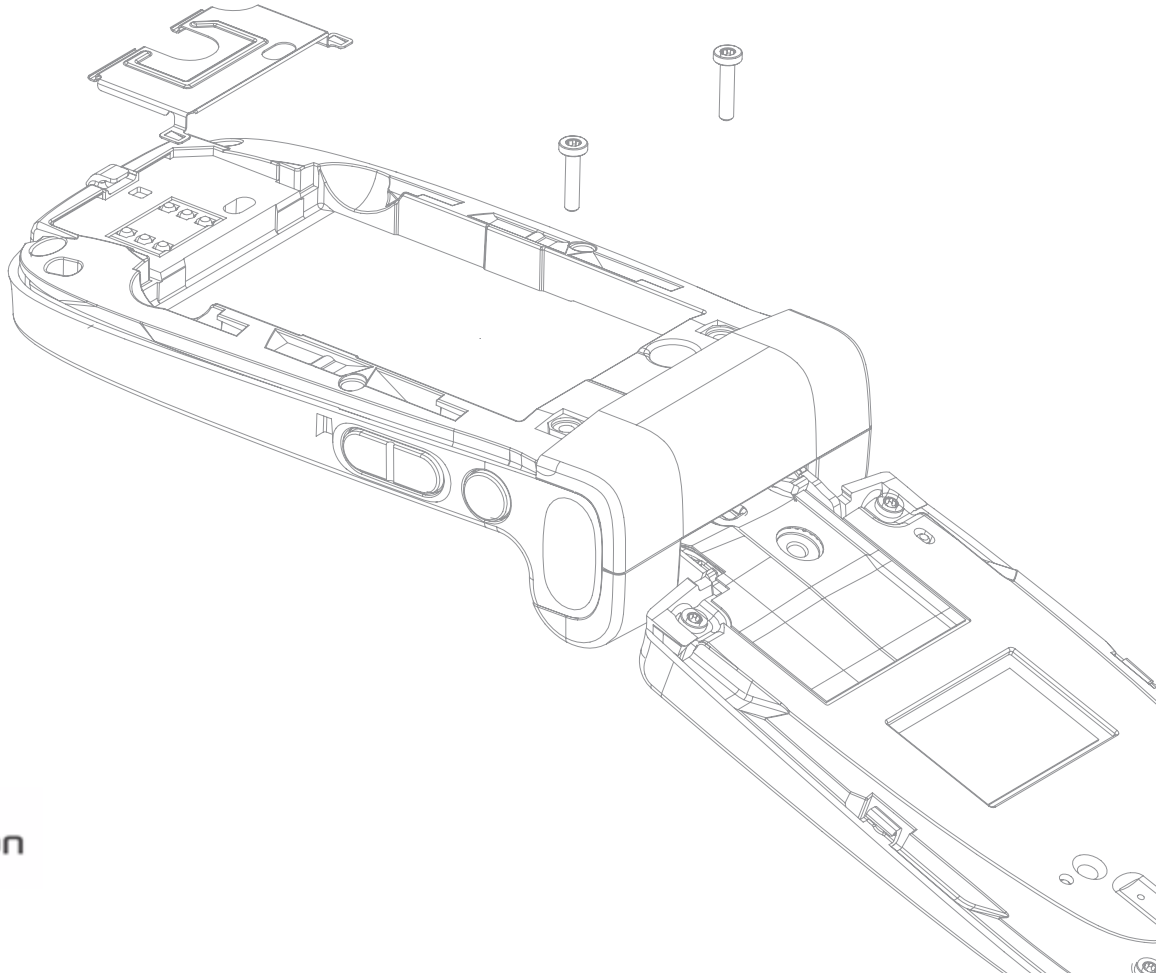
Tutorial

DEVELOPER WORLD THE FAST TRACK FROM MIND TO MARKET

August 2006

Getting started with Macromedia™ Flash Lite™ 1.1

for mobile applications in Sony Ericsson phones



Sony Ericsson

Preface

About this tutorial

This tutorial has been authored by Chris Petty who has been working in multimedia for 6 years before helping to found BlueskyNorth Ltd. a Smashing Ideas company, www.blueskynorth.com, where he is currently Communications Director.

The document describes how to get started with mobile application development in Macromedia Flash Lite™ 1.1.

It is assumed that the reader has intermediate knowledge of using ActionScript. Some knowledge of Macromedia™ Flash™ 4 scripting is also an advantage but not essential.

This tutorial is published by:

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

Phone: +46 46 19 40 00
Fax: +46 46 19 41 00
www.sonyericsson.com/

First edition (August 2006)
Publication number: EN/LZT 108 8939 R1A

Sony Ericsson Developer World

On www.sonyericsson.com/developer, developers will find documentation and tools such as phone White papers, Developers guidelines for different technologies, SDKs (Software Development Kits) and relevant APIs (Application Programming Interfaces). The Web site also contains discussion forums monitored by the Sony Ericsson Developer Support team, an extensive Knowledge base, Tips and tricks, example code and news.

Sony Ericsson also offers technical support services to professional developers. For more information about these professional services, visit the Sony Ericsson Developer World Web site.

Document conventions

Products

Sony Ericsson mobile phones that support Flash Lite 1.1:

Generic names Series	**Sony Ericsson mobile phones**
W600	W600i
W550	W550i, W550c
W900	W900i
W810	W810i, W810c, W810a
W300	W300i, W300c
W850	W850i, W850c
W710	W710i, W710c
K610	K610im

Terminology

CDK	Content Development Kit
NTT DoCoMo	Japanese mobile communications company.
WAP	Wireless Application Protocol

Typographical conventions

Code is written in Courier font:

```
on (keyPress "<Up>") {
     gotoAndPlay("subscribe");
}
```

Trademarks and acknowledgements

Document history

Change history		
2006-08-23	Version R1A	First version.

Contents

Introduction

About Flash Lite

Flash Lite was originally developed by Macromedia (now Adobe) to run Flash based content on the latest generation of mobile handsets and devices. It first appeared as Macromedia™ Flash Lite™ 1.0 in 2003 aimed at the Japanese NTT DoCoMo i-mode system. Flash Lite 1.1 followed in 2004, and contained a number of major improvements, allowing better network access and integration with the handset. Both Flash Lite 1.0 and 1.1 are based on a Flash 4 structure, helping to minimise the footprint and processor demands of the player.

Flash Lite implementations

There are 2 main implementations used by Flash Lite enabled handsets. Firstly there is the “stand alone” model, meaning Flash Lite exists as a separate media application on the handset. This tends to be more suitable for larger applications and games due to greater access to processor power.

The second type of implementation, as used by Sony Ericsson, is the “browser based” model, where Flash Lite essentially runs as a “plug in” within the resident browser of the handset. This means the user can access Flash content via a network connection without needing to open a separate application. This is better suited to Web type content and simple games or animations, as it exhibits slower frame rate and screen quality when compared to the stand-alone version.

General design considerations

The introduction of Flash Lite has encouraged existing Flash Based designers to move into the Mobile and Devices area, but designing for these devices represents a very different challenge to designing for a larger screen.

The first and most obvious difference is the size of the screen. For designers used to working at a minimum of 640 x 480 pixels, the move to 176 x 176 requires adjustment. These smaller dimensions should not limit what your application is capable of, but will be a key consideration in the application design. As the saying goes “content is king”, so the fundamental rule that applies with Flash Lite is using the space you have to provide your content in a legible and meaningful manner. Are images large enough to see? Are the fonts sharp and legible at the small point sizes required? Will the application be readily navigable using the handsets controls as opposed to a mouse / keyboard?

The first step in answering these questions is to understand what options the handset offers you. This allows you to address major considerations at the outset before you even start construction of an application. For instance, take some time to look at how navigation is currently achieved on the handset. Do some keys have a standard usage, equivalent to “Back” or “Enter”? Without a mouse, how is directional navigation achieved? Is there a 4-way joystick or direction button? Also look at the ways that menus are presented. Graphically or in list form, decide which you feel would be most effective for your needs.

By carrying out a brief audit of the handset capabilities you can learn a great deal about how these fundamental issues have already been addressed. You do not need to copy existing models, but be informed by them. As on the Web, mobile devices have their own set of established conventions. Using these can prevent usability problems later in the project.

In terms of presentation of content you will notice that placing a meaningful amount of text on a page while keeping it legible can be a challenge. Handsets generally solve this by having a range of device fonts preinstalled, but one of the advantages of Flash Lite is the ability to present information in different ways. So consider using pixel fonts, generally smaller but just as legible as device fonts when used right. For images you must be careful not to overtax the handset processor by using or animating large JPEG, GIF or PNG. Here, optimisation of the image outside Flash is crucial. A mobile handset will generally have a more limited colour range than a PC, but this can work to your advantage in cutting down image sizes.

Read the documentation provided by Adobe and study examples of existing Flash based content. These will provide information on how you can best integrate your Flash Lite application with the handset and interact with remote data sources. Remember, Flash Lite provides many unique capabilities in terms of graphical quality and animation, giving applications the all important “wow factor” that can set them apart in the congested mobile content market. This has traditionally been the strength of Flash, so bear this in mind and take full advantage of it in your design.

Finally, get hold of a handset for testing purposes before you start your project. This means you can become acquainted with it’s features and produce test movies as you build up to producing you application. Once you commence a project, test constantly and try and get opinions from people removed from the work, as they will invariably point out the glaring error that has been staring you in the face for weeks!

Tutorial

Application outline

For this tutorial we will be working on a small magazine application called MyWorld. This is viewable at: www.blueskynorth.com/se/my_world.swf and aims to show how Web style information can be viewed on a mobile handset without being a WAP style list of text and blue links. MyWorld features:

- News & Info
- Two brief Photo Articles
- A section for users to download wallpapers
- A Subscribe page, allowing users to register for the service via SMS.

Basic design principles covered in the construction of this application are:

- Use of animated elements
- Design of navigation
- Use of fonts and images

ActionScript principles covered are:

- Flash 4 scripting basics
- Interaction between Flash Lite and the handset

- Loading external text data
- Downloading external images to the handset
- Sending user info via SMS

What you need

This tutorial requires you to have a copy of Macromedia™ Flash™ 8 Professional and the Flash Lite 1.1 CDK for Sony Ericsson Handsets, available from the Adobe mobile site at www.adobe.com/devnet/devices/development_kits.html. We will also be using a pixel font, FFF Harmony that needs to be downloaded and installed before you begin (www.fontsforflash.com).

A Sony Ericsson W810i was used for development and testing throughout this project.

All files needed for this tutorial can be found in the same .zip file as this document.

Setting up the application

On opening Flash, be sure you have set the player as Flash Lite 1.1 in your publish settings. From the Files folder, open `my_world_start.fla`. Notice the root timeline, broken into the seven sections of the app (5 sections, plus the main menu and splash screen). You can see that each section is clearly labelled, as this will help us target them later. By moving the play head across the timeline you can see the content within each of the sections.

Step 1

Our Menu will be the hub of our application, so we need a simple way to return to it, which we can use wherever possible throughout. On the latest Sony Ericsson handsets, the left selection key always represents the **Select** action, as does clicking on the navigation key. As the word **Select** appears over the selection key, it is reasonable that we should use this key to enter the application and return to the menu from each section.

On the frame labelled “news” on the root timeline, you will see that there is already a Menu label at the bottom left of the stage. This is positioned so that it will appear above the shoulder key when viewed on the handset. Just off-stage below this label you will see the word “button” in light blue. This is the generic invisible button that we will use throughout this project. Flash Lite is not generally controlled by a pointing device (for example a mouse), but rather by key inputs, so it is useful to use generic invisible buttons and keep them off stage so they are never shown in focus (there are exceptions to this as we will see later). This approach can also help avoid confusion with tab ordering.

Select the button and open the Actions window. You will see no action is currently applied here, so we need to send the movie back to the frame labelled “Menu” when the “Menu” button is clicked. To do this we use an `on (keyPress)` action. As we saw earlier, the left selection key is our default **Select** or Enter key, so we refer to it using the `<Enter>` command. Place the code below on this button:

```
on (keyPress "<Enter>") {
     gotoAndStop("menu");
}
```

This code has already been applied wherever else the Menu button appears in the application.

Step 2

The next step is to look at the construction of the main menu. On the frame labelled “menu” on the root timeline you will find a movie clip labelled “menu”. This is our main menu movie Clip and it allows the selection of items and sends the movie to the relevant frame. Open the movie clip and you can see it is fairly simple in construction, with a highlight tweening in for each section. You should also notice another instance of our invisible button just off-stage on the right-hand side. This will control the movement of the menu and target the selected section on the root timeline. On frame 3, click the button instance (on button layer) and open the Actions window. We need this button to perform three actions for us:

1. Move to the previous item when the user presses **Up**

2. Move to the next item when the user presses **Down**

3. Move to the selected item when the user presses **Select**

To achieve this we use the actions shown below:

```
on (keyPress "<Up>") {
     gotoAndPlay("subscribe");
}
on (keyPress "<Down>") {
     gotoAndPlay("article1");
}
```

Note: You will note that we are specifically targeting the frame labels, as these are where our animation begins

We still need to add the action for the **Select** or **Enter** key, which will tell the root timeline which frame to move to. This requires us to use a `tellTarget` action (Remember, Flash 4 ActionScript does not allow for the use of dot syntax). This should be added as follows:

```
on (keyPress "<Enter>") {
     tellTarget ("/") {
          gotoAndPlay("news");
     }
}
```

In this script we are targeting the root level, represented by “`/`”, and telling it to go to the frame “news”. We could also have used “`../`” since this would target the parent, in this case the root level.

Step 3

Now we have a menu that can be controlled by direction movements and will send us to the right section when clicked. We also have a way of getting back to the menu, using the menu button. What would be useful to finish this main navigation is a way to make sure that on returning to the main menu, your previously selected item is still highlighted, as opposed to the menu defaulting to item 1.

As we have seen, all of the sections are already labelled within the menu movie clip. This means that if we target these labels when the menu button is pressed, we will be able to trigger the correct animation.

What we need to do now is set a variable to keep track of the last section visited as we return to the menu. We will call this variable `lastFrame`. We need to place a value for this variable on every section, so on the root timeline select the actions layer at the frame marked “news”. Open the Actions window and you will see that this frame currently has a `stop()` action assigned to it. We need to set our variable as follows:

```
lastFrame = "news"
```

Now we need to send the menu to the correct frame. We can do this by placing an additional action on the Menu button we set up in step 1. While still on the "news" frame of the root timeline, click again on the invisible button at below bottom left hand of the stage. On opening the Actions window, we see the code created in step1. Now let us add a further action within this. Below the `gotoAndStop("menu")` action, but still within the `keyPress("<Enter>")` action, we need to target the menu clip and send it to the value of `lastFrame`, again using `tellTarget`:

```
tellTarget ("menu") {
     gotoAndPlay(../:lastFrame);
}
```

Notice that the `lastFrame` variable is written as `../:lastFrame`. Flash 4 "slash syntax" dictates that we prefix our variables using "`/:`". In this case, as we know that the variable we created in step 2 is located on the root level, we target this using "`..`", moving us to the parent. The final code on the button should appear as follows:

```
on (keyPress "<Enter>") {
     gotoAndStop("menu");
     tellTarget ("menu") {
          gotoAndPlay(../:lastFrame);
     }
}
```

You should now be able to test your work so far. This can be done using the emulator included in Flash (as detailed in the CDK), or by transferring the file to your test handset via Bluetooth or transfer cable.

Step 4

Now we have our main navigation set up, it is time to look at sub navigation within the magazine sections. As we discussed earlier our means of control on the handset are limited to key presses. We have already assigned the Left selection key as our select key, and we are mainly using that to return to the menu. The right selection key cannot be accessed by Flash, but we can still use any of the numeric keys or the navigation key. As many of the sections within MyWorld have multiple pages that we need to access, we need to decide how to navigate through them in a consistent way. The two most intuitive ways to do this would be using **Up** & **Down** or **Left** & **Right**. Within Flash Lite applications a common problem is displaying body text within the limited space available. This means regularly using scroll bars, utilising **Up** & **Down**, so it is worth reserving these keys for this purpose. As a result we will be using **Left** & **Right** to navigate pages.

On the root timeline go to the frame marked `article1`. You will see that the content layer contains a movie clip named `article1`, which in this case is an article on the Yosemite Valley. This movie clip consists of four pages, so we need a system to move through them as required. Open the movie clip and you can see that, for simplicity, this has been set up using tweens. The user needs to be able to cycle though the pages, moving backwards and forwards within the article.

As with the main menu, we can achieve this easily by using a button to catch the **Left** & **Right** inputs and move the clip to the correct frame. Within the movie clip, click on the button layer at frame 5 and you will see that another instance of our invisible button is located at the bottom of the stage. Click on this button and open the Actions window. Currently, this button has no actions on it, so we need to tell it what to do if **Left** or **Right** keys are used. As before we do this with a simple `keyPress` action adding the code below:

```
on(keyPress"<Left>"){
     gotoAndPlay("back one")
```

```
}
on(keyPress"<Right>"){
     gotoAndPlay("two")
}
```

You will notice that we are targeting the relevant animations within the movie clip to take us back one page, or forward one page. Obviously, on the first page you can only move forward, and on the final page you can only move back. Now we have our article moving smoothly, it would be useful to provide the user with visual cues as to which way they can move. Return to the root timeline, still on the frame labelled `article1`. On the left of the content layer you will see a movie clip called `arrows`. Open this movie clip and you will notice it contains three frames: left, both and right. We need to trigger this clip to display the correct arrows as we move through article1. Once again, open the `article1` movie clip, click on frame 5 of the actions layer and open the Actions window. Currently this contains a `stop()` action. We need to add a `tellTarget` action below this to tell the `arrows` clip to move, in this case to display `both`:

```
tellTarget("/arrows"){
     gotoAndStop("both")
}
```

The movie clip is targeted using `/arrows`, and as before this slash targets the root timeline where `arrows` is located. So, we now have arrows visible to show the user where they can move, but on testing this we see we have a problem, as our arrows obscure the content. We need a way to make them appear and disappear when required. To do this we can set up a simple looping timer, which will turn the arrows "off" after a delay. Returning to the root timeline notice the words `arrowTimer` written in red above the application. This is our `arrowTimer` movie clip. Open this and you can see the actions layer consists of an initial `stop()` frame, followed by 12 blank frames. When the clip is on the stop frame we want the arrows to be invisible, but on frame 2 onwards they should be visible. We can control this by using `set-Property` actions. Click on frame 1 of the actions layer and open the Actions window. Below the `stop()` action we need to add our command to set the visibility of `arrows` as follows

```
setProperty("/arrows", _visible, false);
```

Again, this is a Flash 4 action that newer Flash users may not be conversant with. As the name suggests, `setProperty` sets a property of a particular movie clip to a defined value. The statement itself is made up as follows:

```
setProperty(target, property, value);
```

In our case, the target is `/arrows` and the property we need to alter is `_visible`. This property requires a Boolean value (true or false), in this case we want the clip to be invisible, so `_visible = false`. We now need a corresponding action to make the arrows visible, so on frame 2 of the actions layer place the following:

```
setProperty("/arrows", _visible, true);
```

This movie clip has 12 blank frames from frame 2 onwards, meaning at a frame rate of 12fps the arrows should appear for around 1 second. All we need to do now is trigger this timer and we do this by going back into our `article1` movie clip. Returning to frame 5 of the actions layer you will see the action we previously entered to control the `arrows` movie clip. Below this add the following action:

```
tellTarget ("/arrowTimer") {
     play();
```

```
}
```

This will mean when the movie enters this frame, the arrowTimer will kick off, displaying the relevant arrows for a second.

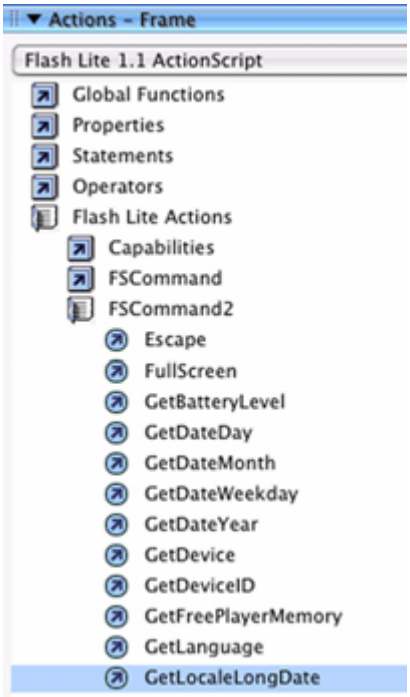
Step 5

Now it is time to look at how Flash Lite allows us to interact with the handset. Within Flash Lite a whole new group of commands appeared, known as FSCommand2 actions (they are fully detailed within the Flash Lite 1.1 CDK), which allow us to do just this. We are going to take a look at a couple of examples of how we can use them.

Go to Frame 1 on the root timeline (labelled "`splash`"). On the bg layer you can see a dynamic text box on the right side of the stage. We are going to use this to display the current date as set on the handset. By clicking on the text box you will notice its variable name is `dateDisplay` (**Note:** We cannot use instance names for text boxes in Flash Lite 1.1), now we need to set up the command to retrieve this data from the handset and display it in this location. Click on the actions layer at Frame 1 and open the Actions window. Within your actions library, you should see "Flash Lite Actions" as a separate category. Open this and from the sub categories, select "Fscommand2". You will see a list of all FSCommand2 actions, move down this and find `GetLocaleLongDate`. This action will import the date in long form, so click on it and it will appear below the existing `stop()` action as:

```
fscommand2("GetLocaleLongDate");
```

We now need to target the location where we wish this to be displayed, in this case our dynamic text box `dateDisplay`, so add this to the action:




```
fscommand2("GetLocaleLongDate", "dateDisplay");
```

Testing your movie you should now see the date appear. You will need to ensure that you have embedded all the characters you are likely to need in any given text box prior to publishing, or your data will be displayed in a device font. When using pixel fonts, it is also essential use the correct point size (generally 8 or multiples of 8) and be sure that the text box is left aligned and located on whole X and Y values (e.g. 1.0, not 1.5) as failure to do any of these steps may result in "blurring".

Step 6

Another FSCommand2 we can use on Sony Ericsson handsets is `StartVibrate`. By utilising this we can make the handset vibrate in response to particular user inputs. If you move to the `subscribe` frame of the root timeline, you will see we have a simple form comprising of three text fields and two buttons. We are going to use a `StartVibrate` action on our send button to indicate if a text field has not been completed.

Click on the `send` button at the lower right of the stage and open the Actions window. You will see the following code on the button:

```
on (keyPress "<Right>") {
     if (myname eq "") {
```

```
          tellTarget ("field1") {
               gotoAndPlay("flash");
          }
     }
     if (surname eq "") {
          tellTarget ("field2") {
               gotoAndPlay("flash");
          }
     }
     if (email eq "") {
          tellTarget ("field3") {
               gotoAndPlay("flash");
          }
     } else {
          myMessage = "Name = " add /:myname add ", " add "Surname = " add /
:surname add ", " add "Email = " add /:email;
     }
}
```

This code is checking if any of the three fields (named `myname`, `surname` and `email`) has a value of `""`, meaning they contain no data. If this is the case, a movie clip named `Field1,2 or 3` is made to flash as a visual cue. If all text fields contain data, a variable called `myMessage` is created to contain a string constructed of the three pieces of data.

Note: You may have noticed that in Flash 4 syntax, we use the `eq` operator to check the text fields as opposed to "==" (this being used for numerical values). Also, in the construction of the `myMessage` string we use the `add` operator as opposed to "+". See the CDK for more details on operator usage in Flash 4 syntax

We now need to add our `StartVibrate` command. This comprises of three values:

```
FSCommand2( "StartVibrate", time_on, time_off, repeat )
```

Here, `time_on` represents the duration of the vibrations, `time_off` represents the time between vibrations (both expressed in milliseconds up to a maximum of 5 seconds), and `repeat` represents the amount of vibrations that will occur. For this example we will have a vibration of 2000ms followed by a gap of 2000ms with 2 repeats:

```
FSCommand2( "StartVibrate", 2000, 2000, 2)
```

We now need to place this command within each of the `if` statements in our code block (make sure you place them outside the `tellTarget` statements). The code should now be as follows:

```
on (keyPress "<Right>") {
     if (myname eq "") {
          tellTarget ("field1") {
               gotoAndPlay("flash");
          }
          fscommand2("StartVibrate", 2000, 2000, 2);
     }
     if (surname eq "") {
          tellTarget ("field2") {
               gotoAndPlay("flash");
          }
          fscommand2("StartVibrate", 2000, 2000, 2);
```

```
     }
     if (email eq "") {
          tellTarget ("field3") {
               gotoAndPlay("flash");
          }
          fscommand2("StartVibrate", 2000, 2000, 2);
     } else {
          myMessage = "Name = " add /:myname add ", " add "Surname = " add /
:surname add ", " add "Email = " add /:email;
     }
}
```

Step 7

Before leaving the Subscribe section, it is worth looking at a couple of other issues that we encounter when using forms. On a mobile handset, we ca not just click and type in text fields as we are used to doing on the Web. Instead, we have to activate the text field by clicking `select` when it is highlighted. This opens the native text input window of the handset. This raises two major issues for us on this application.

Firstly, we need to use the **Select** key to open the text input feature, but we have already assigned this key as our default “back to menu” button. The only real solution is to assign another key with the “menu” function and you can see that this is already in place, prompting the user to press **Left** to return to the main menu.

Secondly, the user needs to be able to see which text field is highlighted. We achieve this using the `_focusrect` command. This places a yellow rectangle around any button or input field currently in focus, so as we move down the page the focus changes accordingly. The `_focusrect` command has a default value of `true` when your movie is published. The one problem this creates is that any graphical buttons you have on the page will also have a yellow rectangle around them when in focus, and this may not be desirable as part of your design. We can get round this problem by amending our `back` and `send` buttons. Click again on the `send` button and open the Actions window. We need to tell `_focusrect` to equal `false` when the focus is on this button, but then return the value to `true` when the focus moves off the button. We do this by adding `on(rollover)` and `on(rollout)` commands:

```
on (rollOver) {
     _focusrect = false;
}
on (rollOut) {
     _focusrect = true;
}
```

Step 8

Our application now works as a contained entity, but what we need is to be able to communicate with a server to send or receive data. In this case, we need to send the information gathered in the subscribe form as an SMS message. We can do this using a `getURL` statement containing two main pieces of information, our data (the `myMessage` string) and the address (the SMS number of the recipient). This should be placed within the else statement on the send button and written as follows:

```
getURL("sms:000000000?body=" add myMessage);
```

Notice that we prefix the phone number (000000000) with `sms:` as this will trigger the handset SMS editor to open so you can review the message before sending.

We can also use `getURL` to bring data back to our handset. Go to the `wallpaper` frame on the root timeline. You will recognise that we have a movie clip arrangement similar to that covered in step 4. In this case our content is contained within the movie clip `wallpaperHolder` on the `content` layer. Click and open this movie clip and go to frame 2, labelled `start`. On the `button` layer you will see we have our invisible button at the base of the page, and as before this contains our actions to move left or right (in this case, only right). We now need to add an action to initiate the download of the image from a remote location. On the stage, you will see that we are using number keys to select images, so on this frame we press `1` to download. Go back to the invisible button, and in the Actions window add the following code below the existing actions:

```
on (keyPress "1") {
     getURL("http://www.blueskynorth.com/se/stairs.jpg");
}
```

This code will trigger a download of `stairs.jpg` from the URL listed when key 1 is pressed. The handset will prompt you to make a network connection, the image opening in a new browser window once the download is complete. This image is now saved in the "Pictures" directory of the handset, and can be set as wallpaper. This approach can also be used to link to other pages / movies if required.

Step 9

Our final step is to import text data from the server to populate our `News & Info` section. We can use `loadVariables` to do this. Go to the news section on the root timeline and, again, you will see our content is set up in the sideways scrolling format. In this example we have three sets of dynamic text boxes set up to represent our three news stories within the movie clip `news`. Each story has a one line header, and a larger multiline box for body text (again we have use FFF Harmony and embedded all necessary characters). Now all we need to do is obtain our data from the server and make sure it goes in the right locations. So let us start by getting the data from the server. The data in this instance is all contained under separate headings within a document named `myworld_news.txt`. We can call this document as we open MyWorld so all the variables it contains will be preloaded, ready for when we need them. To do this go to the `splash` frame of the root timeline and click on the actions layer. In the Actions window you will see the FScommand2 we created earlier. Below this we need to add our `loadVariables` command as follows:

```
loadVariables("myworld_news.txt", "");
```

Note: loadVariables requires two parameters to work, in this case we only need the name of the target file, so the second parameter is represented by empty quotes.

Our text is now loaded, but we need to extract the correct elements for each of our text boxes. If you view `myworld_news.txt`, you will see that it is simply set out with head(x) and story(x) and these variables now exist on the root level of our movie. Let us return to the `news` frame and within the `newsholder` movie clip, open the `news` movie clip. The text box currently containing the words `head 1` represents our first header, which is now located at the root level as `head1` so we need to target this variable. We are currently two levels deeper than the root level, so our variable name needs to reflect this, so input the variable name `../../:head1` for this textbox. On the multiline box below, we follow the same steps, using `../../:story1`.

Now test the final movie on your handset. All features should now work, giving you a fully functional example of Flash enabled content.

Conclusion

This tutorial has shown you a range of skills you will need if developing Flash Lite content for mobile handsets. The next recommended step is to develop your own application from scratch to build up your experience. For further reading on this subject I recommend the Adobe mobile site (www.adobe.com/mobile) where you can find news and articles as well as a user forum and plenty of examples of existing content.