Tutorial

DEVELOPER WORLD THE FAST TRACK FROM MIND TO MARKET

June 2009

Project Capuchin Service API Generator

Flash Designer

Sony Ericsson

Preface

About this tutorial

This tutorial shows Flash Lite™ designers how to create a simple platform service application using a given MXP package.

Sony Ericsson Developer World

At www.sonyericsson.com/developer, developers find the latest technical documentation and development tools such as phone White papers, Developers guidelines for different technologies, Getting started tutorials, SDKs (Software Development Kits) and tool plugins. The Web site also features news articles, go-to-market advice, moderated discussion forums offering free technical support and a Wiki community sharing expertise and code examples.

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

This Tutorial is published by:

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

www.sonyericsson.com/

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

First edition (June 2009)
Publication number: 1228-9113.1

Typographical conventions

In this document code examples are written in Courier font:
`Calendar.addPIMChangeEventListener(this,PIMChangeHandler);`

Names of labels, buttons and menus are written in *italics*, for example, Select *File – Install extension…*

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

Trademarks and acknowledgements

Document history

Change history		
2009-06-01	Doc. no. 1228-9113.1	First version published on Developer World

Contents

Tutorial – Flash designer

The main purpose of this section is to show Flash Lite designers how to create a simple Platform Service application using a given MXP package. The application will be published as a Capuchin application in the end of the process.

The Flash designer role within the overall Platform Service development process is described in the Tutorial: Project Capuchin service API generator – Overview.

Prerequsites

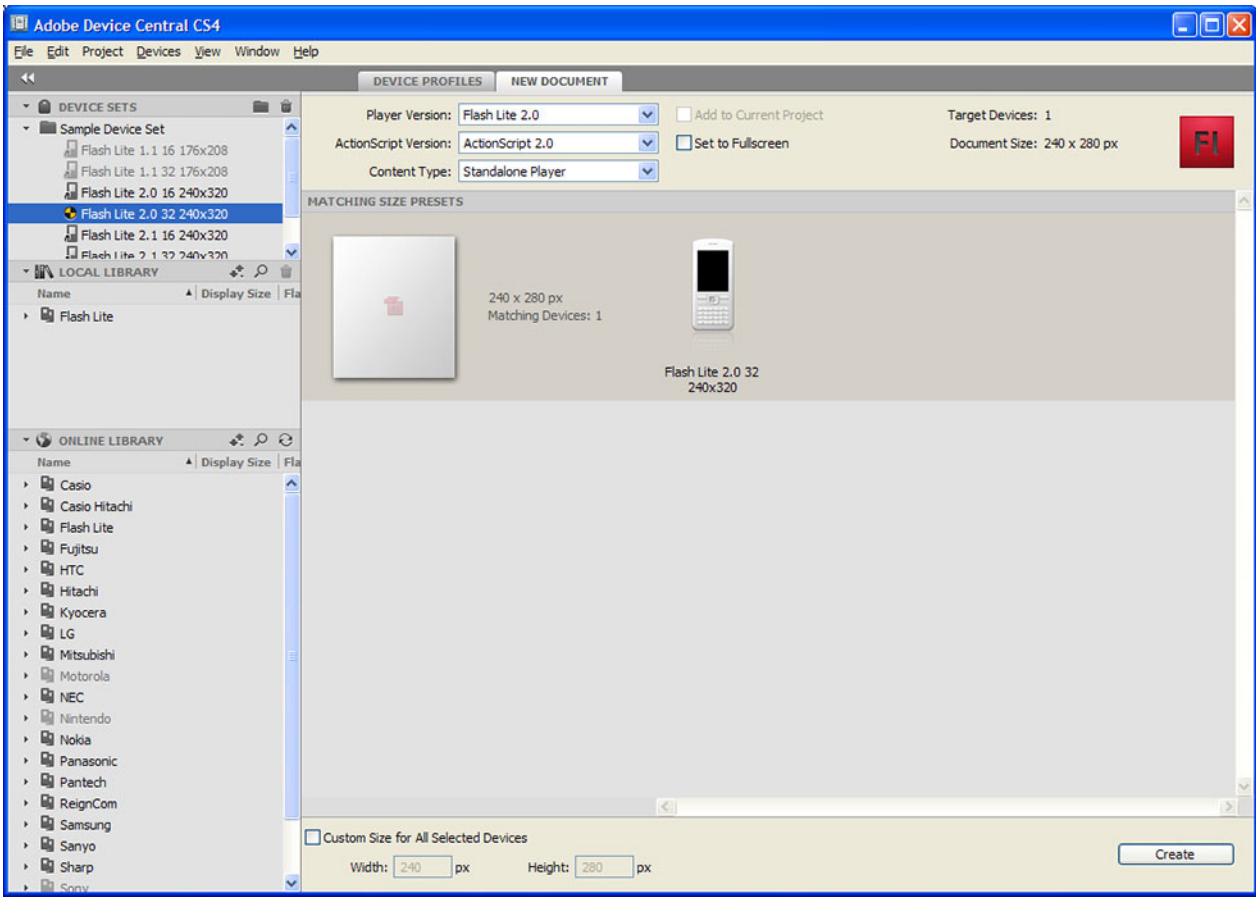
The applications listed below are necessary to use the Service API Generator:

- Adobe Flash (CS3 or higher version)
- Adobe Extension Manager (make sure that the Adobe Extension Manager is installed with the same language as Adobe Flash)
- Adobe Device Central
- Java SE Development Kit
- Capuchin Kit

Creating a Capuchin application

To create a new Capuchin application, open Adobe Device Central and select *File – New Document* or *New Document in Flash* from the menu and select “Flash File (Mobile)”. Select “Flash Lite 2.0” from the *Player Version* drop-down list and “ActionScript 2.0” from the *Action Script Version* drop down list.

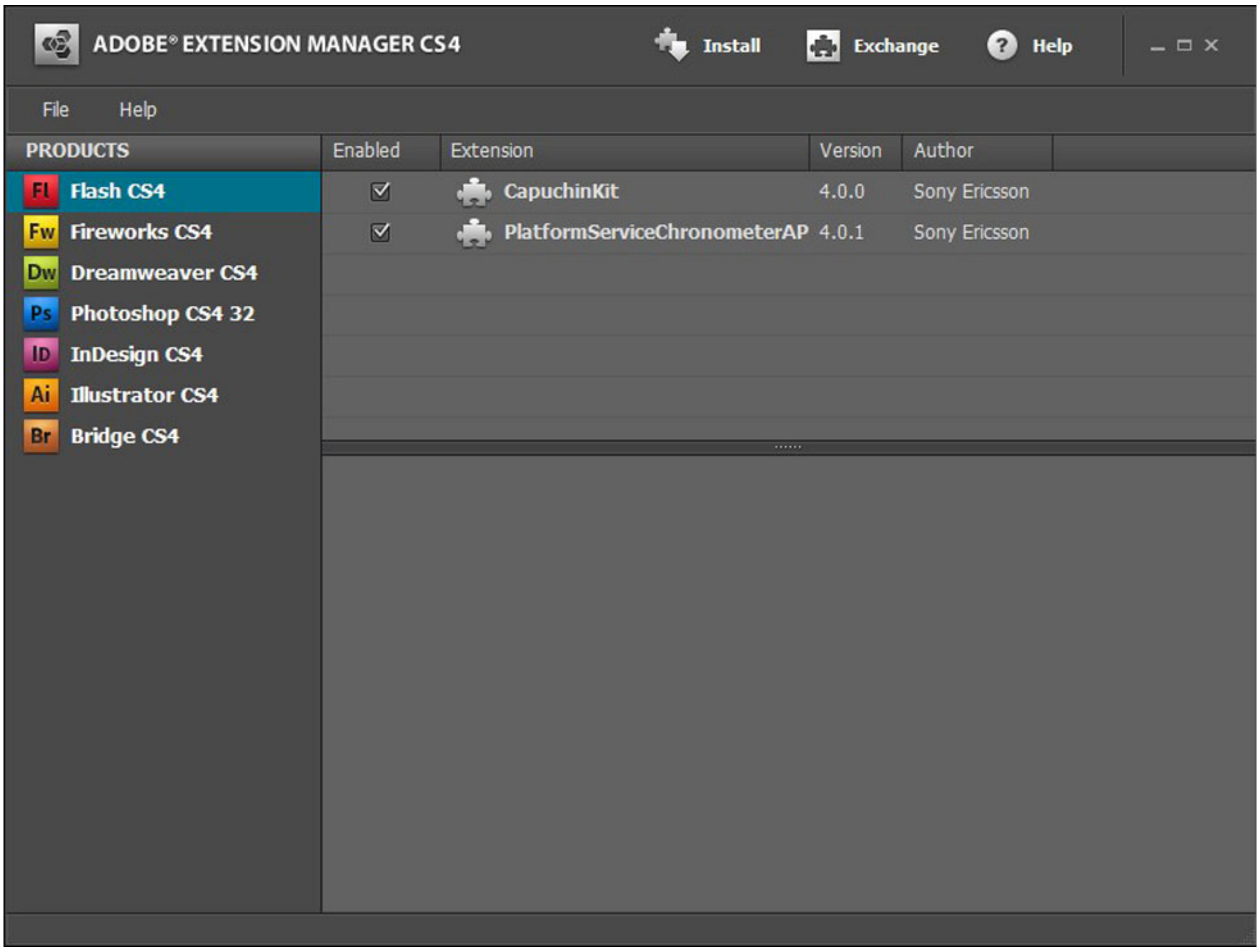
Click *Create* and Adobe Flash opens with a new Flash Lite document (.FLA file). Save the new FLA file in the folder where you want to store your Capuchin Project.



Player Version: Flash Lite 2.0
ActionScript Version: ActionScript 2.0
Content Type: Standalone Player
Add to Current Project
Set to Fullscreen
Target Devices: 1
Document Size: 240 x 280 px
Custom Size for All Selected Devices
Width: 240 px Height: 280 px

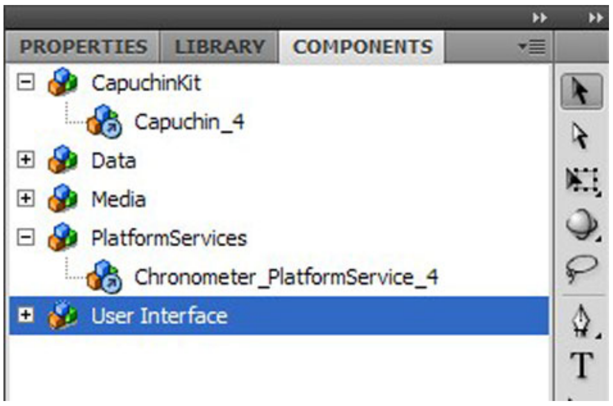
CapuchinKit and the desired service MXPs should be installed at this point to be used in the application development. To do that, double-click on the files and they will be installed. If you are using multiple versions of Flash (CS3, CS4…), be sure that the MXPs are installed in the corresponding Extension Manager.

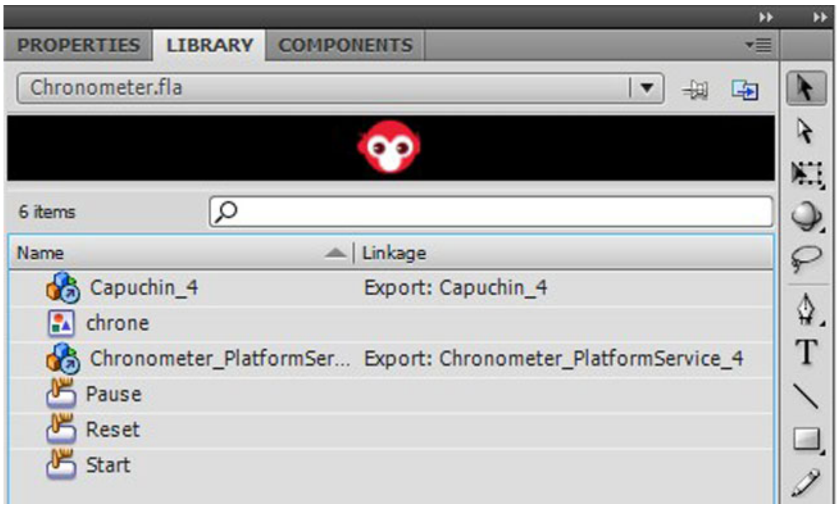
The image below shows Adobe Extension Manager CS4 with CapuchinKit and Service MXP installed.



Enabled	Extension	Version	Author
✓	CapuchinKit	4.0.0	Sony Ericsson
✓	PlatformServiceChronometerAP	4.0.1	Sony Ericsson

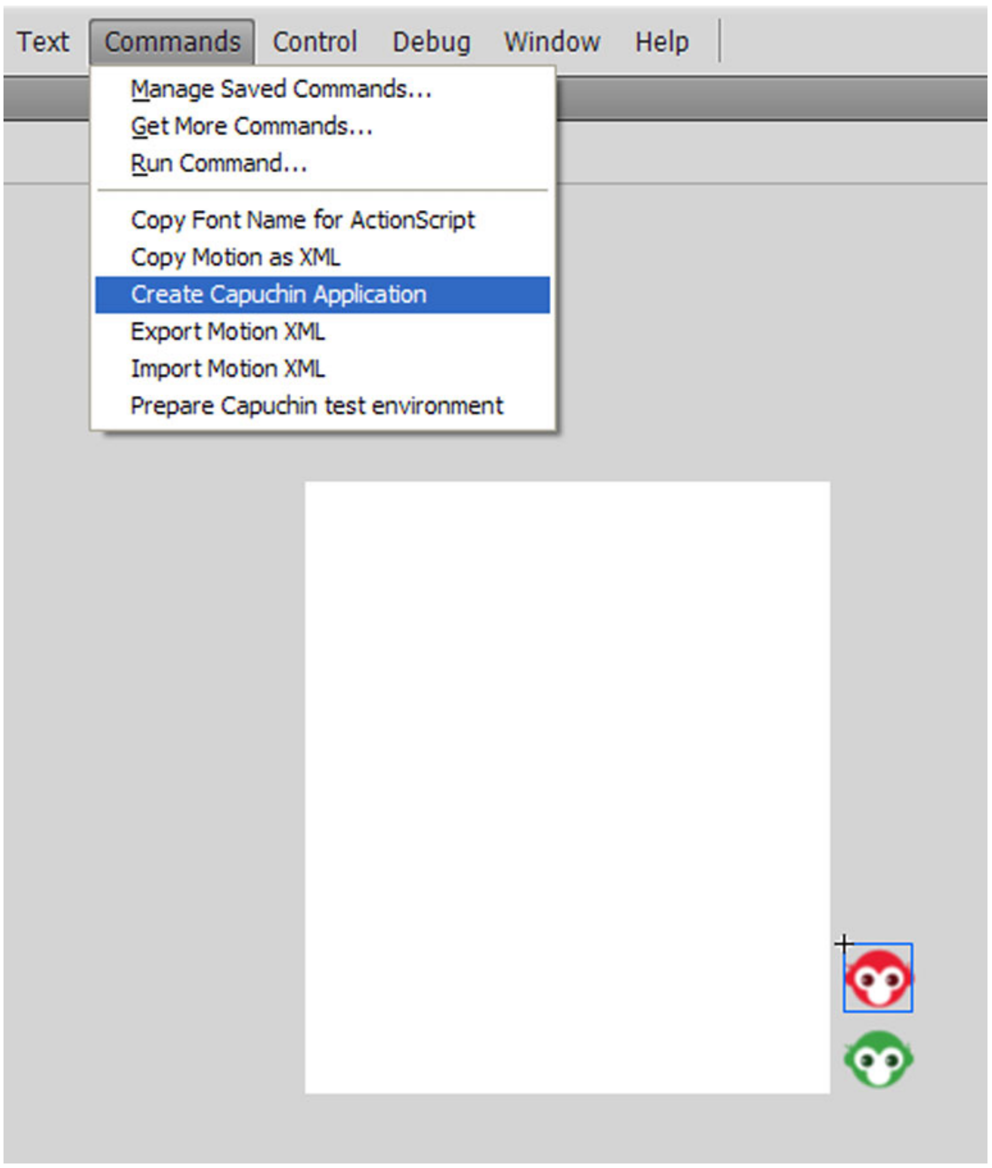
With the FLA document open in Adobe Flash, select *Window – Components* to open the “Components” pane, expand CapuchinKit and double-click on “Capuchin_x” (where x defines the framework version) in order to add it to the Library. Do the same with services, such as “PlatformService_y” (where y defines the Platform Service version). You can see that they have been added in the Library pane (select *Window – Library* from the menu).



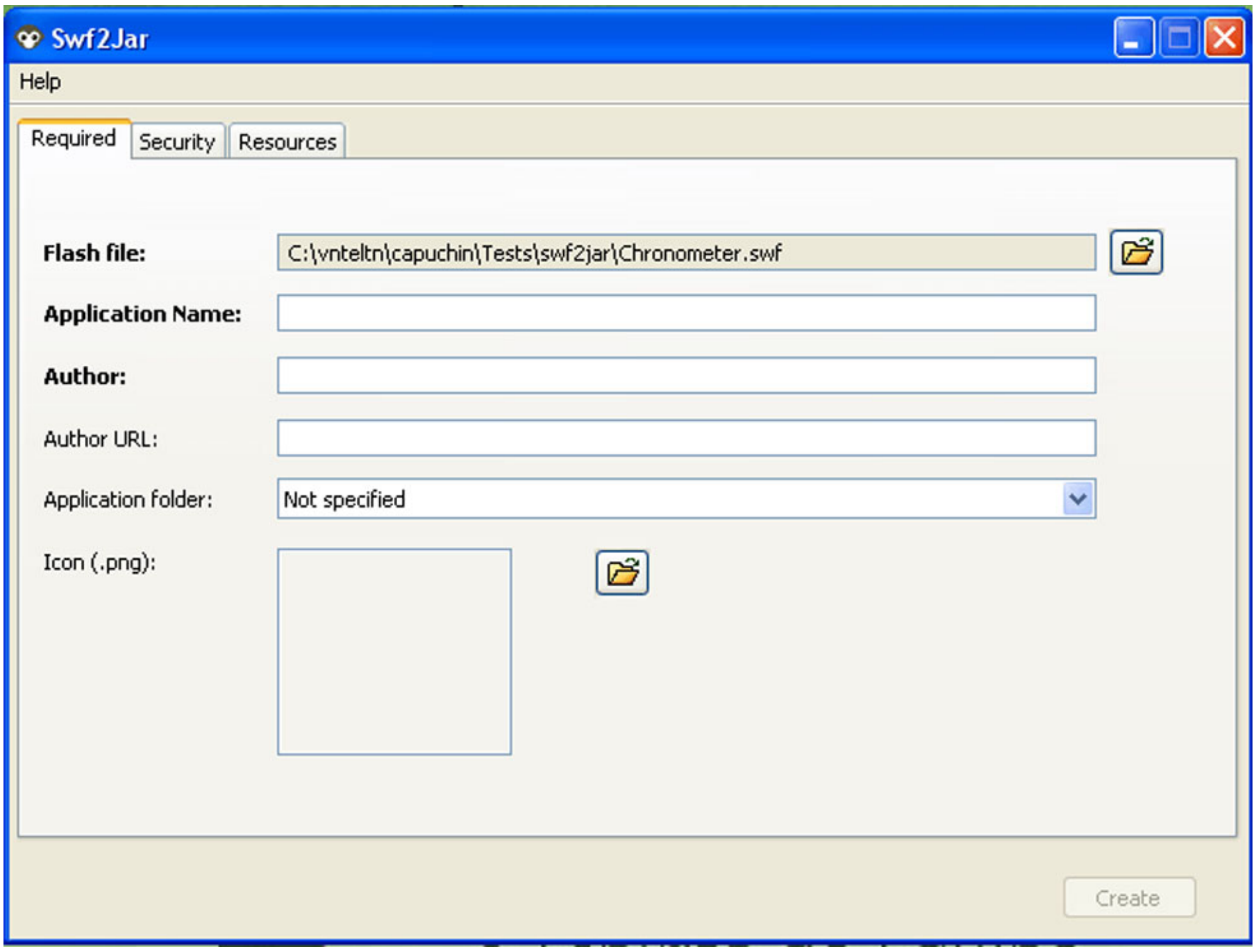


Name	Linkage
Capuchin_4	Export: Capuchin_4
chrone	
Chronometer_PlatformSer...	Export: Chronometer_PlatformService_4
Pause	
Reset	
Start	

From this moment on, any functionality provided by the Service MXP can be used by the Flash designer. Once the work is done, to publish the Capuchin application, go to Commands – Create Capuchin Application.



This opens the Swf2Jar application and the process from this point is very straightforward. Swf2Jar is an application for packaging a Flash file (.swf) into a Java MIDlet .jar file. The application supports setting Jad properties for the MIDlet, as well as signing the MIDlet. After packaging, the MIDlet can be transferred to a phone. When running the MIDlet on the phone, the packaged Flash content is automatically played. There is an internal manual in the Swf2Jar tool that can be opened by pressing the help button if more information is required.



Flash file: C:\vnteltn\capuchin\Tests\swf2jar\Chronometer.swf

Application Name:

Author:

Author URL:

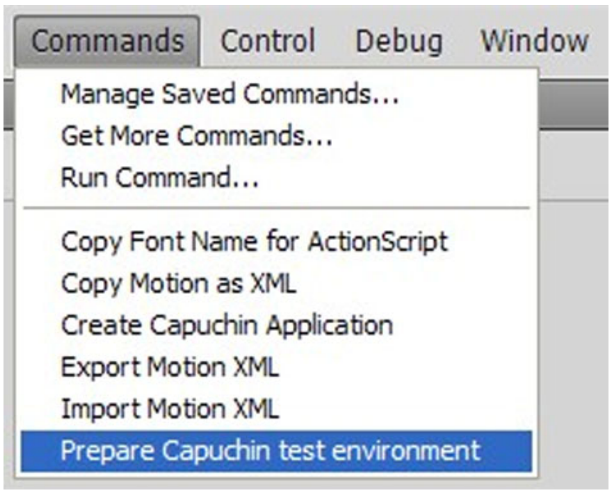
Application folder: Not specified

Icon (.png):

Testing the application

The emulator application requires an XML file that simulates response data from the Java side. The XML file contains static data that on a phone would be sent from Java to Flash.

To copy the adequate service XML file, select *Commands – Prepare Capuchin* test environment from the menu.



Keep in mind that every Platform Service has its own XML file with a name matching that of the API. In this example the file is "Chronometer.xml". The emulator recognizes which file to read by its name, so the XML file has to have a name matching that of a Platform Service, that is, files must not be renamed.

The test values stored in the XML file provide a mechanism for the Flash developer to test the application without having an actual phone to test on. Each method call in ActionScript returns dummy values defined in the XML file. Input parameters of called methods are not taken into consideration – returned values remains always the same – regardless of passed parameters.

To emulate, just press Ctrl + Enter and emulate as if it was a regular Flash file.

Resources

The following files can be used as an example of a Capuchin application. It contains the Flash files used to create a simple chronometer and the MXP of the service providing the chronometer functionality. In the “API designer" tutorial, the definition of the interfaces can be found. In the "Java developer" tutorial, the implementation of the Java stubs can be found.

The ActionScript code below uses phone keys with the Chronometer application.

Implemented ActionScript Flash file – Chronometer.fla

```
import com.sonyericsson.capuchin.chronometer.*;
import com.sonyericsson.capuchin.chronometer.datatypes.*;

//Define the status of the application
var isRunning = false;
pauseButton._visible=false;
resetButton._visible=false;

keyListener = new Object();
keyListener.onKeyDown = function()
{
     switch (Key.getCode())
     {
          case 'soft1': clickedSend(); break;
          case 'soft2': clickedReset(); break;
     }
}
Key.addListener(keyListener);

function clickedSend(){
     if (isRunning == false){
          startChronometer();
     }
     else{
          pauseChronometer();
     }

}

function clickedReset(){
     resetChronometer();
}

function startChronometer(){
      Chronometer.addTimerReceivedListener(this, onaddTimerReceivedListener);
      pauseButton._visible=true;
      resetButton._visible=false;
      startButton._visible= false;
      isRunning=true;
}

function resetChronometer(){
     Chronometer.resetChronometer(this, onResetChronometer);
}

function pauseChronometer(){
     Chronometer.removeTimerReceivedListener(this,
onremoveTimerReceivedListener);
}

function onStartChronometer(owner:Object, status:Boolean)
{
     if (status==true){
          pauseButton._visible=true;
          resetButton._visible=false;
          startButton._visible= false;
          isRunning=true;
     }
}

function onResetChronometer(owner:Object, status:Boolean)
{
     if (status==true){
          //resets the chronometer
          clock = "00:00:00";
          resetButton._visible=false;
          isRunning=false;
     }
}

function onaddTimerReceivedListener(owner:Object, status:Boolean,
chronTime:ChronometerTimer)
{
    clock = fillZeros(chronTime.getMinute(),2) +
":"+fillZeros(chronTime.getSecond(), 2)+":"+
fillZeros(chronTime.getMiliseconds(),2) ;
}

function onremoveTimerReceivedListener(owner:Object, status:Boolean)
{
     if (status==true){
          pauseButton._visible=false;
          startButton._visible= true;
          resetButton._visible=true;
          isRunning=false;
     }
}
function fillZeros (num, places) {

  // Convert the number to a string and eliminates the '.'
  var filledVal = (num.toString()).split(".")[0];

  // Get the length of the string
  var len = filledVal.length;

  // Use a for statement to add the necessary number of characters.
  for (var i = 0; i < (places - leaan); i++) {
  // If trailing is true, append the zeros; otherwise, prepend them.
      filledVal = "0" + filledVal;
  }
  // Return the string
  return filledVal;
};
```