Using Single Sign-On (SSO)

Published 2014-10-28 | (Compatible with SDK 2.5,3.5,4.5,5.0,5.1 and 2011,2012,2013,2014 models)

> Example of using Single Sign-On (SSO) feature for log in to the Google Account

Contents

- Prerequisites
- Development environment
- Application Design
- Source Files
- Class description
- SSO Login
  - Creating the Basic Application
  - Creating SSO Login
  - Passing Login Information to the Application

The Single Sign-On (SSO) functionality is provided by the Application Manager to overcome the inconvenience of inputting information through remote controller keys. The Application Manager saves the user account information and sends it to the respective applications. This saves the user from re-entering their account information repetitively for every application. The account information is encrypted and saved in a database.

The steps involved in the SSO authentication process are:

1. Registering in the Service Site Area
2. Verifying of the entered information with the relevant site
3. Retrieving the login information of the App from the Application Manager when the App is launched after enabling the SSO feature

The sample application being created in this tutorial uses the SSO feature to login into the Gmail services and display the login status. If the authentication details are not available in the SSO database, it displays the status as failed to login. This application also demonstrates the optional notifications such as “Invalid User ID or Password” and displays various status messages for the available SSO options.

Prerequisites

To create applications that run on a TV screen, you need:

Samsung TV connected to the Internet
SDK or a text editor for creating HTML, JavaScript and CSS files (using Samsung Smart TV SDK is recommended)

> Important
>
> The SSO feature is not supported in the SDK Emulator. In order to test this feature you need to verify it on a real device.

Development environment

Use Samsung Smart TV SDK to create the application. You can also use the emulator provided with the SDK to debug and

test the application before uploading it in your TV. Later, you can run the application on a TV; see Testing Your Application on a TV. Note that applications may perform better on the TV than on the emulator.

In this tutorial, index.html file uses relative paths to include Common Modules for the code to run on the Samsung Smart TV emulator. It will work when first testing on a real Samsung Smart TV device. However, when an application is launched on the live Samsung Smart TV service, it should access the common modules from their correct location, using an absolute path beginning with $MANAGER_WIDGET.

For example, instead of:

```
<script type="text/javascript" src="Common/API/Plugin.js"></script>
```

Use:

```
<script type="text/javascript" src="$MANAGER_WIDGET/Common/API/Plugin.js"></script>
```

Application Design

The modules that form part of the application are illustrated in the figure below. The ID and password are encrypted and stored in secured storage. The Application Manager is responsible for sending the ID and password for authentication. The Authentication module is responsible for the authentication using the ID and password and logs on to the server.

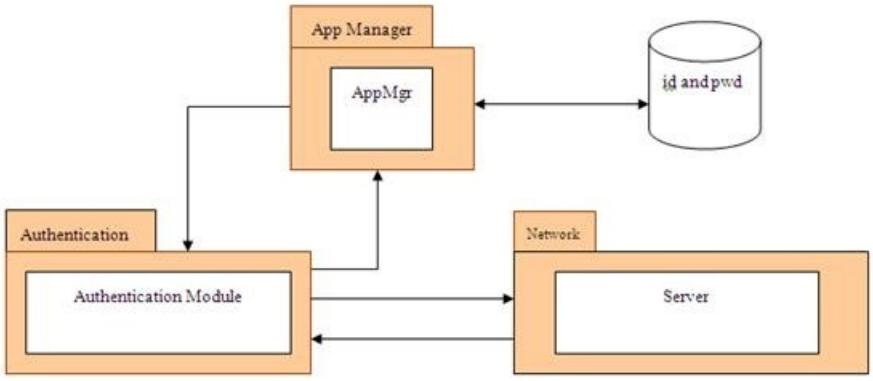

Figure: Application design

Source Files

> Note
>
> The files needed for the sample application are **here**. The application must contain a JavaScript file (Main/Main.js), a HTML file (index.html) and a CSS file (CSS/Main.css).

The directory structure for the tutorial application:

File/Directory	Description
Common	Contains Common modules provided by the Application Manager.
Main	Contains the JavaScript files.
CSS	Contains CSS files.
Resource	Contains the relevant image files.

Class description

The participating classes for this application and their responsibilities are as follows:

Class	Description
Main	Parses the user ID and password sent by the Application Manager.
Authorization	Registers the login information with the Application Manager.

SSO Login

Creating the Basic Application

1. Start the SDK for Samsung TV applications. Create a new application using the following config.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<App>
  <previewjs>previewSSO</previewjs>
  <cpname>SSO-Google</cpname>
  <login>y</login>
  <cplogo></cplogo>
  <prelcon></prelcon>
  <cpauthjs>Authorization</cpauthjs>
  <ThumbIcon>Resource/image/SSO2.png</ThumbIcon>
  <BigThumbIcon>Resource/image/SSO3.png</BigThumbIcon>
  <ListIcon></ListIcon>
  <BigListIcon></BigListIcon>
  <category>Information</category>
  <autoUpdate>n</autoUpdate>
  <ver>0.100</ver>
  <mgrver>1.000</mgrver>
  <fullApp>y</fullApp>
  <srcctl>y</srcctl>
  <ticker>n</ticker>
  <childlock>n</childlock>
  <audiomute>n</audiomute>
  <videomute>n</videomute>
  <dcont>y</dcont>
  <type>user</type>
  <Appname>Single Sign On</Appname>
  <description>SSO App</description>
  <width>960</width>
  <height>540</height>
  <author>
    <name>Samsung SDS</name>
    <email></email>
    <link>http://acme-App.example.com</link>
    <organization>Acme Examples, Inc.</organization>
  </author>
</App>
```

The application uses the following settings:

<fullApp>y</fullApp>
Set this as ‘y’. This makes the application run in full screen mode. This affects what keys are registered by default.
<type>user</type>
This enables the user application feature for testing on a real TV set. This tag has no effect on the emulator.
<cpauthjs>Authorization</cpauthjs>
This is the name of the Authorization file. The name of this file should be unique for each application.
<cpname>SSO-Google</cpname>
This is the name by which the necessary service will appear in the Application Manager’s service site area for registration.
<login>y</login>
Set this as ‘y’. This tag defines whether sign on service is required.

1. Add main.js file with the following code in the Main folder:

```
var widgetAPI = new Common.API.Widget(),
    tvKey = new Common.API.TVKeyValue(),
    Main = {};

Main.onLoad = function () {
    alert("Main.onLoad()");
    widgetAPI.sendReadyEvent();
}

Main.onUnload = function() {
    alert("Main.onUnload()");
}
```

   If you have unzipped the provided files, the application already has an HTML index page.
2. Start the SDK emulator. If you see the 'alert() : Main.onLoad()' message in the log manager, it means you have successfully created the application. You should be able to see the provided layout on the screen. You should also be able to test it on a TV .

Creating SSO Login

The basic requirement of SSO is that a TV account is created. This has to be done before creating the Authorization.js file.

The purpose of Authorization.js file is to verify the login information supplied by the user with the service site. To create the Authorization.js file:

1. Declare an object which has the same name as the one entered in the <cpauthjs> tag of the config.xml file.
2. Add a checkAccount method to this object. This method will be responsible for validating the account information and delivering the result by means of a callback function already passed to the method. The code for Authorization.js file is as follows:

```
var Authorization = {};

Authorization.checkAccount = function (id, pw, fnCallback) {
    alert("id=====" + id);
    alert("pw====" + pw);
}
```

The alerts return the relevant ID and password passed to the Authorization.js file by the Application Manager. This data is passed to the Application Manager when the user enters his/her login information in the service site area for the first time.

The account information must be validated. The method for this varies with the service site. This tutorial uses Google services to demonstrate SSO using the client login feature of Google. The application should programmatically log into Google accounts using the user login ID and password provided in the SSO feature of the Application Manager.

For this, an understanding of Google client login feature is necessary. The login feature works as follows:

1. For logging on to Google services, an application makes a request to Google authorization service.
2. If authorization is not successful, a failure response (HTTP 403) is returned.
3. If it is successful, a success response is returned (HTTP 200) with the required authorization code.
4. The authorization code is referenced in all subsequent requests made to the Google service for this account.

To validate the account information:

1. Make an HTTP post request to the authorization service. This is achieved through XHR communications. The code for this is as follows:

```
var Authorization = {},
    bURL = "https://www.google.com/accounts/ClientLogin",
```

```
    gmailData = null,
    xhttpObject = null;

Authorization.checkAccount = function (id, pw, fnCallback) {
    gmailData = "accountType = HOSTED_OR_GOOGLE&Email=" + id + "&Passwd=" + pw + "&service=mail&sour
ce=";
    alert("bURL" + bURL);
    parsebrowserData (fnCallback);
}

function getResponse (fnCallback) {
    if (isValidAccount) {
        fnCallback("TRUE");
    } else {
        fnCallback("FALSE");
    }
}

function parsebrowserData (fnCallback) {
    if (window.XMLHttpRequest) {
        xhttpObject = new XMLHttpRequest();
    }
    if (xhttpObject != null) {
        xhttpObject.onreadystatechange = function () {
            if (xhttpObject.readyState == 4) {
                if (xhttpObject.status == 200) {
                    receivebrowserXHRResponse(fnCallback);
                }
                if (xhttpObject.status == 403) {
                    isValidAccount = 0;
                    getResponse(fnCallback);
                }
            }
        }
        xhttpObject.open("POST", bURL, true);
        xhttpObject.setRequestHeader("Content-Type", "application/x-www-form- urlencoded; charset=UTF-8");
        xhttpObject.send(gmailData);
    }
}

function receivebrowserXHRResponse (fnCallback) {
    var AuthData = bxhttp.responseText,
        startIndex = AuthData.indexOf("Auth"),
        endIndex = AuthData.indexOf(""),
        subStringElement = AuthData.substring(startIndex+5);

    if (subStringElement) {
        isValidAccount = 1;
    }
    getResponse(fnCallback);
}
```

For more information on Google authorization, see **Authentication and Authorization for Google APIs**.

2. This module is responsible for sending the request to the service site for login. For this purpose, create an XMLHttpRequest object named bxhttp using the following code:

```
bxhttp = new XMLHttpRequest();
```

Using HTTP POST request, send the user ID and password to the service site for authentication along with service you wish to log into. The code for this is as follows:

```
if (xhttpObject != null) {
  xhttpObject.onreadystatechange = function () {
    if (xhttpObject.readyState == 4) {
      if (xhttpObject.status == 200) {
        receivebrowserXHRResponse(fnCallback);
      }
      if(xhttpObject.status == 403) {
        isValidAccount = 0;
        getResponse(fnCallback);
      }
    }
  }
  xhttpObject.open("POST", bURL, true);
  xhttpObject.setRequestHeader("Content-Type", "application/x-www-form-urlencoded; charset=UTF-8");
  xhttpObject.send(gmailData);
}
```

If the request is successful, a server message comprising number ‘200’ along with the authorization token number is returned by the service site.

3. Next, parse the token number:

```
var AuthData = xhttpObject.responseText,
  startIndex = AuthData.indexOf("Auth"),
  endIndex = AuthData.indexOf(""),
  subStringElement = AuthData.substring(startIndex + 5);
```

4. Depending on the response, send ‘true’ or ‘false’ with the callback function:

```
function getResponse(fnCallback) {
  if (isValidAccount) {
    fnCallback("TRUE");
  } else {
    fnCallback("FALSE");
  }
}
```

Passing Login Information to the Application

The account information is encrypted and saved by the Application Manager after it has been successfully validated. This information appended with the URL of the application’s index page, is passed to the relevant application by the Application Manager. To retrieve this login information:

1. Add the following code to the Main.js module:

```
Main.ParseLoginInfo = function() {
    var params = window.location.search,
    i,
    data;
    if (params.charAt(0) == "?") {
        params = params.slice(1, params.length);
    }
    params = params.split("&");
    for (i = 0; i < params.length; i++) {
        data = params[i].split("=");
        if (data.length == 2) {
            switch (data[0]) {
                case "id":
                    this.picasaID = data[1];
                    break;
                case "pw":
                    this.picasaPW = data[1];
                    break;
                default:
                    break;
            }
        }
    }
}
```

2. The ID and password from the Application Manager is received using:

```
var params = window.location.search;
```

3. Next, parse the relevant details required from the URL string so obtained:

```
if (params.charAt(0) == "?") {
  vars = params.slice(1, vars.length);
}
```

In the case of SSO, the ID and password values passed to the encodeURIComponent() function come from the Application Manager. The value should be used in the application only after using decodeURIComponent(). The reason for using this method is to prevent the problem related to the special character '#', that is a keyword already used in the browser.

This login information is used by the application to login into the required service site programmatically. This part of the code is similar to the one used by the Application Manager in the Authorization.js file. The application receives an authorization code once the login is successful. This is the reference that the application needs to send for retrieving account information from the service site.

The functions used in the main.js and Authorization.js files are provided by the SSO common module.