Archived:Flash Lite API Bridge Interface

Archived: This article is **archived** because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

We do not recommend Flash Lite development on current Nokia devices, and all Flash Lite articles on this wiki have been archived. Flash Lite has been removed from all Nokia Asha and recent Series 40 devices and has limited support on Symbian. Specific information for Nokia Belle is available in Flash Lite on Nokia Browser for Symbian. Specific information for OLD Series 40 and Symbian devices is available in the Flash Lite Developers Library.

13 Dec 2009

Purpose of this document

This document explains how to use the services provided by the APIBridge component from Flash Lite applications on Nokia devices. The document describes the API used to access the services provided by the APIBridge and how to set up your Flash Lite applications to make use of the APIBridge.

APIBridge High-Level Architecture

The APIBridge is a Symbian server that exposes an HTTP interface for communication between the server and its clients. The APIBridge can be used by Flash Lite ActionScript code by making loadVars calls to the local host port that the APIBridge listens to. The following architecture diagram explains the different parts of the system:

Here is a description of each of the components:

Component	Description
Access Layer ActionScript API	The ActionScript API provides Flash Lite applications with a function-based interface to the services of the APIBridge. It performs the following activities: ▪ Internally generates the HTTP requests to the APIBridge from the function calls, abstracting this complexity from the application developer; This layer is represented by the apibridge.as file, which can be included and used in any Flash Lite application that intends to use the APIBridge.
Framework Layer APIBridge Server	The APIBridge Server is responsible for authenticating clients, receiving requests, routing requests to the appropriate APIBridge plug-in for execution, and returning the results.
ECOM Plug-	

ECOM Plug-ins Layer Plug-ins	Plug-ins are responsible for analysing the parameters in the request, calling the appropriate Symbian API's to execute it, and creating the HTTP response.

An Important Note for Flash Lite Content Embedded in a Web Runtime Widget

A popular technique for building rich mobile Internet mobile applications for Nokia devices is to combine Flash Lite content in a web runtime widget. More and more developers are finding that this architecture gives them maximum flexibility in developing their applications for Nokia. Web runtime lets them use the ease of HTML, JavaScript and CSS to build the majority of their application UI. Flash Lite allows them to embed multimedia features such as audio and video, advanced animations, and the like into their applications easily. For applications of this type, it might be simpler to access the API Bridge from the JavaScript side of the application, and pass data from JavaScript to ActionScript by means of ActionScript external interfaces. This leaves the Flash Lite code API Bridge-free, so that the Flash Lite components of the application can be built without requiring an understanding of how to program to the API Bridge from ActionScript.

APIBridge ActionScript API

Flash Lite access the API Bridge via an ActionScript API. This API is modeled after the S60 Platform Services interface model. Except in a few cases, such as the File Upload API, the programming model is similar to the S60 Platform Services model.

An application creates a service by specifying the service name and the desired service interface, then calls methods implemented by that service to perform the actions desired. This chapter describes the ActionScript API available for Flash Lite applications. The function API described here is contained in the ApiBridge.as file that developers will need to include in their Flash Lite.

The Flash Lite API Bridge interfaces are implemented as ActionScript classes that need to be downloaded and imported into your Flash Lite applications. The class package, as well as a set of Flash Lite sample applications using the API Bridge, can be downloaded from this link:

File:APIBridge FlashLiteCode.zip

This file contains the definition of the APIBridge class, through which applications create instances of services. A service is similar to an S60 Platform Services service object. Each service defines one or more service-specific methods that expose the functionality of that particular service. The general usage model of the ActionScript API Bridge interface is illustrated in the code below:

General ActionScript API Bridge Usage Model

```
import si.apibridge.*;

//Create a new instance of APIBridge, with callback error function
var bridge:APIBridge = new APIBridge(onBridgeError);

//Create a new service
var myService = bridge.Service("ServiceName", "InterfaceName");

//APIBridge error callback function
function onBridgeError (outParam:Object) {
  trace("APIBridge error " + outParam.ErrorCode + " " + outParam.ErrorMessage);
}

var param:Object = new Object();
param.property1 = value1;
param.property2 = value2;
...
myService.method(param,onResult);
//Callback called when service method returns a response
function onResult(transactionID:Number, eventID:String, outParam:Object) {
   ...
```

```
}
```

File Upload

Description

This function allows for the upload of a file to a web server using a multipart/form-data POST request. It also supports the addition of accompanying parameters in the same POST request. Note that this is a case where the Flash Lite API Bridge interface is NOT modeled after S60 Platform Services.

```
UploadFile ( inParam:Object, onReceive:Function):Void
```

Parameters

Parameter	Description
inParam	Structure containing name / value pairs with the parameters needed to upload a file. These are detailed in the inParam Argument Properties table below.
onReceive	Callback function invoked when the file upload operation completes or reports an error.

inParam Argument Properties

Property Name	Description
VarName	Name of the variable that will contain the binary data of the uploaded file. This name will be used by the server side script.
FileName	Local path to the file to be uploaded.
Url	URL of the server side upload script.

Usage

```
import si.apibridge.*;

var bridge:APIBridge = new APIBridge(onBridgeError);

function fileUpload()
{
  txtOutput.text = "Uploading file " + filenames[1];
  var inParams:Object = new Object();

  inParams.VarName = "uploadfilename";
  inParams.FileName = filenames[1].toString();
  inParams.Url = "http://192.168.1.100:9090/uploader/upload.php"; //Set to your upload
URL

  bridge.UploadFile(inParams,onFileUpload);
}

function onFileUpload (transactionID:Number, eventID:String, outParam:Object) {
  if (outParam.ErrorCode != 0)
  {
    txtOutput.text = "Error occured while uploading. " + outParam.ErrorCode;
    return;
  }
  else
  {
```

```
    var outList = outParam.ReturnValue;
    trace(outList);
  }
}
```

New File Service

Description

This function allows Flash Lite applications to embed native applications in order to capture images, videos, and audio. This interface is implemented via the NewFileService class defined in the NewFileService.as ActionScript file. The TakePhoto method can be used to capture all of these media types. This method launches the native camera (for image and video capture) or audio recorder on the device. After the media file is captured by the user, the onReceive callback is called. Applications typically use this interface in combination with the file upload interface described above to capture and upload media files to web services.

```
TakePhoto ( inParam:Object, onReceive:Function):Void
```

Parameters

Parameter	Description
inParam	Structure containing name / value pairs with the parameters needed to capture media. These are detailed in the inParam Argument Properties table below.
onReceive	Callback function invoked when the method completes.

inParam Argument Properties

Property Name	Description
NewFileType	Specifies the type of file being captured. Values incluude "Image", "Audio", and "Video".

Usage

This sample shows how to capture an image by lauching the native camera. To capture audio of video, the inParams.NewFileType property would be changed to "Audio" or "Video".

```
//Capturing a photo
//Import APIBridge library
import si.apibridge.*;

//Create a new instance of APIBridge, with callback error function
var bridge:APIBridge = new APIBridge(onBridgeError);
```

```
//Create a new newfile service
var fileService = bridge.Service("Service.NewFileService", "IDataSource");

var inParams:Object = new Object();
```

```
inParams.NewFileType =  "Image";
fileService.TakePhoto(inParams,onPhoto);

function onPhoto (transactionID:Number, eventID:String, outParam:Object) {
  ...
}
```

Call Log Service

Description

This interface allows Flash Lite applications to access the device call log:

```
loggingService.GetList (inParam:Object, onReceive:Function):Void
```

Parameters

Parameter	Description
inParam	Structure containing name / value pairs with the parameters needed to read the call log. Currently not used. Parameter is provided for future additions to the service.
onReceive	Callback function invoked when the call log operation completed or reports an error.

inParam Argument Properties

Property Name	Description
Currently Not Used	

Usage

```
import si.apibridge.*;

var bridge:APIBridge = new APIBridge(onBridgeError);

//Create a new logging service
var logging = bridge.Service("Service.Logging", "IDataSource");

//Retrieve calllog through APIBridge
function getCalllLog()
{
  txtOutput.text  = "Retrieving phone numbers ...";
  var filter = {EventType:0};
  var inParams = {Type:"Log", Filter:filter};
  logging.GetList(inParams, onCallLog);
}

//Callback for displaying retrieved call log information
function onCallLog (transactionID:Number, eventID:String, outParam:Object) {

  txtOutput.text  = "Results: " + outParam.ReturnValue.length + "\n" ;

  if (outParam.ErrorCode != 0)
  {
    txtOutput.text = "Error occured while retrieving values.";
    return;
  }
  else
  {
    var outList = outParam.ReturnValue;
    var outputEntry = null;
    do {
      outputEntry = outList.next();

      if (null != outputEntry)
```

```
      {
        txtOutput.text += outputEntry.PhoneNumber
        txtOutput.text += "\n";
      }
      else
      {
        break;
      }
    } while (true);
  }
}
```

Location Service

Description

This interface allows Flash Lite applications to access device location / GPS information:

```
locationService().GetLocation (inParam:Object, onReceive:Function):Void
```

Parameters

Parameter	Description
inParam	Structure containing name / value pairs with the parameters needed to read GPS information. Currently not used. Parameter is provided for future additions to the service. Presently, the GetLocation method works like the Platform Services location interface GetLocation method with Basic as the location information argument. In other words, the API Bridge GetLocation method currently only returns longitude, latitude and altidude information.
onReceive	Callback function invoked when the call log operation completed or reports an error.

inParam Argument Properties

Property Name	Description
Currently Not Used	

Usage

```
//Import APIBridge library
import si.apibridge.*;

//Create a new instance of APIBridge, with callback error function
var bridge:APIBridge = new APIBridge(onBridgeError);

//Create a new location service
var location = bridge.Service("Service.Location", "ILocation");

var inParams = null;
location.GetLocation(inParams,onLocation);

function onLocation (transactionID:Number, eventID:String, outParam:Object) {

  if (outParam.ErrorCode != 0)
  {
    txtOutput.text = "Error occured while retrieving values.";
    txtOutput.text += "\nErrorCode: " + outParam.ErrorCode +
      " - ErrorMessage: " + outParam.ErrorMessage;
```

```
    return;
  }
  else
  {
    var outList = outParam.ReturnValue;
    txtOutput.text += "Langitude: " + outList.Longitude;
    txtOutput.text += "\nLatitude: "  + outList.Latitude;
    txtOutput.text += "\nAltitude: "  + outList.Altitude;
  }
```

}

Media Management Service

Description

This function allows Flash Lite applications to access and enumerate the device media files:

```
mediaManagementnService().GetList (inParam:Object, onReceive:Function):Void
```

Parameters

Parameter	Description
inParam	Structure containing name / value pairs with the parameters needed to enumerate media files. These are detailed in the inParam Argument Properties table below.
onReceive	Callback function invoked when the method returns or encounters an error.

inParam Argument Properties

Property Name		Description
Filter		
Parameter	**Possible Values**	
FileType	"Music" "Sound" "Image" "Video" "StreamingURL"	

Usage

Here is some sample code that retreives all of the music files from the device

```
import si.apibridge.*;

var bridge:APIBridge = new APIBridge(onBridgeError);

//Create a new media service
var media = bridge.Service("Service.MediaManagement", "IDataSource");

//Callback for displaying retrieved values
function onMedia (transactionID:Number, eventID:String, outParam:Object) {

  if (outParam.ErrorCode == 1)
  {
    txtOutput.text = "Error occured while retrieving values.";
  }
  else
  {
    txtOutput.text  = "Results: " + outParam.ReturnValue.length + "\n" ;
```

```
    var outList = outParam.ReturnValue;
    var outputEntry = null;
    do {
      outputEntry = outList.next();

      if (null != outputEntry)
      {
        txtOutput.text += outputEntry.FileName;
        txtOutput.text += "\n";
        trace(logId);
      }
      else
      {
        break;
      }
    } while (true);
  }
}

function getMedia()
{
  var filter = {FileType:"Music"};
  var param:Object = new Object();
  param.Filter = filter;
  media.GetList(param,onMedia);
}
```

How to Package A Flash Lite Application that uses the APIBridge

For this step, developers should have at least limited knowledge of Symbian programming as well as the build system and compiler in order to create the installation package for their widget. The APIBridge is delivered with a template source code of a Widget Installer executable. This program will take care of installing the .wgz file after all of the required Symbian native components are installed.

Applications that use the APIBridge will need to be packaged as Symbian Installable Files (.SIS). This is required in order to install the APIBridge framework and its plug-ins in the device (in case they weren't already there). Your Flash Lite content in your application will have been published by Adobe CS4 as a .SWF file. This file needs to be packaged as part of this SIS file in some way. As a Flash Lite developer, you have two options for this.

The first is to package your Flash Lite .SWF in a SIS file of its own. The Nokia Developer Online Packager was the tool of choice for packaging Flash Lite content in SIS format: Nokia Developer Online Packager

However, until this online tool is up and running again, the other way to package a Flash Lite application in a SIS file is to create a stub S60 application for this purpose, as described in this article: Packaging Flash Lite Content in an S60 Stub Application

Building an S60 stub application can be a bit tricky. Fortunately, there is an easier way to package your application. This is to embed the Flash Lite content in a Web Runtime widget, and then package that widget, along with the API Bridge components, in a SIS file.

To embed Flash Lite content in a widget, use the technique described here: How to Package Flash Lite Content in a Widget

Then, follow the steps outlined below to create the installation package containing your new widget and the API Bridge components.

Creating the WgzInstaller for the widget

The template sources for creating the WgzInstaller can be found at

<APIBridge Directory>\WgzInstaller

and the project can be imported to Carbide by using the *bld.inf* in the *group* directory.

To compile this utility, developers must install the SWInstaller Plug-in to their S60 SDK. For more information go to:

SW Installer Launcher API

Follow these steps to create the WgzInstaller:

1. Open the *WgzInstaller\group\WgzInstaller.mmp* file.

- Replace the UID2 with you own UID.

2. Open the *WgzInstaller\src\WgzInstaller.cpp* file.

- Change the value of the *KWidgetInstallerFileName* Literal with the name of your .wgz file.

3. Compile the project to generate the customised *WgzInstaller.exe.*

Creating the .sis file for the widget

Once the .exe has been generated, it is time to create the installation package. Follow these steps:

1. Copy the .wgz and the *ApiBridge_vx_x_x.sis* to the *WgzInstaller/content* directory.

2. Modify the WgzInstaller*/sis/WgzInstaller_template.pkg*:

- Application name
- Installation UID
- Vendor name
- APIBridge version if not correct
- .wgz file name

3. Generate the .sis file using the modified .pkg file.

4. Sign the .sis file.

The signed .sis file is ready to be installed on the device.