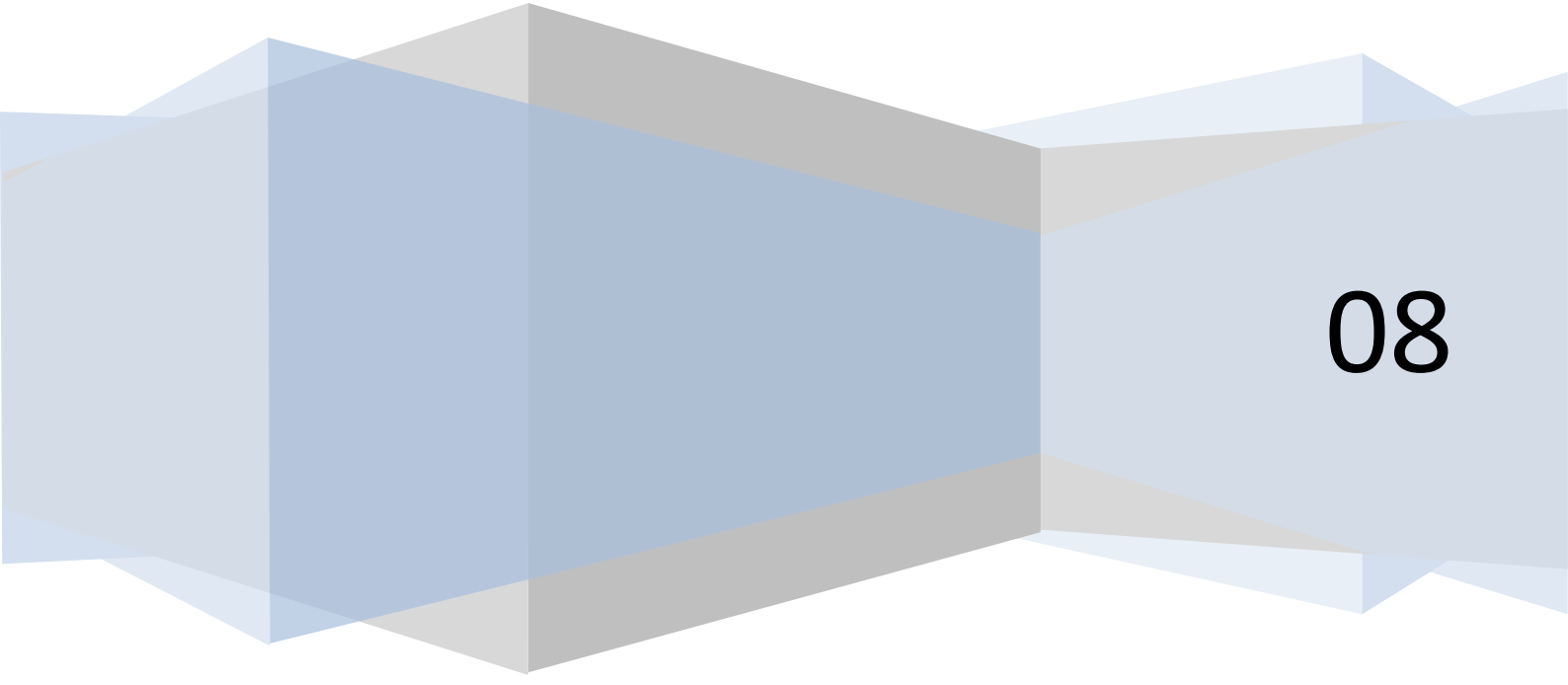
Version 1.0

Creating Themes for the KU990®

A beginner's Flash guide

Joseph Earl



08

TABLE OF CONTENTS

Requirements 2

Introduction 2

Guide 5

1 Setting up the Document 5

2 Loading Files 7

3 Creating the Icon Reel 11

4 Creating the Launcher 14

5 Creating the Scrollbar Control for the Icon Reel 16

6 Creating the clock 18

7 Making the 'Main' Scene 19

8 Making the 'Menu' Scene 20

9 Publishing Your Theme 20

10 Comments 22

Reference 23

DEP and ID Codes 23

Code Snippets 25

Copyright Information 27

This Guide 27

The 'MyTheme' Example theme 27

REQUIREMENTS

SOFTWARE: **Adobe Flash® CS3 Professional** (or Flash® 8 Professional or later)
A Text editor (such as Notepad®)

FILES: Download (Includes finished theme, source code and resources)

INTRODUCTION

Some previous experience with the Flash® CS3 interface is useful, though the guide does contain numerous screenshots to guide you through (if you are using a previous version of Flash® professional, there may be some differences between the program interface and the screenshots); and the guide comes with several resources:

- The completed FLA is included in Flash® CS3 format, and Flash® 8 format in the Resources/Source/ folder
- The completed published SWF and relevant external files are included in Resources/Publish/
- All images required for the task are included in Resources/Images/
- A sample external user image is included in Resources/User Image/

Developing a theme is no different to creating a standard Flash application, though there are some special considerations that must be given as you are publishing to a mobile device (these are taken from the Adobe Flash® CS3 reference:

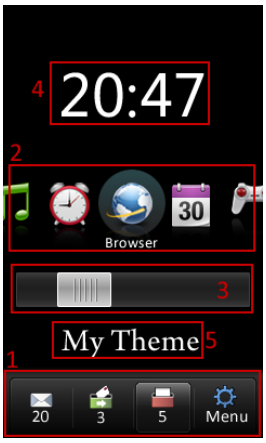
- Keep the file and its code as simple as possible. Remove unused movie clips, delete unnecessary frame and code loops, and avoid too many frames or extraneous frames.
- Using `FOR` loops can be expensive because of the overhead incurred while the condition is checked with each iteration. When the costs of the iteration and the loop overhead are comparable, execute multiple operations individually instead of using a loop. The code may be longer, but performance will improve.
- Stop frame-based looping as soon as it is no longer needed.
- When possible, avoid string and array processing because it can be CPU-intensive.
- Always try to access properties directly rather than using ActionScript getter and setter methods, which have more overhead than other method calls.
- Manage events wisely. Keep event listener arrays compact by using conditions to check whether a listener exists (is not `null`) before calling it. Clear any active intervals by calling `clearInterval`, and remove any active listeners by calling `removeListene`r before removing content using `unloadapplication` or `removeapplicationClip`. Flash does not re-collect SWF data memory (for example, from intervals and listeners) if any ActionScript functions are still referring to the SWF data when a movie clip is unloaded.
- When variables are no longer needed, delete them or set them to `null`, which marks them for garbage collection. Deleting variables helps optimize memory use during run time, because unneeded assets are removed from the SWF file. It is better to delete variables than to set them to `null`.
- Explicitly remove listeners from objects by calling `removeListener` before garbage collection.
- If a file consists of multiple SWF files that use the same ActionScript classes, exclude those classes from select SWF files during compilation. This can help reduce file download time and run-time memory requirements.
- Avoid using `Object.watch` and `Object.unwatch`, because every change to an object property requires the player to determine whether a change notification must be sent.
- If ActionScript code that executes on a keyframe in the timeline requires more than 1 second to complete, consider splitting up that code to execute over multiple keyframes.
- Remove `trace` statements from the code when publishing the SWF file. To do this, select the Omit Trace Actions check box on the Flash tab in the Publish Settings dialog box.
- When one SWF file loads another SWF file that contains a custom ActionScript class (for example, foo.bar.CustomClass) and then unloads the SWF file, the class definition remains in memory. To save memory, explicitly delete any custom classes in unloaded SWF files. Use the `delete` statement and specify the fully qualified class name, such as: `delete foo.bar.CustomClass`.
- Limit the use of global variables, because they are not marked for garbage collection if the movie clip that defined them is removed.

- Avoid using the standard user interface components (available in the Components panel in Flash). These components are designed to run on desktop computers and are not optimized to run on mobile devices.
- Whenever possible, avoid deeply nested functions.
- Avoid referencing nonexistent variables, objects, or functions. Compared to the desktop version of Flash Player, Flash Lite 2 looks up references to nonexistent variables slowly, which can significantly affect performance.
- Avoid defining functions using anonymous syntax. For example, `myObj.eventName = function{ ...}`. Explicitly defined functions are more efficient, such as `function myFunc { ...}; my Obj.eventName = myFunc;`.
- Minimize the use of Math functions and floating-point numbers. Calculating these values slows performance. If you must use the Math routines, consider precalculating the values and storing them in an array of variables. Retrieving the values from a data table is much faster than having Flash calculate them at run time.
- Most devices that support Flash Lite play back content at about 15 to 20 frames per second (fps). The frame rate can be as low as 6 fps. During development, set the document frame rate to approximate the playback speed of the target device. This shows how the content will run on a device with limited performance. Before publishing a final SWF file, set the document frame rate to at least 20 fps or higher to avoid limiting performance in case the device supports a higher frame rate.
- When using `gotoAndPlay`, remember that every frame between the current frame and the requested frame needs to be initialized before Flash plays the requested frame. If many of these frames contain different content, it could be more efficient to use different movie clips rather than using the Timeline.
- Although preloading all content by putting it at the beginning of the file makes sense on the desktop, preloading on a mobile device can delay file startup. Space content throughout the file so that movie clips are initialized as they are used.

Themes may vary considerably in their complexity, and this guide is only an introduction, the rest you will discover playing around for yourselves or talking to other people. However this guide does aim to touch on all of the important aspects of theme creating. The theme you create using this guide will hide the main phone launcher, feature different backgrounds which can be set by the user through an XML configuration file (where we will also store the id and dep codes for the phones apps) and a user icon which is stored externally so it can be changed by the end user.

Our theme (and most themes in fact) will consist of only a few main elements:

1. A launcher, with 3 buttons showing SMS count sand linking to the inbox, drafts and sent folders, and another button which opens the normal phone menu.
2. A reel of 12 icons, which the user will be able to scroll left or right. The user will be able to press the icon in the center to launch it.
3. A horizontal scrollbar to control the reel of icons
4. A clock
5. A user image which will be loaded externally.



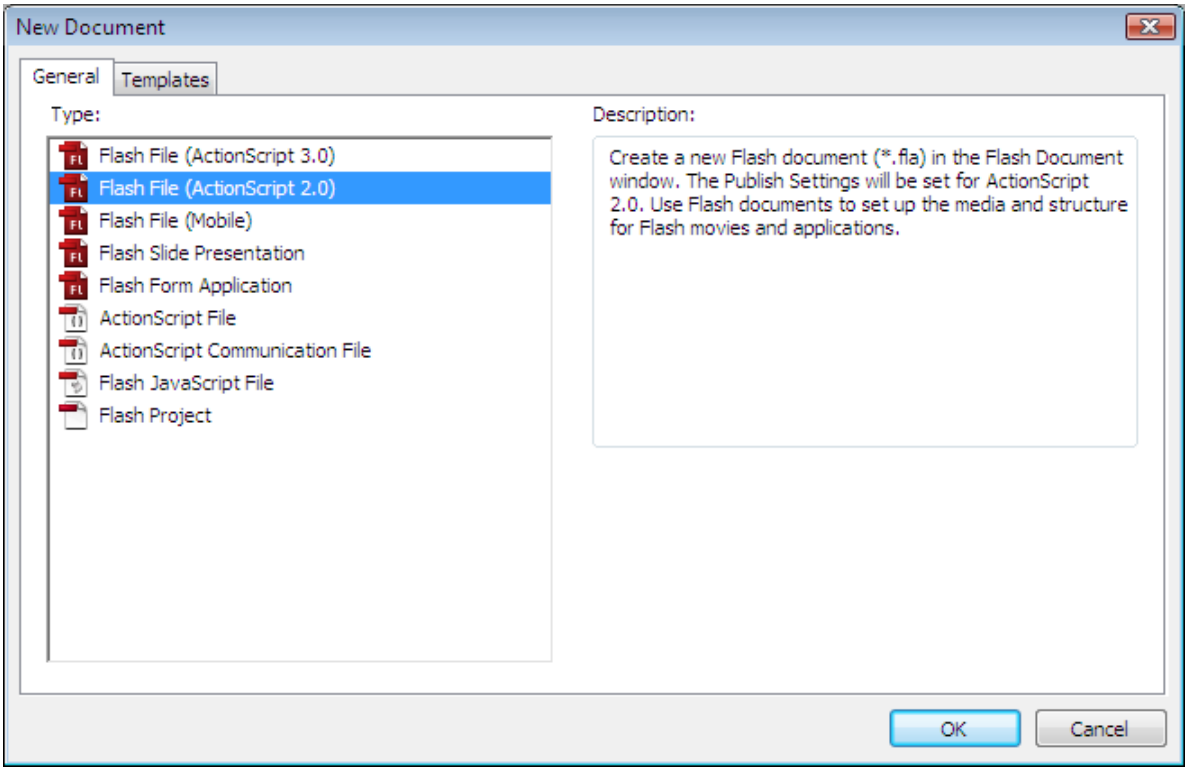
Here's a quick idea of what the theme will look like. I drew this up quickly in Fireworks before making this theme; it's a good idea to do this so you can get a feel for the size and positioning of the elements in your theme.

All that said, lets begin...

GUIDE

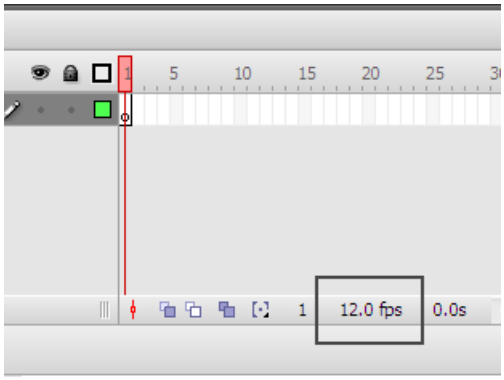
1 SETTING UP THE DOCUMENT

Select FILE -> NEW from the toolbar. When the New Document window appears select 'Flash File (ActionScript 2.0)' from the options. We choose this option rather than the Mobile one so we can use the default Flash debugger rather than Device Central (which will probably give you lots of warnings and not work very well).

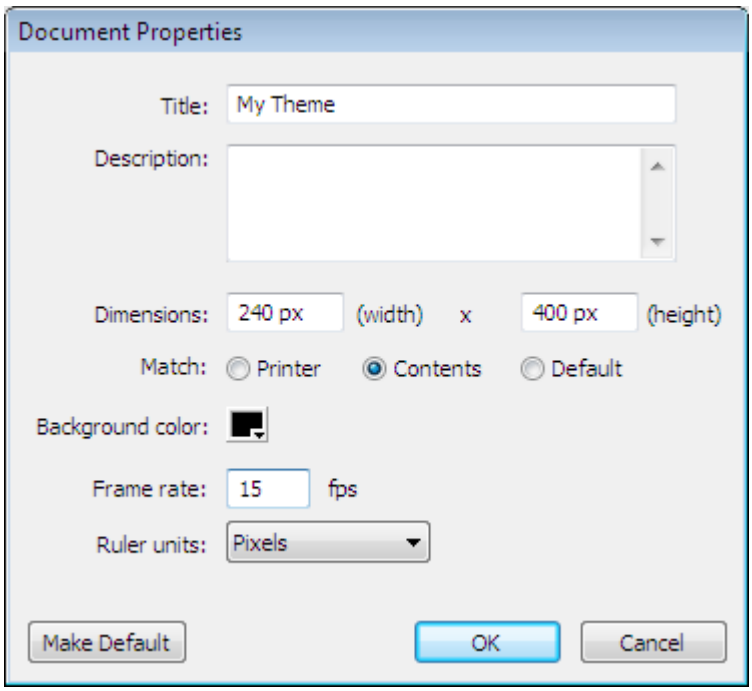


However when it comes to publishing for testing or use on the phone, you must remember to change the publishing profile.

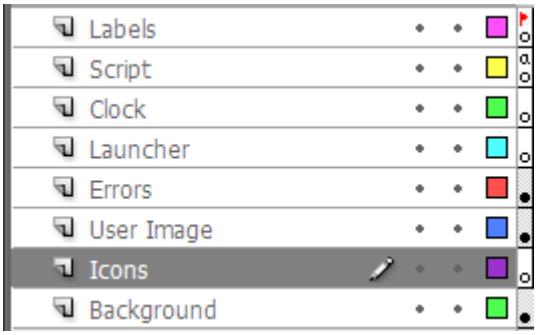
Now we have a new document we need to set it up with the right properties. Double click on the **FPS display** below the **timeline** to open the **document properties** window, or select MODIFY -> DOCUMENT from the toolbar (CTRL+J).



In the **Document Properties** window, enter a title for your theme, and set the dimensions of the **document** to 240 x 400 pixels. Choose the background color as black ('#000000'), and then set the frame rate of the document to 15FPS, and click 'OK'.



Now create seven new **layers** using the insert **layer** button and name them as shown opposite, these will be used to hold our **MovieClips** for the corresponding content. Now select the first frame of the '`Labels`' **layer** and enter '`LOADING`' for the **frame label** in the **properties inspector**.



Now to create our **XML** configuration file: first create a new folder with the name of your theme in the same directory as you are publishing your theme to, this will contain all of your external files for your theme. For the purposes of this guide we will be using the '**MyTheme**' folder. It is best practice to do this as it avoids cluttering up a user's **/LGAPP/Media/swf/theme/** folder. Then open a new document in Notepad® or your favourite text editor and enter the following **XML** code:

```
<root>
	<background num="1" />
	<icons>
```

```
            <icon0 str="Alarms" id="fid:0x10000ff7" dep="3,4" />
            <icon1 str="Bluetooth" id="fid:0x10000ff1" dep="4,6" />
            <icon2 str="Browser" id="" dep="3,1,1" />
            <icon3 str="Calendar" id="fid:0x10000fff" dep="3,3,1" />
            <icon4 str="Games & Apps" id="" dep="2,8,1" />
            <icon5 str="Memo" id="fid:0x20000ffd" dep="3,5" />
            <icon6 str="Music" id="fid:0x20000f3c" dep="2,4" />
            <icon7 str="My Stuff" id="" dep="2,1,1" />
            <icon8 str="Settings" id="" dep="4,3,1" />
            <icon9 str="Tools" id="" dep="3,6,1" />
            <icon10 str="Video Playlists" id="fid:0x20000f09" dep="2,5" />
            <icon11 str="Voice Recorder" id="fid:0x20000f3d" dep="2,6" />
      </icons>
</root>
```

Save this as '**config.xml**' inside the '**MyTheme**' folder. Now to get started! Import all of the images from the **Resources/Images/** folder into the Flash library by dragging and dropping them into the library. You could replace the icons or the other images with some of your own if you wish.

2 LOADING FILES

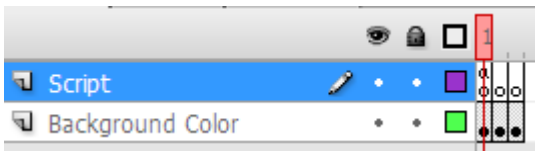
Create a new **symbol** in the **Library**, make it a **MovieClip**, and give it the name '`User Image`'. Return to editing the **main timeline** by selecting 'Scene 1' from just below the **timeline** display. Now select the '`UserImage`' **MovieClip** from the **library** and drag it onto the **stage**, then position it at x: 250px, y: 0px so that is outside the drawing area of the stage; finally give it the **instance name** '`userImage_mc`' in the properties inspector.

Now create another new **MovieClip** in your **library**, and give it the name '`Background`'. Create a new **layer** inside the '`Background`' **symbol**, and name it '`Script`', name the **layer** below it '`Background Color`'. Select the first **frame** of the '`Script`' layer, open the **Actions editor** and enter:

```
stop();
```

Now select the first **frame** of the '`Background Color`' **layer** and create a **Rectangle** of width: 240px, height: 400px and position it at x: 0px, y: 0px. Give the **rectangle** no outline, and a fill of '#000000' (Black).

Create two new **keyframes** in the '`Background Color`' **layer**, and two new **blank keyframes** in the '`Script`' **layer**, as shown in the screenshot opposite.



Select the second **frame** of the '`Background Color`' **layer**, and change the fill of the **rectangle** to '#212121' (Dark Gray), then select the third **frame** of the '`Background Color`' **layer** and change the fill of the **rectangle** to '#000033' (Dark Blue). You can of course use whatever colors you like in place of these. Now return to the **main scene**, select the first **frame** of the '`Background`' **layer** and drag the '`Background`' **MovieClip** from your **library** onto the **stage**, and position it at x: 0px, y:0px and give it the **instance name** '`background_mc`'.

Select the first **frame** of the '`Errors`' **layer**, and create a **dynamic text-box** of width: 210px, height:230px and position it at x: 15px, y:80px. Give the **text-box** the **instance name** '`error_tf`', set the font-color to '#FFFFFF' (white), set the font-size to 20pts and make the text center-aligned. Now select the first **frame** of the '`Script`' **layer**, open the **Actions editor** and enter the following **ActionScript** code:

```
stop();//Stop the playhead from continuing
```

```
//Declare variables
var errorMessage:String = "";//For any errors loading XML or user icon
var thisDirectory:String = this._url.slice(0, this._url.lastIndexOf("/"))+"/";
/*Gets this file's current location, takes off anything at and after the last occurence of
the
forward slash "/" character, then re-adds the "/" to the end to give the directory of this
file.*/
var iconLinkArr:Array = new Array();//Array to hold dep and id values for our icons
var backgroundNum:Number = 1;//User-selected background number
var currentIconNum:Number = 0;//Currently selected icon in the reel
var maxIconNum:Number = 12;//Max. number of icons in the reel

//Hide the show/hide shortcut arrows
_root.idleChanger_mc.unloadMovie();

//Hide the normal phone menu
_root.mc_idle.idelLauncher.gotoAndStop(11);

//Hide the operator text
_root.cur_main = "shortCut";
platform.setIdleCondition();

//Hide the shortCut launcher
_root.mc_idle.shortCut_mc._y = 401;

//Function to load the configuration data from the XML file
function loadXML() {
	var configXML = new XML();//Create a new XML document
	configXML.ignoreWhite = true;//Ignore white-space
	configXML.onLoad = function(success:Boolean) {
		if (success) {
			//Import XML to variables
			var rootNode:Array = configXML.firstChild.childNodes;
			var backgroundNode:XMLNode = rootNode[0];
			var iconNode:Array = rootNode[1].childNodes;//Array type rather than
XML node
			//as it holds many XML nodes
			backgroundNum = Number(backgroundNode.attributes.num);//Get the
background number

			//Loop through iconNode and put all icon properties into iconLinkArr
			for (var i:Number = 0; i<maxIconNum; i++) {
				iconLinkArr[i] = new Object();
				iconLinkArr[i].__label = iconNode[i].attributes.str;
				iconLinkArr[i].__id = iconNode[i].attributes.id;
				iconLinkArr[i].__dep = iconNode[i].attributes.dep;
			}

			//Now load user image
			loadUserImage();

		} else {
			//Could not find XML file or it is not a valid XML file
			//Show an error message to the user so they know whats up.
			errorMessage += "Sorry we could not find your config.xml file,"
			+ " it should be in the following place: "
			+ " thisDirectory + "MyTheme/"
			+ "\n" //New line character
			+ "Please put it in the right place.";

			//Now load user image
			loadUserImage();
		}
	};
	configXML.load(thisDirectory+"MyTheme/"+"config.xml");
}

function loadUserImage() {
	//Create our MovieClipLoader, we use this instead of the straight
MovieClip.loadMovie()
	//method so we know whether the icon was loaded successfully
	var imageLoader:MovieClipLoader = new MovieClipLoader();

	//Create our listener for MovieClipLoader
```

```
	var imageListener:Object = new Object();
	//Occurs when file has loaded and is ready to be used
	imageListener.onLoadInit = function() {
		checkErrors();
	};
	//Occurs when file cannot be found or is an invalid type
	imageListener.onLoadError = function() {
		//Add a couple of new line characters to error message if it already has some
		//text in it (from errors loading the XML)
		if (errorMessage != "") {
			errorMessage += "\n\n";
		}
		errorMessage += "Sorry we could not find your user_image.jpg file,"
		+ " it should be in the following place: "
		+ thisDirectory + "MyTheme/"
		+ "\n" //New line character
		+ "Please put it in the right place.";
		checkErrors();
	};
	imageLoader.addListener(imageListener);
	imageLoader.loadClip(thisDirectory+"MyTheme/"+"user_image.jpg",userImage_mc);
}

function checkErrors() {
	if (errorMessage == "") {
		//No errors, we're fine to continue
		gotoAndStop("INIT");
	} else {
		//Oops something's up, output the error to the text box
		error_tf.text = errorMessage;
	}
}

//Function is called when user exits the phone NYX menu
function effectIn() {
	//We need to hide the normal launcher again

	//Hide the show/hide shortcut arrows
	_root.idleChanger_mc.unloadMovie();

	//Hide the normal phone menu
	_root.mc_idle.idelLauncher.gotoAndStop(11);

	//Hide the operator text
	_root.cur_main = "shortCut";
	platform.setIdleCondition();

	//Hide the shortCut launcher
	_root.mc_idle.shortCut_mc._y = 401;

	gotoAndStop("MAIN");
}

//Function is called when user opens the phone NYX menu
function effectOut() {
	gotoAndStop("MENU");
}

//Event call function
function eventCall(dep:String, id:String) {

	if (!platform) {
		return undefined;
		//Don't do anything when you test it on the Computer
		//because the apps aren't there
	}


	if (id == undefined || id == "") {
		//No id means it's not an application,
		//so open the NYX menu at the page
		//specified by the dep

		platform.setMenuOn();
		_root.idleQuick = true;
```

```
                _root.QuickSet(dep);
                return; //Stop going any further in this function
        }

        //If we got this far it must be an application
        //Use the id to launch the correct application
        platform.launch_module(id);
}

//Start loading XML
loadXML();
```

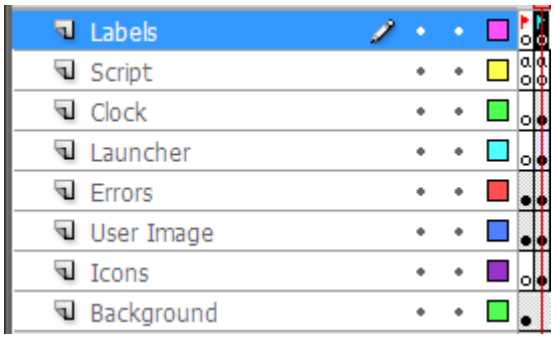
So what does this **ActionScript** actually do? In the first section we declare some **variables** we're going to need to keep track of what's going. The function `loadXML()` loads the '**config.xml**' file and extracts the information into some of the **variables** we declared. After it's done loading, it calls the `loadUserImage()` function. This function tries to load the '**user_image.jpg**' file from the '**MyTheme**' directory.

After it has tried to load the **image**, the **ActionScript** then calls `checkErrors()` to see if we had any problems loading the XML file or user image. If there are no errors, we continue on to the '`INIT`' **frame** (don't worry, we haven't created that yet), otherwise we show the user an error message. If you test the file now (CONTROL -> TEST MOVIE from the toolbar, or press CTRL+ENTER) you will get an error message saying it could not find the '**user_image.jpg**' file. To solve that, copy the '**user_image.jpg**' file from the '**Resources/User Image/**' folder included in this guide to the '**MyTheme**' folder.

The `effectIn()` **function** is called by the NYX menu when it is exited, and the `effectOut()` **function** is called when the NYX menu is opened. The `effectIn()` **function** hides the normal phone interface so we can use our own and move the **playhead** to the 'MAIN' **frame**, which is where our theme will run; We do not move the **playhead** to the '`LOADING`' or '`INIT`' **frames**, because we do not want to load anything more times than is necessary. The `effectOut()` **function** will be used to move the **playhead** to a **frame** '`MENU`' where only the background will be visible, to stop our theme showing through underneath the NYX menu.

Now we're really ready to get started on this theme!

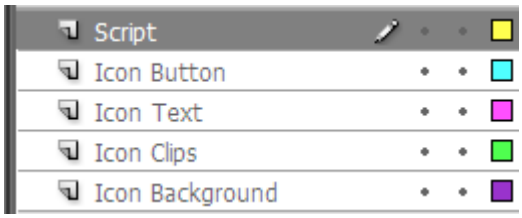
Select the second **frame** of the '`User Image`' **layer** and create a new **keyframe**. Create a new **frame** for the second **frame** of the '`Background`' layer, and new **blank keyframes** for the second **frame** of all the other **layers** as shown. Select the second **frame** of the '`labels`' **layer** and give the **frame** the name '`INIT`' in the **properties inspector**.



Select the second **frame** of the '`User Image`' **layer** and select the '`userImage_mc`' **MovieClip**. Move it to x: 120px, y: 270px in the **properties inspector**.

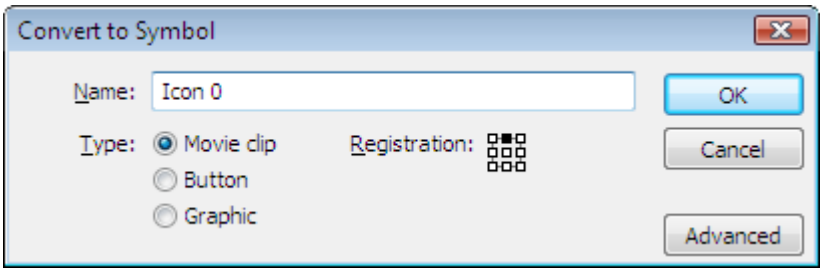
3 CREATING THE ICON REEL

Create a new **MovieClip** named '`Icon Reel`' in the **Library**, and create four new **layers** inside this **MovieClip**, name them as shown opposite.



Drag all of the large icon **images** (`'Alarms', 'Bluetooth', 'Browser', 'Calendar', 'Games & Apps', 'Memo', 'Music', 'My Stuff', 'Settings', 'Tools', 'Video Playlists', 'Voice Recorder'`) from your **library** onto the first **frame** of the '`Icon Clips`' **layer**, it doesn't really matter where. Flash may display some of them at different sizes because of their resolutions, so make sure all the **images** are set to width: 64px, height: 84px.

Now select the '`Alarms`' **image** on the stage and press function key F8 (or select MODIFY -> CONVERT TO SYMBOL… from the toolbar) to convert it to a **symbol**. Give it the name '`Icon 0`' and make the **registration point** of the **symbol** top-center.



Give this **symbol** the **instance name** '`icon0_mc`' and make sure it is positioned at y: 0px. The x position does not matter at the moment, as we will be using **ActionScript** to order and manipulate all of the icons. You now need to repeat this process for all of the icon **images** in alphabetic order (this is important otherwise your icons will launch the wrong applications) except give the next **image** ('`Bluetooth`') the name '`Icon 1`', and give the '`Icon 1`' **symbol** on the **stage** the **instance name** 'icon1_mc' and so on for all of the icons, taking you up to the '`Voice Recorder`' icon which should be '`Icon 11`' in your **library** and have the **instance name** '`icon11_mc`' on the **stage**. Make sure all of the icons have their y-position as 0px.

Now select the first **frame** of the '`Icon Background`' **layer** and drag the '`Icon Selected Background`' **image** onto the **stage**, position it at x: -31px, y: 0px.

Select the first **frame** of the '`Icon Text`' **layer**, and create a new **dynamic text box**. Give it the following properties: width: 120px, height: 30px, x: -60px, y:64px, font-color: '#FFFFFF', font-size: 16pts, text-align: center; finally give it the **instance name** '`label_tf`'.

Create a new **Button symbol** in your **library**, and name it '`Reel Button`'. This will be used to launch the currently selected icon. Leave all of the **frames** of the **button** blank except the '`Hit`' **frame**, where you will need to create a new **keyframe**. In the '`Hit`' **keyframe** create a **rectangle** of width: 60px, height: 60px and position it at x: -30px, y: -30px.

Return to editing your '`Icon Reel`' and drag the '`Reel Button`' **button** onto the '`Icon Button`' **layer**. Position it at x: 0px, y: 30px, and give it the instance name '`icon_btn`'.

Now enter the following **ActionScript** code in the first **frame** of the '`Script`' **layer**

```
//Declare variables for the icon reel
var isReelMoving:Boolean = false; //Whether the icon reel is moving
```

```
var iconWidth:Number = 64; //Keep track of the size of our icons

var themeRoot:MovieClip = this._parent; //So we can access the root of our theme easily

function initReel() {
	for (var i:Number = 0; i < _parent.maxIconNum; i++) {
		var iconMc:MovieClip = this["icon" + i + "_mc"];

		//Only 5 or six icons are going to be visible at each time
		//put the first 6 starting in the center
		//then loop the next 6 around
		if (i < 6) {
			iconMc._x = 0 + (iconWidth * i);
		} else {
			iconMc._x = 0 - (iconWidth * (_parent.maxIconNum - i));
		}
	}

	//Update label
	updateIconLabel();
}

//Move reel to the left or right
//Can set speed to 0 or 1 (faster)
function moveReel(moveSpeed:Number, doReverseRotate:Boolean) {
	if (isReelMoving == false) {
		//Only move reel if it isn't already moving

		//Make sure function can't be called again until
		//the reel has stopped moving
		isReelMoving = true;

		var directionNum:Number = 1;


		if (doReverseRotate == true) {
			//Move icons left

			directionNum = -1;

			//Loop round and put the last icon to the left on the right
			//Get the number of the last icon to the left
			var lastMcNum:Number = _parent.currentIconNum + 6;
			if (lastMcNum > 11) {
				lastMcNum -= 12;
			}
			var lastMc:MovieClip = this["icon" + lastMcNum + "_mc"];
			lastMc._x = (iconWidth * 6);
		} else {
			//Move icons right

			//Get number of the last icon to the right
			var lastMcNum:Number = _parent.currentIconNum + 5;
			if (lastMcNum > 11) {
				lastMcNum -= 12;
			}

			var lastMc:MovieClip = this["icon" + lastMcNum + "_mc"];

			lastMc._x = -(iconWidth * 7);
		}

		//Animation variables
		var xIncrement:Number = 18;
		var maxFrameCount:Number = 3;

		if (moveSpeed == 1) {
			xIncrement = 27;
			maxFrameCount = 2;
		}

		var frameCount:Number = 0;
		this.onEnterFrame = function() {
			//Move icons
			for (var i:Number = 0; i < _parent.maxIconNum; i++) {
```

```
                                var iconMc:MovieClip = this["icon" + i + "_mc"];
                                if (frameCount < maxFrameCount) {
                                        iconMc._x += (directionNum * xIncrement);
                                } else {
                                        iconMc._x += (directionNum * 10);
                                }
                        }

                        if (frameCount == maxFrameCount) {
                                //Adjust the currently selected icon
                                _parent.currentIconNum -= directionNum;

                                if (_parent.currentIconNum < 0) _parent.currentIconNum = 11;
                                if (_parent.currentIconNum > 11) _parent.currentIconNum = 0;

                                //Update the icon label
                                updateIconLabel();

                                //Finished moving the reel
                                delete this.onEnterFrame;
                                isReelMoving = false;

                        } else frameCount++;
                }
        }
}

//Update label to current icon name
function updateIconLabel() {
        var iconLabel:String = _parent.iconLinkArr[_parent.currentIconNum].__label;

        if (iconLabel == undefined) {
                //Can happen if XML takes a while to load, if so
                //do it again and hopefully it'll be loaded
                updateIconLabel();
        } else {
                //Update the icon label
                label_tf.text = iconLabel;
        }
}

//When icon button is pressed
icon_btn.onPress = function() {
        //Show circle animation
        _root.show_pressAni(120, 110 + 25);
        if (isReelMoving == false) {
                //Haptic vibration
                platform.touch_sound();
        }
}

//When icon button is released
icon_btn.onRelease = function() {
        //Hide circle animation
        _root.out_pressAni();

        //Only call function if its no longer moving
        if (isReelMoving == false) {
                //Get id and dep of current icon
                var idCode:String = themeRoot.iconLinkArr[themeRoot.currentIconNum].__id;
                var depCode:String = themeRoot.iconLinkArr[themeRoot.currentIconNum].__dep;

                themeRoot.eventCall(depCode, idCode);
        }
}

icon_btn.onReleaseOutside = function() {
        //Hide circle animation
        _root.out_pressAni();
}

//Initialise the reel
initReel();
```

So, you're probably wondering what all that math is about. It might look scary, but it's not difficult really. The `initReel()` **function** sets up the positions of all of the icons in the reel. To do this it puts the first icon in the middle, and the next five after that to the right. Then to give the effect of the reel 'wrapping round', it places the next six icons to the left, with the last icon (11) being the icon just to the left of the first icon (0).

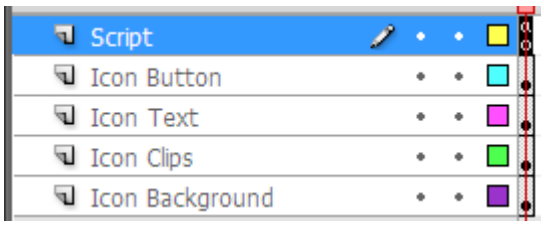
The `moveReel()` **function** does what it says! Its good practice to name your **functions** like this – give all **functions** descriptive names and give methods names that begin with verbs. How does it move the reel? First notice the **function** takes a **Number** parameter `moveSpeed`, which can be either `0` or `1` and sets how quickly the animation should occur, then there is **Boolean** parameter `doReverseRotate`; if this is `true` it rotates the reel to the left, otherwise it rotates the reel to the right. Then it finds the furthest icon to the left or right (depending on which direction it is rotating) and sticks it on at the opposite side of the reel. Then we use an `onEnterFrame` **function**, which is called at the frame-rate of the document, to move all of the icons to the left or right.

We also set up the handlers for the icon button – we show the built-in circular press animation by calling the `show_pressAni()` **function** when the button is pressed using the `onPress` **handler**, and hide it when it is let go using the `out_pressAni()` **function** with the `onRelease` **handler**. Notice we only do something if the reel isn't moving, this because otherwise we might call the wrong **function** as the currently selected icon might have just changed.

Return to editing the **main scene**, select the **frame** 2 of the '`Icons`' **layer** and drag the '`Icon Reel`' MovieClip from your **library** onto the **stage**. Place it at x: 120px, y: 110px, and give it the **instance name** '`iconReel_mc`' in the **properties inspector**.

4 CREATING THE LAUNCHER

Create a new **MovieClip** in your **library** and give it the name '`Launcher`'. Create three new **layers** inside this **symbol**, and give them names as shown opposite. Select the first **frame** of the '`Launcher Background layer`', and drag the '`Launcher Background`' **image** onto the **stage**. Place it at x: -120px, y: 0px.



Now select the first **frame** of '`Launcher Icons`' **layer**, and drag the small '`Inbox`', '`Sent`', '`Draft`' and '`Menu`' icons onto the **stage**. Place the '`Inbox`' icon at x: -98px, y: 23px; the '`Draft`' icon at x: -41px, y: 23px, the '`Sent`' icon at x: 15px, y: 23px and finally put the '`Menu`' icon at x: 72px, y: 23px.

Select the first **frame** of the '`Icon Labels`' **layer**, and create a new **dynamic text box**. Give it the following properties: width: 50px, height: 20px, x: -111px, y: 46px, font-size: 12pts, font-color: '#FFFFFF', text-align: Center; and finally set the **variable** to '`inboxCount`'. Create another three **text-boxes** with same properties, except give the first: x:-54px**, variable**: '`draftCount`'; the second: x: 2px, **variable** '`sentCount`' and the last: x: 58px, no **variable**. Enter the **text** '`Menu`' into the last **text-box** on the right. Your **stage** should look similar to the picture to the right.

Create a new **symbol** in your **library**, and make it a **Button**. Give it the name '`Icon Button`'. Select the 'Down' **frame** of the '`Icon Button`' and insert a new **keyframe**. Drag the '`Launcher Button`' **image** onto the **stage** and place it at x: -27px, y: -25px. Now insert a new **blank keyframe** into the '`Hit`' **frame** of the

button, and draw a **rectangle** of w: 50px, height: 50px; the style of the **rectangle** does not matter because it will not be visible, it just serves as a hit area for the **button**. Place the **rectangle** at x: -25px, y: -25px.

Return to editing the 'Launcher' **MovieClip**, and place four of the 'Launcher Button' **buttons** onto the first **frame** of the 'Icon Buttons' **layer**. Put the first at x: -85px, y: 41px and give it the **instance name** 'inbox_btn'; the second at x: -29px, y: 41px and give the **instance name** 'draft_btn'; the third at x: 27px, y: 41px and give it the **instance name** 'sent_btn'; place the last one at x: 83px, y: 41px and give it the **instance name** 'menu_btn'.

Select the first **frame** of the 'Script' **layer** in the 'Launcher' **MovieClip**, and enter the following **ActionScript** code:

```
//Declare message counters
//Set the inboxCount, sentCount and draftCount variables
//to a number so they don't appear as undefined
var inboxCount:Number = 0;
var draftCount:Number = 0;
var sentCount:Number = 0;

//This will keep the id for stopping and starting
//a timer to check message counts through the
//setInterval and clearInterval functions
var intervalID:Number;

function updateMessageCounters() {
	//These come from the NYX
	inboxCount = _root.inboxCount;
	draftCount = _root.draftCount;
	sentCount = _root.sentCount;

	//Check for undefined if handset hasn't
	//yet loaded data to the NYX
	if (inboxCount == undefined) inboxCount = 0;
	if (draftCount == undefined) draftCount = 0;
	if (sentCount == undefined) sentCount = 0;
}

//Check messages every now and then
function startMessageCheck() {
	//Update the message count every 30 seconds
	//need to use intervalID so we can stop it later on
	intervalID = setInterval(updateMessageCounters, 30000);

	//Update counters now
	updateMessageCounters();
}

//Stop checking messages
//used for when menu is opened
function stopMessageCheck() {
	clearInterval(intervalID);
}

//Set up the buttons to launch applications
var themeRoot:MovieClip = this._parent; //So we can access the parent of this clip easily

inbox_btn.onPress = draft_btn.onPress = sent_btn.onPress = menu_btn.onPress = function() {
	platform.touch_sound(); //Make a haptic vibration
}

inbox_btn.onRelease = function() {
	themeRoot.eventCall("", "fid:0x27e00000");
}

draft_btn.onRelease = function() {
	themeRoot.eventCall("", "fid:0x27c00000");
}

sent_btn.onPress = function() {
	themeRoot.eventCall("", "fid:0x27a00000");
```

```
}

menu_btn.onPress = function() {
	themeRoot.eventCall();
}
```

Let's take a quick look at that **ActionScript**. First we declare some **variables** to track the inbox, sent and draft count and another to keep track of a timer so we can check messages at regular intervals.

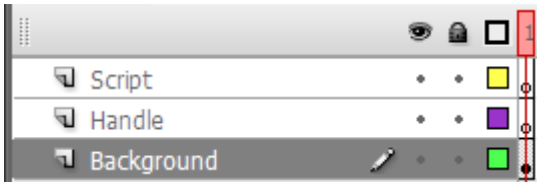
The `updateMessageCounters()` **function** uses the `_root` **keyword** to get the message counts from the NYX menu. The `startMessageCheck()` and `stopMessageCheck()` **functions** are quite self-explanatory, they start and stop the regular checking of the messages.

We then set the **handlers** for the launcher **buttons**. They all have the same `onPress` **handler**, which causes a haptic vibration. When the buttons are released we call the `eventCall()` **function** defined in the first **frame** of the **main scene** our theme, don't worry about those values in the `eventCall()` **function** and where they come from, there is a list in the reference at the end of this guide with codes for all of the phone functions.

Now return to the **main scene** and select the second **frame** of the '`Launcher`' **layer**. Drag the '`Launcher`' **MovieClip symbol** from your **library** onto the **stage**, give it the **instance name** '`launcher_mc`', and place it at x: 120px, y: 330px.

5 CREATING THE SCROLLBAR CONTROL FOR THE ICON REEL

Create a new **MovieClip symbol** in your **library** with the name '`Scrollbar`'. Make two new **layers** inside this **MovieClip**, and name them as shown opposite.



Then select the first **frame** of the '`Background`' **layer** and drag the '`Scrollbar Background`' **image** from your **library** onto the **stage**. Place it at x: -110px, y: -18px. Now drag the '`Scrollbar Handle`' **image** onto the first **frame** of the '`Handle`' **layer**, and press F8 to convert it to a **symbol**. Set **the registration point** if the **symbol** as middle-center, make it a **button** type, and name it '`Handle Button`'. Place it at x: 0px, y: 0px and give it the **instance name** '`handle_btn`'.

Now select the first frame of the '`Script`' **layer**, open the **Actions editor** and enter the following **ActionScript** code:

```
//Some variables to keep track of what we're up to
//These are for tracking how fast we want to move the reel
var delayCount:Number = 0;
var scrollSpeed:Number = -1;

//So we can access the root of our theme easily
var themeRoot:MovieClip = this._parent;

//Scroll the reel at a regular interval if the scrollbar is
//over to one side far enough
function scrollIconReel() {
	var maxDelayCount:Number = 8;
	//Check how far the user has moved the scrollbar over

	if (Math.abs(handle_btn._x) < 10) {
		//Do nothing
		scrollSpeed = -1;
```

```
	} else {
		if (Math.abs(handle_btn._x) < 50) {
			//Scroll slowly
			scrollSpeed = 0;
		} else {
			//Scroll more quickly
			scrollSpeed = 1;
			maxDelayCount = 4;
		}
	}

	//Move reel if its time
	if (delayCount == 0 && scrollSpeed >= 0) {

		var doReverse:Boolean = false;

		//Check which side scrollbar is on
		if (handle_btn._x < 0) {
			doReverse = true;
		}

		//Move reel
		themeRoot.iconReel_mc.moveReel(scrollSpeed, doReverse);
	}

	delayCount++;

	if (delayCount >= maxDelayCount) {
		delayCount = 0;
	}
}

//Event handlers for our button
handle_btn.onPress = function() {
	platform.touch_sound(); //Haptic vibration

	//Maximun range of the handle
	var xLimit:Number = 110 - (this._width / 2);

	//Start dragging the handle
	//and keep its y-position constant
	startDrag(this, false, xLimit, this._y, -xLimit, this._y);

	//Stop moving handle back to center (if it is)
	delete this._parent.onEnterFrame;

	//Start scrolling icons
	this._parent.onEnterFrame = scrollIconReel;
}

handle_btn.onRelease = handle_btn.onReleaseOutside = handle_btn.onDragOut = function() {
	//Stop dragging our handle
	stopDrag();

	//Stop checking position of handle
	delete this._parent.onEnterFrame;

	//How far the handle has moved
	var xDisplacement:Number = handle_btn._x;
	//How much the handle will move each frame (rounded)
	var xIncrement:Number = Math.round(handle_btn._x / 3);

	var frameCount:Number = 0; //Number of animation frames

	//Return handle to center
	this._parent.onEnterFrame = function() {
		handle_btn._x -= xIncrement; //Change x position

		frameCount++; //Update frame counter

		if (frameCount == 3) {
			//Make sure handle is centered
			handle_btn._x = 0;

			//Stop animation
```

```
                    delete this.onEnterFrame;
                }
        }
}
```

The **function** `scrollIconReel()` is called every **frame** once the handle **button** is selected, it checks how far over the handle is, and then calls our `moveReel()` **function** we defined in the '`Icon Reel`' **symbol** to actually move the reel.

Then we set up the **event handlers** for our scrollbar **button**: when the user presses it, we give them some haptic feedback, and then call the `scrollReel()` **function** to check the position of our handle.

When the user releases the **button**, we stop dragging it around and use an `onEnterFrame` **function** to return the handle to its center position.

Now drag the '`Scrollbar`' **MovieClip** onto the second **frame** of the '`Errors`' **layer** (we may as well reuse it as it's no longer needed), position it at x: 120px, y: 230px, and give it the **instance name** '`scrollbar_mc`'.

6 CREATING THE CLOCK

There are many ways you can show the time, we're just going for a simple 24 hour clock for this theme, but you could make yours show the date, or show the time in a 12 hour format if you like.

Then insert a **keyframe** into the second **frame** of the '`Clock`' layer in the **main scene**.

Now create a new **MovieClip symbol** in your **library** and name it '`Clock`'. Now create a new **layer** inside this, and name your **layers** as shown opposite.

Select the '`Clock Text`' **layer**, and create a new **dynamic text box**. Give it the following properties: width: 100px, height: 40px, x: -50px, y: -20px, font-size: 30pts, font-color: '#FFFFFF', and make it center-aligned. Set the variable of the text-box to '`timeText`'.

Now enter the following **ActionScript** code into the first **frame** of the '`Script`' **layer**:

```
var timeText:String = ""; //Time display
var intervalID:Number; //interval id for clearInterval function

//Update clock display
function updateClock() {
      //Get the current Date/Time
      var currentDateTime:Date = new Date();

      //Hours & minutes
      var hours:String = String(currentDateTime.getHours());
      var mins:String = String(currentDateTime.getMinutes());

      //Add zero onto the beginning if single digit
      if (length(hours) < 2) hours = "0" + hours;
      if (length(mins) < 2) mins = "0" + mins;

      //Update clock text
      timeText = hours + ":" + mins;
}

//Start checking time at regular intervals
function startClock() {
      updateClock();
```

```
        intervalID = setInterval(updateClock, 30000)
}

//Stop checking time at intervals
function stopClock() {
        clearInterval(intervalID);
}
```

The `updateClock()` **function** gets the current time and updates the '`timeText`' **variable**. The `startClock()` **function** makes our script check the time every 30 seconds (which is fine because we are only showing hours and minutes) and the `stopClock()` **function** stops our script checking the time (to use when the menu is opened).

Now return to editing the **main scene** and select the second **keyframe** of the '`Clock`' **layer**. Drag your '`Clock`' **symbol** onto this **layer**, and give it the **instance name** '`clock_mc`', then position it on the stage at x: 120px, y: 72px.

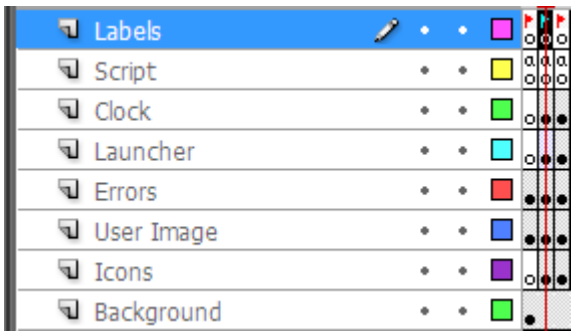
Finally select the third **frame** of '`Script`' layer in the **main timeline**, and enter the following **ActionScript** code in the **Actions editor**:

```
//This function will center the user image horizontally,
//as we don't know exactly how wide it could be
function centerUserImage() {
        //Get the width of the image
        var imageWidth:Number = userImage_mc._width;

        //Calculate the center position
        var centerPosition:Number = 120 - (imageWidth / 2);

        userImage_mc._x = centerPosition; //Center the MovieClip
}

//Center user image now
centerUserImage();

//Set users chosen background
background_mc.gotoAndStop(backgroundNum)

//Finally go to main frame
gotoAndStop('MAIN');
```

The function `centerUserImage()` centers the '`userImage_mc`' horizontally, since the user may make their image wider or smaller than the original one. We call it straight away to center the image. Then we set the background to the one the user chose in their XML configuration file, and finally we jump to the '`MAIN`' frame.

7 MAKING THE 'MAIN' SCENE

Insert a new **keyframe** into the third **frame** of all of your **layers** except the '`Background`' **layer**, were you want to insert just a normal **frame** instead. Your main timeline display should now look like that opposite.



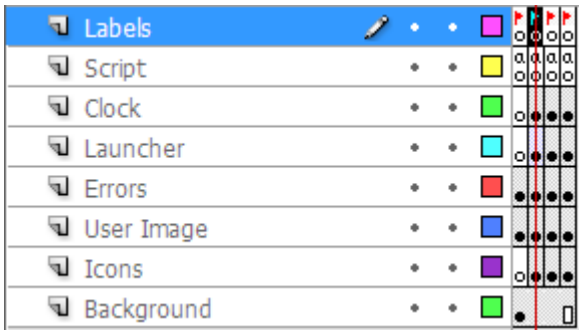
Select the third **frame** of the '`Labels`' **layer** and give it the **frame** name '`MAIN`'. Now select the third **frame** of the '`Script`' **layer** and enter the following

ActionScript code:

```
//Start Clock
clock_mc.startClock();

//Check SMS counters
launcher_mc.startMessageCheck();

//Make sure image is in right place
userImage_mc._y = 270;

//Update icon label
iconReel_mc.updateIconLabel();
```

This starts the clock and message counters using the **functions** we created earlier. We make sure the user image is in the right place because we set its position programmatically, so making a **keyframe** and dragging it off the stage won't work for hiding it on the menu screen – we have to hide it with **ActionScript** as well.

8 MAKING THE 'MENU' SCENE



Insert a new **keyframe** into the fourth frame of all of your **layers** except the background **layer**, where you just want to insert a **frame**. Give the fourth **frame** of the '`Labels`' **layer** the name '`MENU`'. Now drag the `scrollbar_mc`, `clock_mc`, `userImage_mc` and `launcher_mc` and `background_mc` **MovieClips** off the visible area of the **stage**.

Now create a new **MovieClip** in your **library** named '`Menu Background`'. In this **MovieClip** create a **rectangle** of width: 240px, height: 400px and fill it with a light color, I chose a light gray ('#999999'). Then position the **rectangle** at x: 0, y: 0. Return to editing the **main scene**, and drag the '`Menu Background`' **symbol** onto the fourth **frame** of the '`Clock`' **layer**. Then position it at x: 0, y: 0.

Enter the following **ActionScript** code in the fourth **frame** of the '`Script`' **layer**:

```
//Stop Clock
clock_mc.stopClock();

//Stop checking SMS counters
launcher_mc.stopMessageCheck();

//Hide user image
userImage_mc._y = -1000;
```

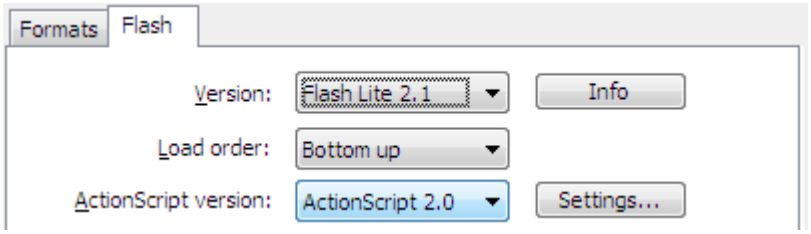
This just stops the clock and message counters from updating while the menu is open because they won't be visible, and we want the CPU to have to do as little as possible. We also hide the user image.

9 PUBLISHING YOUR THEME

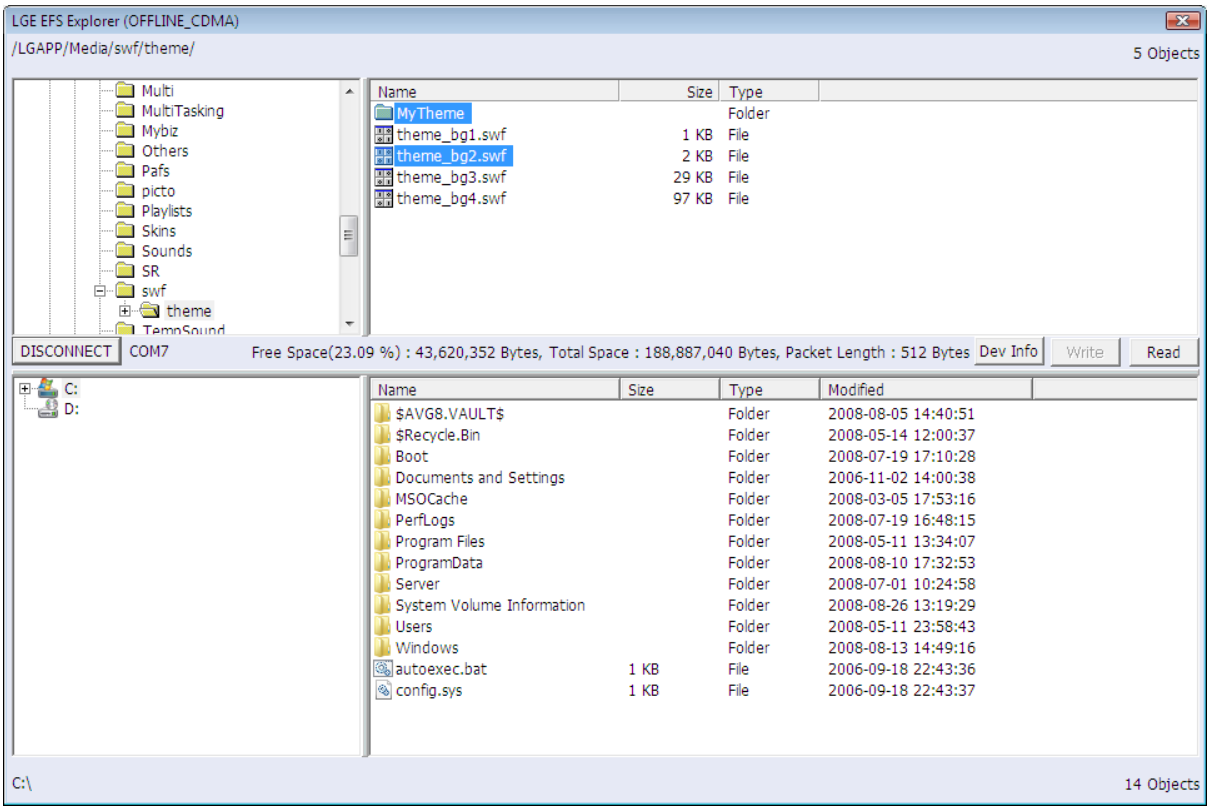
One more step to go and you're done (you can try testing your theme by pressing CTRL+ENTER, although the SMS counters will not work). Change the number of the background in the XML file and test again (it can set from **1 – 3**), to make sure that the background changes correctly.

Double-click on the frame-rate display to show the **document properties** and up the frame-rate of the document to 20FPS, so things run as smoothly as possible on your mobile device.

Now select FILE -> PUBLISH SETTINGS from the toolbar. Make sure only the Flash (.swf) format is selected in the Format tab, and then in the Flash tab change the Flash version to Flash Lite 2.1 (or Flash 7 if you are using an older version of Flash Professional). Now press publish and hey presto! You're theme is done.



Now you're going to want to test it on your phone, so rename the 'MyTheme.swf' file to 'theme_bg2.swf', and use EFS explorer to copy that and the 'MyTheme' folder to /LGAPP/Media/swf/theme/.



Then disconnect your phone, switch to the silver theme, and try it out!

It is good practice to test your theme with Theme Manager, as although it should work the same, sometimes it may not. For instructions on how to install Theme Manager, and get your theme running on Theme Manager, see the Theme Manager topic at http://ku990.co.uk/forum.

Have a play about changing the positions of objects and things such as font-sizes, to get the theme looking just right once you have tested it on your handset.

10 COMMENTS

A nice effect would be to have the Icons fade out towards the edges of the real… So I did not include it because it was too complex for this guide? Actually it wouldn't have been too hard, and it's possible in ActionScript 2.0, but it requires objects to support runtime bitmap caching, which sadly is only supported in Flash Player 8 or later or Flash Lite 3.0. You need to be careful about things like this: there are a couple of things that will work fine when you debug them in a normal Flash format, but when you publish to Flash Lite it may not work.

Another thing to get comfortable with is the debugger (Press CTRL+SHIFT+ENTER to start debugging your Movie). This will tell you all of the values of your variables, arrays, objects etc and the current properties of the items on the stage. All this information is very useful, and can help you check things such as whether the data you're loading from your XML files is really getting imported.

Here are a few ideas you can add yourself to make the theme your own:

- Add a 12 hour mode to the clock, and let the user choose the clock mode in the **XML**. You'll need a new **global variable** to remember the clock mode, to edit the `loadXML()` **function** to get the value from the XML, and add a second **frame** to the clock with an AM/PM **dynamic text-box**. You'll also need to edit the `updateClock()` **function** to give the correct time depending on the mode.
- Add a date display to the clock. This will need a new **dynamic text-box**, and a new set of **functions** to update the date, and start and stop checking the date at regular intervals.
- Add more background colors. You could add **images** instead of just colors as well.
- Let the user load an **animation** instead of an **image**. (This is easy and just requires changing the '.jpg' extension in the `loadUserImage()` **function** to '.swf')
- Let the user drag the userImage_mc, iconReel_mc and `clock_mc` **MovieClips** around the screen so they can swap the order of them.
- Let the user choose a menu background in the **XML**, to do this edit the '`Menu Background`' **MovieClip** in a similar way to the '`Background`' **symbol** and add a new line to **XML** file to hold the information.

REFERENCE

DEP AND ID CODES

Note: These are taken from an open v10E firmware KU990®. Some codes may vary between operators. It is always a good idea to load these from an external file, so the user can alter them if your values do not work.

For applications only the ID code is required, for menus only the DEP is required (when using the standard `eventCall()` function, see the code snippets)

Description	DEP	ID	Type
Main Menu			Menu
Dialing	1,1	event:dialing	Application
Contacts	1,2,1		Menu
Contacts – Search	1,2,1	fid:0x22e00000	Application
Contacts – Add New	1,2,2	fid:0x22f00000	Application
Contacts – Speed Dials	1,2,3	fid:0x22d00000	Application
Contacts – Groups	1,2,4	fid:0x22c00000	Application
Contacts – Settings	1,2,5	fid:0x22900000	Application
Contacts – Information	1,2,6		Menu
Contacts – Information – Service dial numbers		fid:0x22b00000	Application
Contacts – Information – Own number		fid:0x22a0000	Application
Contacts – Information – Memory info.		fid:0x23000000	Application
Contacts – Information – My business card		fid:0x22500000	Application
Call Logs	1,3,1		Menu
Call Logs – All calls	1,3,1	fid:0x2ce00000	Application
Call logs – Dialed calls	1,3,2	fid:0x2c500000	Application
Call logs – Received calls	1,3,3	fid:0x2c400000	Application
Call logs – Missed calls	1,3,4	fid:0x2c300000	Application
Call logs – Call charges	1,3,5		Menu
Call logs – Call charges – Call durations		fid:0x2cd00000	Application
Call logs – Call charges – Call costs		fid:0x2cc00000	Application
Call logs – Data volume	1,3,6	fid:0x2c200000	Application
Messaging	1,4,1		Menu
Messaging – Create new message	1,4,1	fid:0x27f00000	Application
Messaging – Inbox	1,4,2	fid:0x27e00000	Application
Messaging – Email	1,4,3	fid:0x27d00000	Application
Messaging – Drafts	1,4,4	fid:0x27c00000	Application
Messaging – Outbox	1,4,5	fid:0x27b00000	Application
Messaging – Sent items	1,4,6	fid:0x27a00000	Application
Messaging – Templates	1,4,7		Menu
Messaging – Templates – Text templates		fid:0x27900000	Application
Messaging – Templates – Multimedia templates		fid:0x27100000	Application
Messaging – Emoticons	1,4,8	fid:0x27800000	Application
Messaging – Settings	1,4,9		Menu
Messaging – Settings – Text message		fid:0x16700000	Application
Messaging – Settings – Multimedia message		fid:0x16600000	Application
Messaging – Settings – Email		fid:0x16500000	Application
Messaging – Settings – Videomail		fid:0x16400000	Application
Messaging – Settings – Voicemail		fid:0x16300000	Application
Messaging – Settings – Service message		fid:0x16200000	Application
Messaging – Settings – Info. Service		fid:0x16100000	Application
My stuff	2,1,1		Menu
My stuff – My images	2,1,1	fid:0x20000f0f	Application
My stuff – My sounds	2,1,2	fid:0x20000f0d	Application

My stuff – My videos	2,1,3	fid:0x20000f0e	Application
My stuff – My games & apps	2,1,4	fid:0x20000f0c	Application
My stuff – Flash contents	2,1,5	fid:0x20000f07	Application
My Stuff – Documents	2,1,6	fid:0x20000f06	Application
My stuff – My memory card	2,1,7	fid:0x20000f0a	Application
My stuff – Other files	2,1,8	fid:0x20000f08	Application
Camera	2,2	fid:0x20000f3e	Application
Muvee studio	2,3	fid:0x10000f38	Application
Music	2,4	fid:0x20000f3c	Application
Video playlists	2,5	fid:0x20000f09	Application
Voice recorder	2,6	fid:0x20000f3d	Application
FM radio	2,7	fid:0x20000f3a	Application
Games & Apps	2,8,1		Menu
Games & Apps – My games & apps	2,8,1	fid:0x20000fcf	Application
Games & Apps – Settings	2,8,2	fid:0x20000fcd	Application
Games & Apps – Space commando	2,8,3	fid:0x10000fc7	Application
Browser	3,1,1		Menu
Browser – Home	3,1,1	fid:0x20000fc6	Application
Browser – Enter address	3,1,2	fid:0x20000fc3	Application
Browser – Bookmarks	3,1,3	fid:0x20000fc5	Application
Browser – Saved pages	3,1,4	fid:0x20000fc4	Application
Browser – History	3,1,5	fid:0x20000fc7	Application
Browser – Settings	3,1,6	fid:0x20000fc1	Application
Google	3,2	fid:0x20000fc9	Application
Organiser	3,3,1		Menu
Organiser – Calendar	3,3,1	fid:0x10000fff	Application
Organiser – To do	3,3,2	fid:0x10000ffe	Application
Organiser – Date	3,3,3	fid:0x10000ffb	Application
Organiser – Settings	3,3,4	fid:0x20000ff9	Application
Alarms	3,4	fid:0x10000ff7	Application
Memo	3,5	fid:0x20000ffd	Application
Tools	3,6,1		Menu
Tools – Calculator	3,6,1	fid:0x10000ff5	Application
Tools – World clock	3,6,2	fid:0x10000ff4	Application
Tools – Converter	3,6,3		Menu
Tools – Converter – Currency		fid:0x20000fed	Application
Tools – Converter – Surface		fid:0x20000fec	Application
Tools – Converter – Length		fid:0x20000feb	Application
Tools – Converter – Weight		fid:0x20000fea	Application
Tools – Converter – Temperature		fid:0x20000fe9	Application
Tools – Converter – Volume		fid:0x20000fe8	Application
Tools – Converter – Velocity		fid:0x20000fe7	Application
USIM services	3,7	fid:0x20000ff	Application
Screen	4,1,1		Menu
Screen – Main screen theme	4,1,1	fid:0x20000f7f	Application
Screen – Clock & Calendar	4,1,2	fid:0x20000f7d	Application
Screen – Brightness	4,1,3	fid:0x20000f7c	Application
Screen – Backlight	4,1,4	fid:0x20000f7e	Application
Screen – Handset theme	4,1,5	fid:0x20000f7b	Application
Screen – Font	4,1,6	fid:0x20000f7a	Application
Profiles	4,2,1		Menu
Phone settings	4,3,1		Menu
Phone settings – Date & Time	4,3,1	fid:0x100ef000	Application
Phone settings – Power save	4,3,2	fid:0x100e4000	Application
Phone settings – Languages	4,3,3	fid:0x100eb000	Application
Phone settings – Security	4,3,4		Menu
Phone settings – Security – PIN code request		fid:0x200e3000	Application

Phone settings – Security – Handset lock		fid:0x200e2000	Application
Phone settings – Security – Auto key lock		fid:0x100e6000	Application
Phone settings – Connectivity	4,3,5		Menu
Phone settings – Connectivity – Network settings		fid:0x100ee000	Application
Phone settings – Connectivity – Access points		fid:0x100ed000	Application
Phone settings – Connectivity – USB connection mode		fid:0X10000fed	Application
Phone settings – Connectivity – Synch service		fid:0x10000ff0	Application
Phone settings – Connectivity – Streaming settings		fid:0x100e3000	Application
Phone settings – Connectivity – TV out		fid:0x100e2000	Application
Phone settings – Memory manager	4,3,6	fid:0x100ea000	Application
Phone settings – Touchpad calibration	4,3,7	fid:0x100e7000	Application
Phone settings – Reset settings	4,3,8	fid:0x100e8000	Application
Phone settings – Handset information	4,3,9	fid:0x100e9000	Application
Call settings	4,4,1		Menu
Call settings – Call divert	4,4,1	fid:0x2cb00000	Application
Call settings – Call barring	4,4,2	fid:0x2ca00000	Application
Call settings – Fixed dial numbers	4,4,3	fid:0x2c900000	Application
Call settings – Call waiting	4,4,5	fid:0x2c800000	Application
Call settings – Common setting	4,4,6	fid:0x2c700000	Application
Call settings – Video call setting	4,4,7	fid:0x1cf00000	Application
Flight mode	4,5	fid:0x100e5000	Application
Bluetooth	4,6	fid:0x10000ff1	Application

CODE SNIPPETS

LAUNCHING THE HANDSET MENU OR APPLICATIONS

```
//eventCall function
//used to launch handset apps and menus
function eventCall(dep:String, id:String) {

        if (!platform) {
                return undefined;
                //Don't do anything when you test it on the Computer
                //because the apps aren't there
        }


        if (id == undefined || id == "") {
                //No id means it's not an application,
                //so open the NYX menu at the page
                //specified by the dep

                platform.setMenuOn();
                _root.idleQuick = true;
                 root.QuickSet(dep);
                return; //Stop going any further in this function
        }

        //If we got this far it must be an application
        //Use the id to launch the correct application
        platform.launch module(id);
}
```

USING HAPTIC FEEDBACK

```
//Cause haptic vibration
platform.touch_sound();
```

CHECKING MESSAGE COUNTS

```
//Get number of new messages in inbox
var newInboxCount:Number = _root.newInboxCount;

//Get number of messages in inbox
var inboxCount:Number =  root.inboxCount;

//Get number of messages in drafts folder
var draftCount:Number = _root.draftCount;

//Get number of messages in outbox
var outboxCount:Number =  root.outboxCount;

//Get number of sent messages
var sentCount:Number = _root.sentCount;

//You should check whether all these counts are undefined before
//you display them, as the phone can take up 30 seconds at start-up
//to get the required data
```

HIDING THE NORMAL PHONE INTERFACE

```
//Hide the show/hide shortcut arrows
 root.idleChanger mc.unloadMovie();

//Hide the normal phone launcher
_root.mc_idle.idelLauncher.gotoAndStop(11);

//Hide the operator text
 root.cur main = "shortCut";
platform.setIdleCondition();

//Hide the shortCut launcher menu
_root.mc_idle.shortCut_mc._y = 401;
```

USING THE BUILT-IN CIRCULAR PRESS ANIMATION

```
//Show press animation
//Circle is centered at xCoord, yCoord
_root.show_pressAni(xCoord:Number, yCoord:Number, scalePercentage:Number);

//Hide press animation
_root.out_pressAni();
```

COPYRIGHT INFORMATION

THIS GUIDE

VERSION: 1.0.0
PUBLISHED: 27/08/2008

THE 'MYTHEME' EXAMPLE THEME