Sybase Mobiliser Platform

Developer Guide: Smartphone Mobiliser Applications

Version 5.1

Document ID: DC01866-01-0510-01

Last Revised: October 2012

Table of Contents

1. Introduction to Developer Guide for Mobile Smartphone 1
2. Designing a Smartphone Application 2
   Mobiliser Smartphone Architecture 2
   The Reference Smartphone Application 3
3. Developing a Smartphone Application 6
   Environment Setup 6
   Smartphone Code Layers 6
   Main Functions of the Application 7
4. Customizing a Smartphone Application 8
   Custom Look and Feel 8
   Custom Functionality 9
   Custom Mobiliser Transactions 10
   Different UI Packages 11
5. Code Generation/Building 11
6. Deploying the Application to Mobiliser Platform 11
7. Provisioning the Application to the Device 12
8. Debugging 12
9. Testing 12
10. Localizing 12
11. Securing the Application 13
12. Authentication/Registration 13
    Registration 13
    Authentication 14
13. Running Application on the Device 14
    Setting the Server Information 14
14. API Reference 15
    Class MobiliserClient 15

Table of Figures

Figure 1. Smartphone Mobiliser application 1
Figure 2. A PhoneGap application 2
Figure 3. Layers of a PhoneGap application 3
Figure 4. Standard reference application look & feel. 4
Figure 5. A storyboard example 5
Figure 6. Eclipse IDE for Android application development. 6
Figure 7. Application framework code layers. 7
Figure 8. Part of the app.css file 8
Figure 9. Part of the index.html file. 9
Figure 10. An example functions from the SY_Mobiliser.js file 10
Figure 11. Installed application on an Android phone. 11
Figure 12. Example file for Bahasa language 13
Figure 13. Personal information page 14

1. Introduction to Developer Guide for Mobiliser Smartphone

This developer guide provides information about using Sybase® Mobiliser Smartphone product to develop client applications for smartphone mobile phone for the Sybase Money Mobiliser platform.

The audience is Mobile Smartphone developers.

This guide describes requirements for designing and developing a Mobile Smartphone application, how to customize the code, and how to test, secure and deploy the Mobile Smartphone application to the device or simulator.

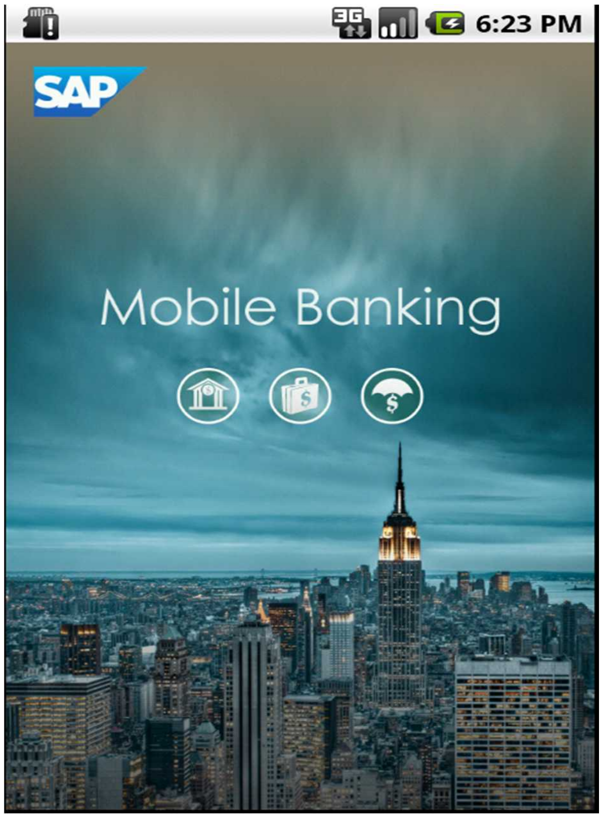


Figure 1. Smartphone Mobiliser application

2. Designing a Smartphone Application

Mobiliser Smartphone Architecture

The Mobiliser Smartphone application is a reference application framework that runs out-of-the-box with any Money Mobiliser server. This framework is built using familiar web technologies that are prevalent within any Information Technology department. The underlying mobile development framework is Adobe PhoneGap, which is an open-source multi-platform mobile application framework.

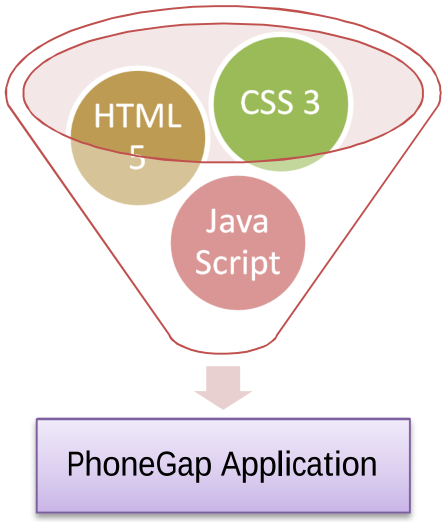


Figure 2. A PhoneGap application

A PhoneGap application is build using HTML 5, CSS 3 and JavaScript code. Developers use these technologies to specify structure of an application layout using HTML 5, to design the look and feel of the application presentation using CSS 3 and finally to implement the business logic for the application using JavaScript.

The PhoneGap application runs inside a browser, which usually is a web-kit based browser running inside the native operating system for mobile platform, for example Safari/iOS on an iPhone mobile phone. PhoneGap provides a wrapper layer implemented using the native code for each specific supported mobile platform to provide the hooks into the mobile device features like: geo-location, camera, accelerometer, and the contact book. Below is a figure 2 that shows how each layer wraps around the one beneath it.

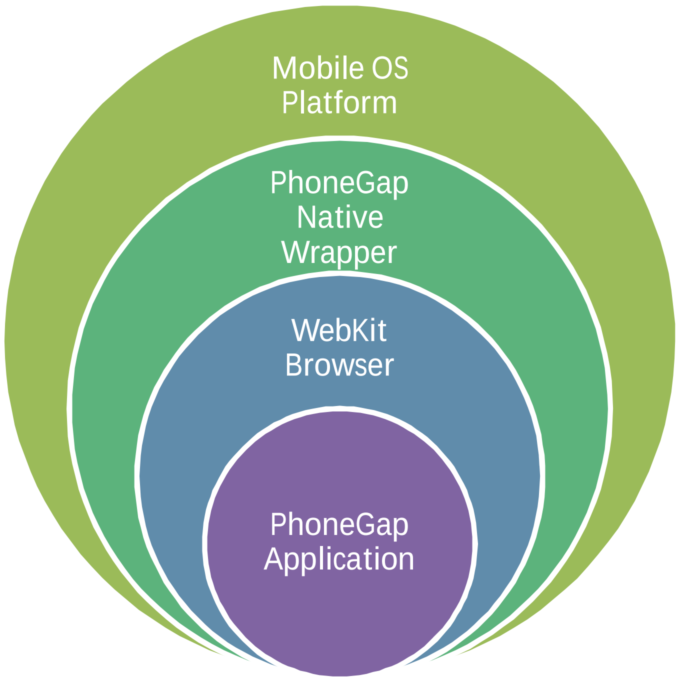


Figure 3. Layers of a PhoneGap application

The Reference Smartphone Application

The reference applications come pre-built with a set of features connected to the back-end server:

- mBanking

  Provides mobile banking functions with various service levels.

- Core Money

  Provides basic mobile wallet functionality and alerts.

- Open Bank API

  Allows signing in via a third party banking system and provides management of checks, various banking accounts and favorites.

They also have a standard SAP® layout, and look and feel, as can be seen below in Figure 3.

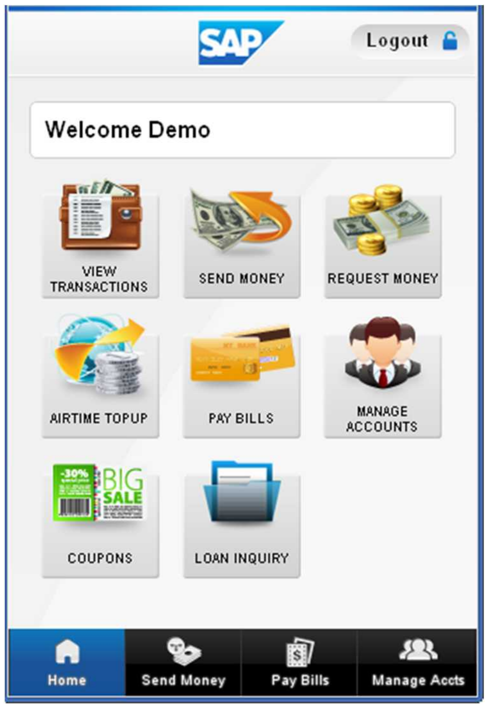


Figure 4. Standard reference application look & feel.

To design a Mobiliser Smartphone application a developer must have a plan for how the different pages that amount to a specific business transaction will interact among each other. (Figure 5 is provided as an example storyboard for one design.)

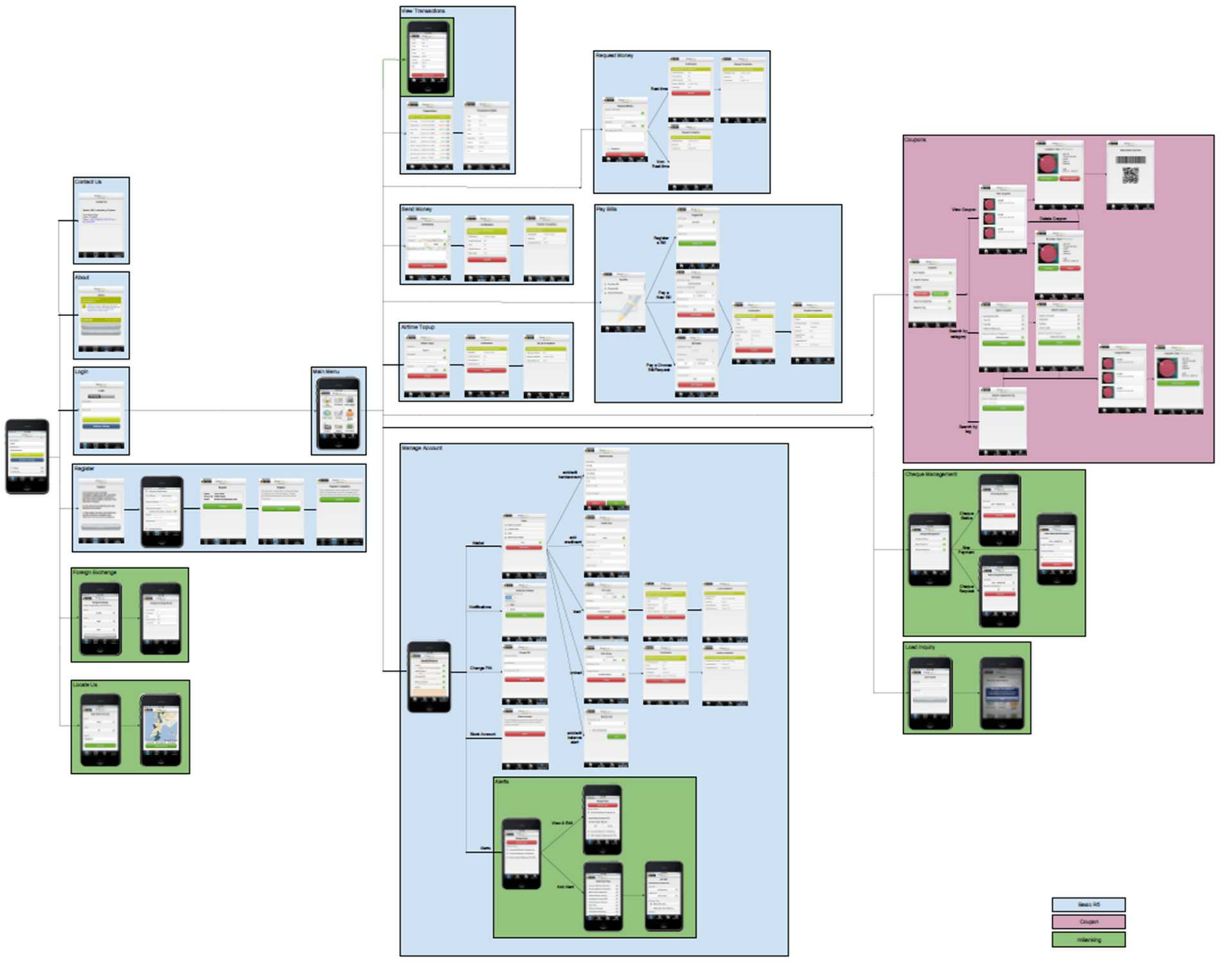


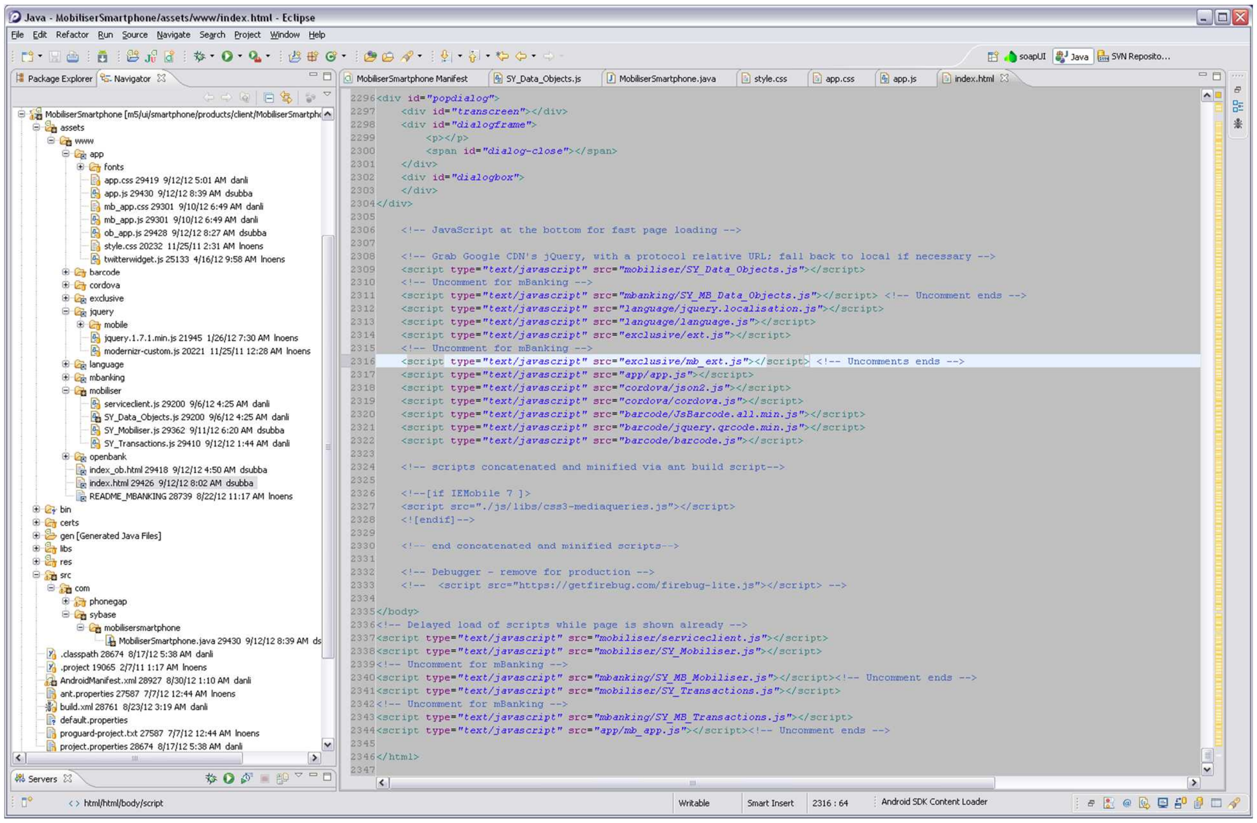
Figure 5. A storyboard example

The look and feel of the application should be provided by graphic designers that can then be implemented as CSS 3 code within the application.

3. Developing a Smartphone Application

Environment Setup

To start development, one needs to setup the development environment for a specific platform. The code can be installed for any of the mobile platforms, such as: Android (see figure 6), Apple iPhone and iPad, Blackberry and others. Please see the *Sybase Mobiliser Platform Installation and Configuration Guide* for the steps to install the development environment and checking out the code from the repository.



```
<div id="popdialog">
    <div id="transcreen"></div>
    <div id="dialogframe">
        <p></p>
        <span id="dialog-close"></span>
    </div>
    <div id="dialogbox">
    </div>
</div>

    <!-- JavaScript at the bottom for fast page loading -->

    <!-- Grab Google CDN's jQuery, with a protocol relative URL; fall back to local if necessary -->
    <script type="text/javascript" src="mobiliser/SY_Data_Objects.js"></script>
    <!-- Uncomment for mBanking -->
    <script type="text/javascript" src="mbanking/SY_MB_Data_Objects.js"></script> <!-- Uncomment ends -->
    <script type="text/javascript" src="language/jquery.localisation.js"></script>
    <script type="text/javascript" src="language/language.js"></script>
    <script type="text/javascript" src="exclusive/ext.js"></script>
    <!-- Uncomment for mBanking -->
    <script type="text/javascript" src="exclusive/mb_ext.js"></script> <!-- Uncomments ends -->
    <script type="text/javascript" src="app/app.js"></script>
    <script type="text/javascript" src="cordova/json2.js"></script>
    <script type="text/javascript" src="cordova/cordova.js"></script>
    <script type="text/javascript" src="barcode/JsBarcode.all.min.js"></script>
    <script type="text/javascript" src="barcode/jquery.qrcode.min.js"></script>
    <script type="text/javascript" src="barcode/barcode.js"></script>

    <!-- scripts concatenated and minified via ant build script-->

    <!--[if IEMobile 7 ]>
    <script src="./js/libs/css3-mediaqueries.js"></script>
    <![endif]-->

    <!-- end concatenated and minified scripts-->

    <!-- Debugger - remove for production -->
    <!--  <script src="https://getfirebug.com/firebug-lite.js"></script> -->

</body>
<!-- Delayed load of scripts while page is shown already -->
<script type="text/javascript" src="mobiliser/serviceclient.js"></script>
<script type="text/javascript" src="mobiliser/SY_Mobiliser.js"></script>
<!-- Uncomment for mBanking -->
<script type="text/javascript" src="mbanking/SY_MB_Mobiliser.js"></script><!-- Uncomment ends -->
<script type="text/javascript" src="mobiliser/SY_Transactions.js"></script>
<!-- Uncomment for mBanking -->
<script type="text/javascript" src="mbanking/SY_MB_Transactions.js"></script>
<script type="text/javascript" src="app/mb_app.js"></script><!-- Uncomment ends -->

</html>
```

Java - MobiliserSmartphone/assets/www/index.html - Eclipse
File Edit Refactor Run Source Navigate Search Project Window Help
Package Explorer
Navigator
MobiliserSmartphone Manifest
SY_Data_Objects.js
MobiliserSmartphone.java
style.css
app.css
app.js
index.html
Servers
html/html/body/script
Writable
Smart Insert
2316 : 64
Android SDK Content Loader

Figure 6. Eclipse IDE for Android application development

Smartphone Code Layers

After the development environment has been setup, there are certain files that most developers will use to build their new application. First, let's take a look at the different layers and modules that make up the main parts of the application framework.

The code in the application framework is generally separated into three logical layers which fulfills a specific function of the application. Figure 7 shows the main files in each layer in the framework.

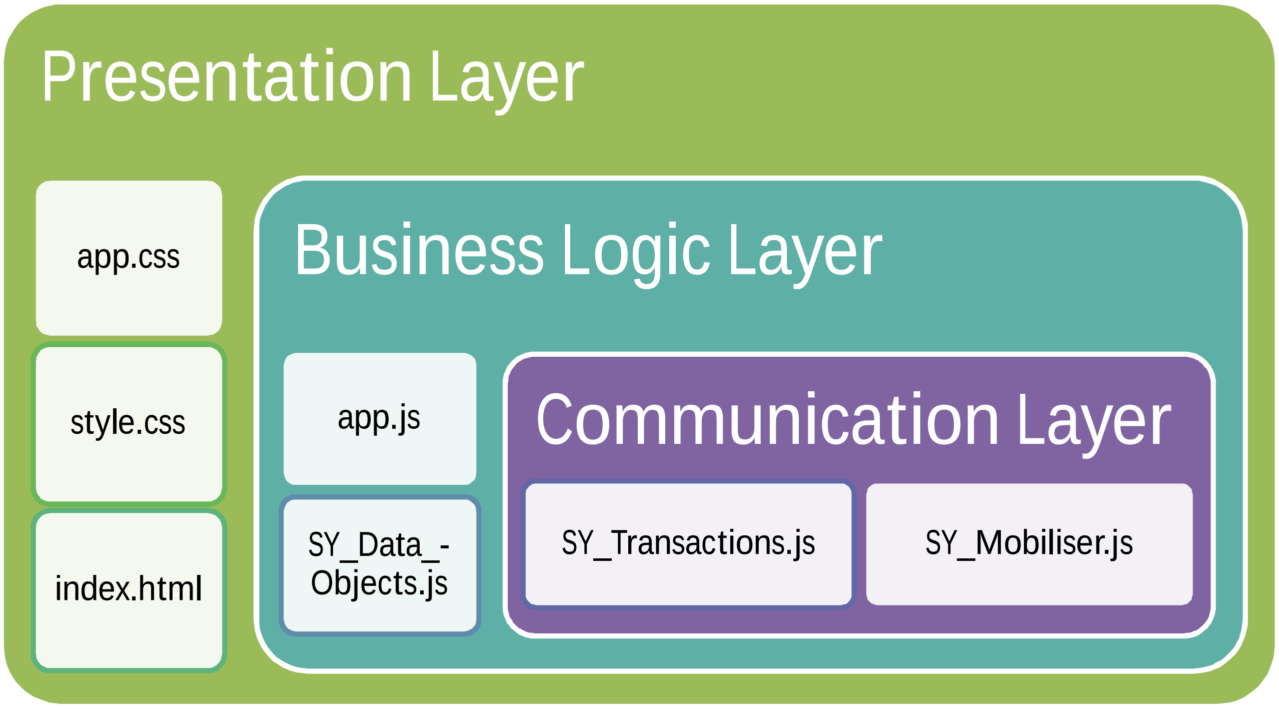


Figure 7. Application framework code layers

Main Functions of the Application

There are eight (8) different main functions in the Mobiliser Smartphone application framework, as shown in figure 4, and are listed below:

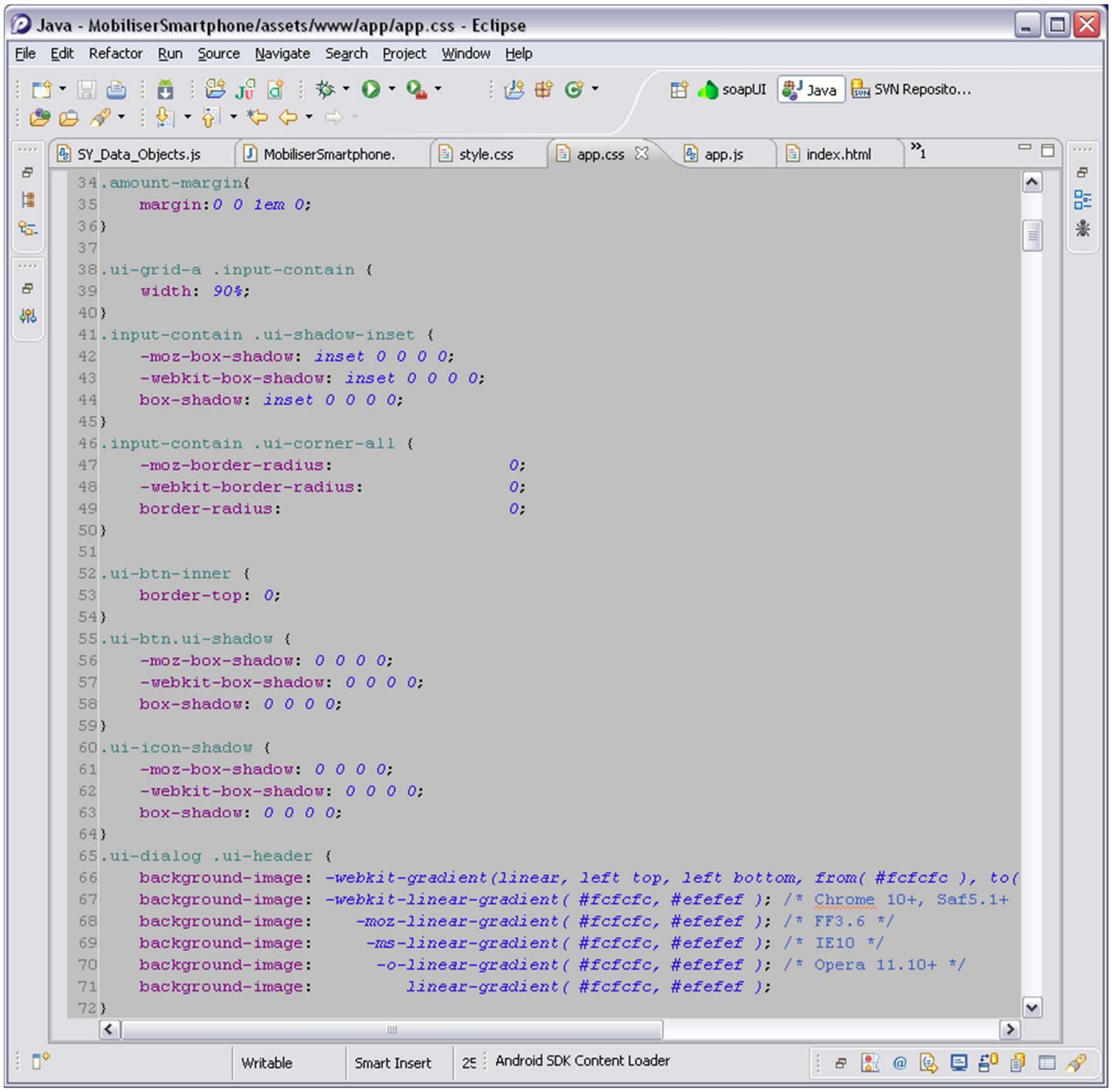
1. Transaction Details
2. Send Money
3. Request Money
4. Airtime Topup
5. Pay Bills
6. Manage Accounts
7. Coupons
8. Loan Inquiry

4. Customizing a Smartphone Application

Custom Look and Feel

The basic change any developer might need is to re-brand the application with the colors and logos of his/her business. The developer then will look at the files in the "Presentation Layer" as listed in figure 7 and modifies the CSS and HTML code as needed.

Below is a view of the app.css file listed CSS code that defined the look and feel of the widgets in the application:


Java - MobiliserSmartphone/assets/www/app/app.css - Eclipse
File Edit Refactor Run Source Navigate Search Project Window Help
soapUI Java SVN Reposito...
SY_Data_Objects.js MobiliserSmartphone. style.css app.css app.js index.html »1
Writable Smart Insert 25 Android SDK Content Loader

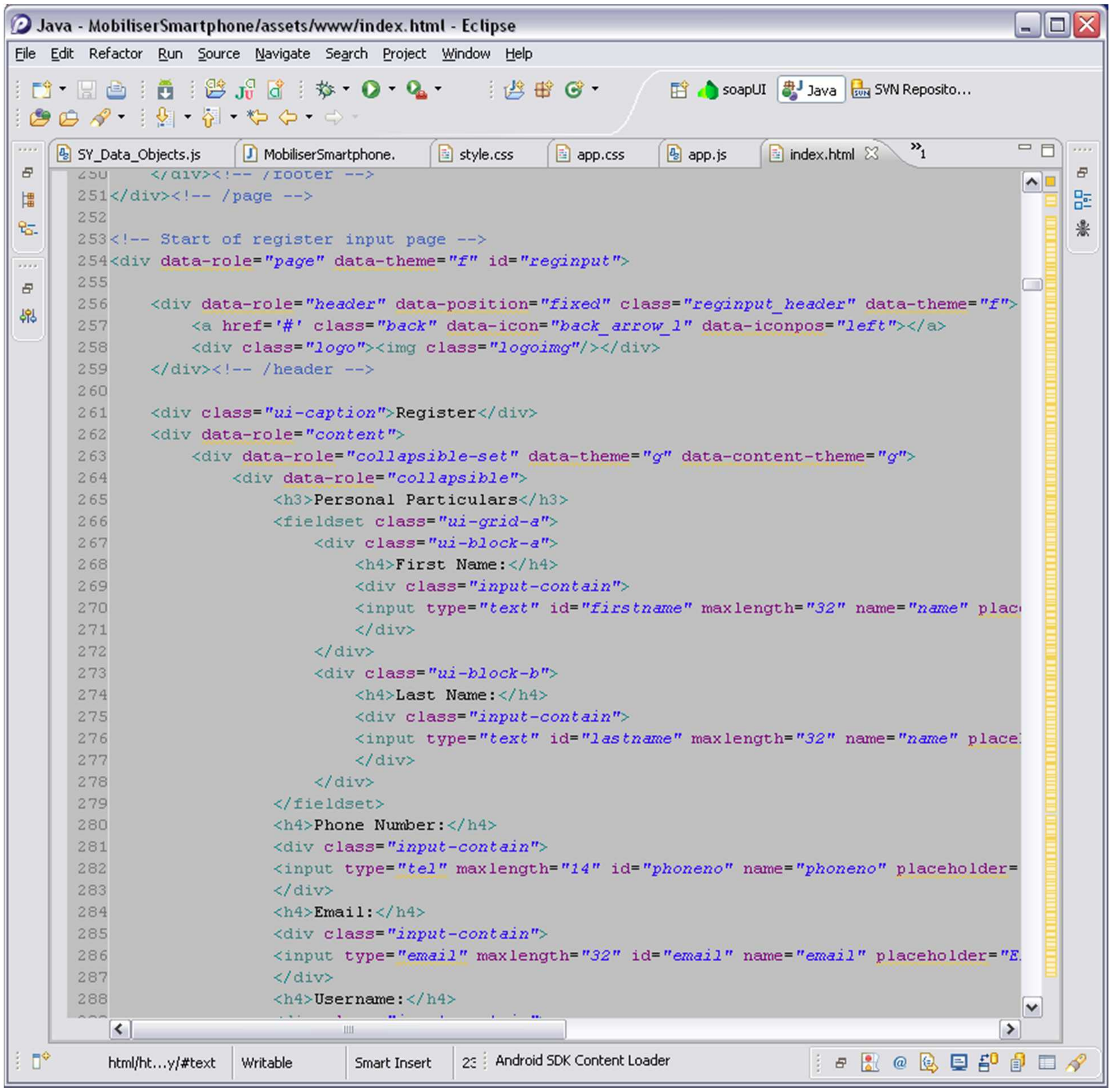
```
.amount-margin{
    margin:0 0 1em 0;
}

.ui-grid-a .input-contain {
    width: 90%;
}
.input-contain .ui-shadow-inset {
    -moz-box-shadow: inset 0 0 0 0;
    -webkit-box-shadow: inset 0 0 0 0;
    box-shadow: inset 0 0 0 0;
}
.input-contain .ui-corner-all {
    -moz-border-radius:                  0;
    -webkit-border-radius:               0;
    border-radius:                       0;
}

.ui-btn-inner {
    border-top: 0;
}
.ui-btn.ui-shadow {
    -moz-box-shadow: 0 0 0 0;
    -webkit-box-shadow: 0 0 0 0;
    box-shadow: 0 0 0 0;
}
.ui-icon-shadow {
    -moz-box-shadow: 0 0 0 0;
    -webkit-box-shadow: 0 0 0 0;
    box-shadow: 0 0 0 0;
}
.ui-dialog .ui-header {
    background-image: -webkit-gradient(linear, left top, left bottom, from( #fcfcfc ), to(
    background-image: -webkit-linear-gradient( #fcfcfc, #efefef ); /* Chrome 10+, Saf5.1+
    background-image:    -moz-linear-gradient( #fcfcfc, #efefef ); /* FF3.6 */
    background-image:     -ms-linear-gradient( #fcfcfc, #efefef ); /* IE10 */
    background-image:      -o-linear-gradient( #fcfcfc, #efefef ); /* Opera 11.10+ */
    background-image:         linear-gradient( #fcfcfc, #efefef );
}
```

Figure 8. Part of the app.css file

Figure 9 shows the register input page in the HTML file index.html. A developer can change the structure of the application by changing the HTML code. Changing images also is done through the HTML and CSS files.


Java - MobiliserSmartphone/assets/www/index.html - Eclipse
File Edit Refactor Run Source Navigate Search Project Window Help
soapUI
Java
SVN Reposito...
SY_Data_Objects.js
MobiliserSmartphone.
style.css
app.css
app.js
index.html
html/ht...y/#text
Writable
Smart Insert
Android SDK Content Loader

```
    </div><!-- /footer -->
</div><!-- /page -->

<!-- Start of register input page -->
<div data-role="page" data-theme="f" id="reginput">

    <div data-role="header" data-position="fixed" class="reginput_header" data-theme="f">
        <a href='#' class="back" data-icon="back_arrow_1" data-iconpos="left"></a>
        <div class="logo"><img class="logoimg"/></div>
    </div><!-- /header -->

    <div class="ui-caption">Register</div>
    <div data-role="content">
        <div data-role="collapsible-set" data-theme="g" data-content-theme="g">
            <div data-role="collapsible">
                <h3>Personal Particulars</h3>
                <fieldset class="ui-grid-a">
                    <div class="ui-block-a">
                        <h4>First Name:</h4>
                        <div class="input-contain">
                        <input type="text" id="firstname" maxlength="32" name="name" plac
                        </div>
                    </div>
                    <div class="ui-block-b">
                        <h4>Last Name:</h4>
                        <div class="input-contain">
                        <input type="text" id="lastname" maxlength="32" name="name" place
                        </div>
                    </div>
                </fieldset>
                <h4>Phone Number:</h4>
                <div class="input-contain">
                <input type="tel" maxlength="14" id="phoneno" name="phoneno" placeholder=
                </div>
                <h4>Email:</h4>
                <div class="input-contain">
                <input type="email" maxlength="32" id="email" name="email" placeholder="E
                </div>
                <h4>Username:</h4>
```

Figure 9. Part of the index.html file.

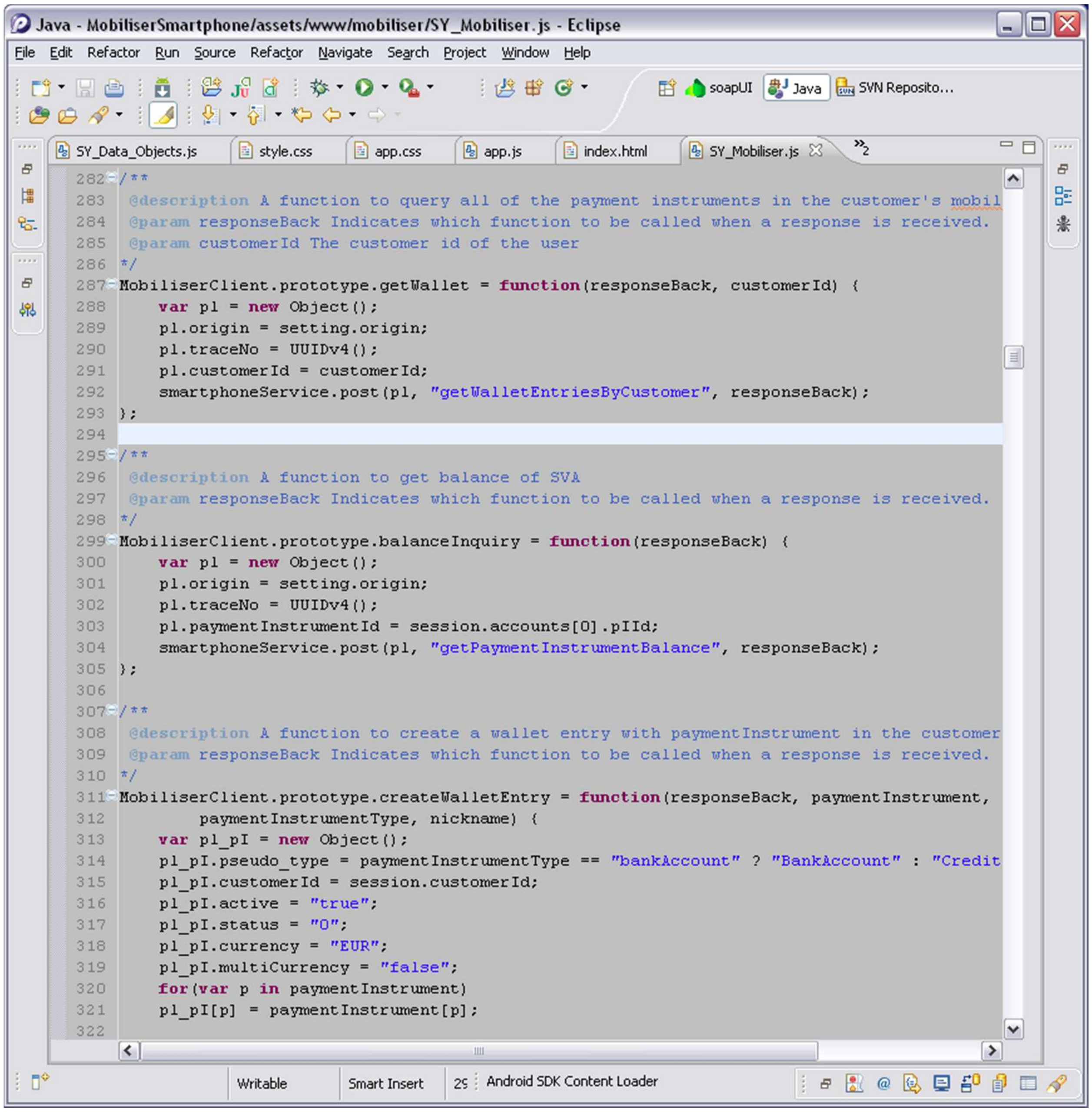
Custom Functionality

Certain applications might not need all of the functions provided by the reference application out-of-the-box or might like to manipulate the data differently. That is where the "Business Logic Layer" comes handy. One can change the way the data is manipulated within these files in addition to the presentation layer files to achieve such customization.

Custom Mobiliser Transactions

Some businesses might need to add custom business transactions to their Money Mobiliser back-end server. For the Smartphone application framework (client) to be able to understand the new parameters and results of the new transaction, the "Communication Layer" files need to be customized.

It is recommended that developers follow the structure currently utilized within the SY_Mobiliser.js file to perform such communication with the back-end Money Mobiliser server. Below is an example:


Java - MobiliserSmartphone/assets/www/mobiliser/SY_Mobiliser.js - Eclipse
File Edit Refactor Run Source Refactor Navigate Search Project Window Help
soapUI
Java
SVN Reposito...
SY_Data_Objects.js
style.css
app.css
app.js
index.html
SY_Mobiliser.js
Writable
Smart Insert
29
Android SDK Content Loader

```
/**
 @description A function to query all of the payment instruments in the customer's mobil
 @param responseBack Indicates which function to be called when a response is received.
 @param customerId The customer id of the user
*/
MobiliserClient.prototype.getWallet = function(responseBack, customerId) {
    var pl = new Object();
    pl.origin = setting.origin;
    pl.traceNo = UUIDv4();
    pl.customerId = customerId;
    smartphoneService.post(pl, "getWalletEntriesByCustomer", responseBack);
};

/**
 @description A function to get balance of SVA
 @param responseBack Indicates which function to be called when a response is received.
*/
MobiliserClient.prototype.balanceInquiry = function(responseBack) {
    var pl = new Object();
    pl.origin = setting.origin;
    pl.traceNo = UUIDv4();
    pl.paymentInstrumentId = session.accounts[0].pIId;
    smartphoneService.post(pl, "getPaymentInstrumentBalance", responseBack);
};

/**
 @description A function to create a wallet entry with paymentInstrument in the customer
 @param responseBack Indicates which function to be called when a response is received.
*/
MobiliserClient.prototype.createWalletEntry = function(responseBack, paymentInstrument,
        paymentInstrumentType, nickname) {
    var pl_pI = new Object();
    pl_pI.pseudo_type = paymentInstrumentType == "bankAccount" ? "BankAccount" : "Credit
    pl_pI.customerId = session.customerId;
    pl_pI.active = "true";
    pl_pI.status = "0";
    pl_pI.currency = "EUR";
    pl_pI.multiCurrency = "false";
    for(var p in paymentInstrument)
    pl_pI[p] = paymentInstrument[p];

```

Figure 10. An example functions from the SY_Mobiliser.js file

Different UI Packages

Currently, the framework uses jQuery® Mobile as its UI package to display the widgets on the pages. There are many other packages that can be used instead, for example Sencha® Touch.

Note that using a different UI package will need changes within the code that is outside the scope of this document.

5. Code Generation/Building

The reference application's code base should be ready to be built and deployed and no special code generation is needed. Depending on the platform you are developing for, the steps to build the code might be different. See the *Sybase Mobiliser Platform Installation and Configuration Guide* for the necessary steps to build the code for each platform.

6. Deploying the Application to Mobiliser Platform

After you build the application and install it on the device, you will get a new icon on your mobile screen such as the one in figure 11 as an example. The reference application comes running out-of-the-box, so no additional deployment instructions are needed (apart from initial configuration; see Section 13) to integrate the Smartphone mobile application to the Money Mobiliser server.

If customization happened on the Money Mobiliser server, then steps should have been taken to integrate the Smartphone application with the server before deployment. See "Custom Mobiliser Transactions" under section 4.

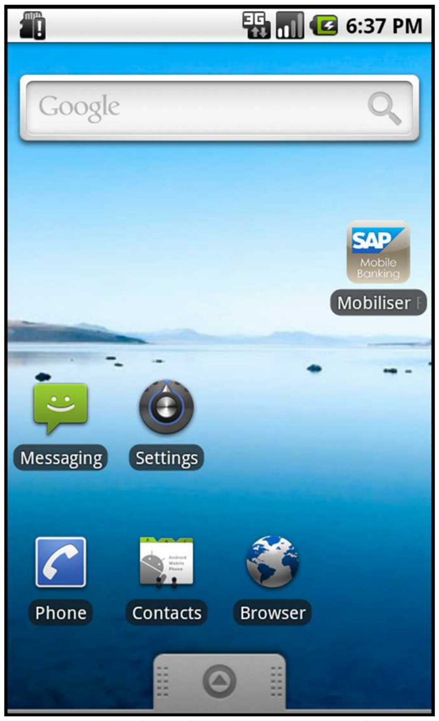


Figure 11. Installed application on an Android phone

7. Provisioning the Application to the Device

Provisioning the finalized application after development is usually done through the official distribution marketplace for each mobile platform:

- iPhone, iPad –App Store
- BlackBerry – BlackBerry App World
- Android – Android Market, Google Play Store

Follow the instructions and policy for each of these distribution channels for provisioning your application through that specific channel.

In the case of Android and BlackBerry applications, it is possible to host and distribute the built package through proprietary means, but that is out-of-scope for this document.

8. Debugging

Although there are lots of tools that will help speed up development, here are the essential tools that can be used during debugging for such a development effort:

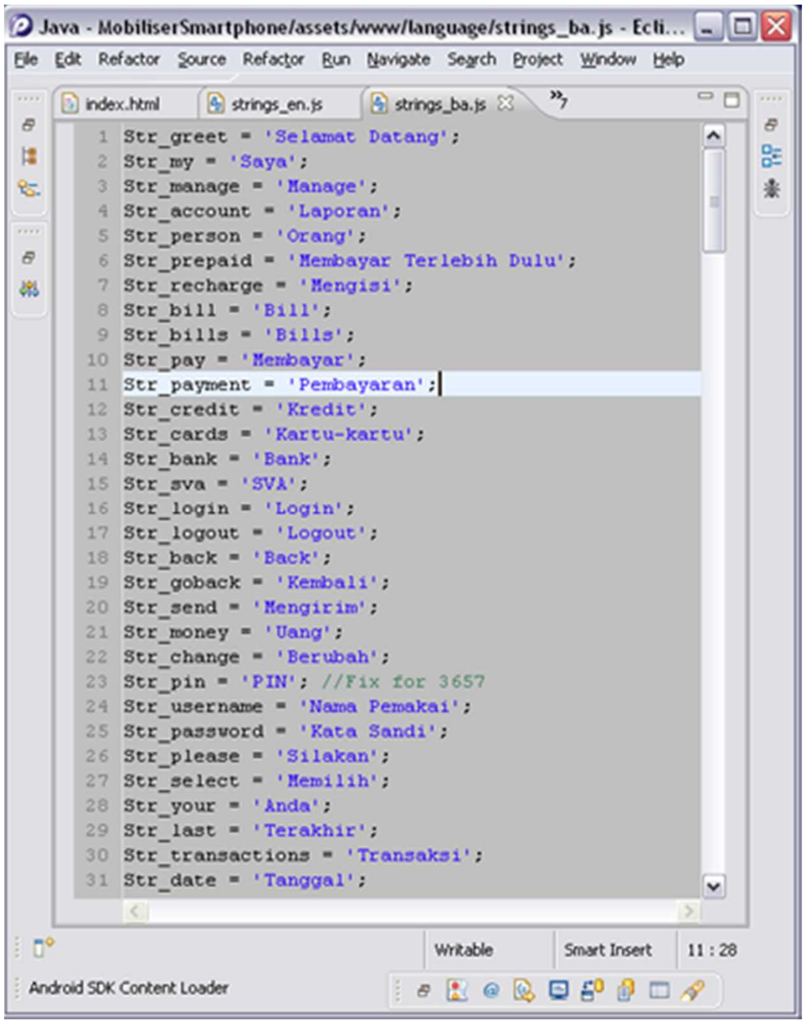
1. Standard browsers, such as Google Chrome, FireFox, or Internet Explorer, can be used because most of the code is in HTML/CSS/JS in addition to using the browser's debugging module for example FireBug on FireFox.
2. Using a TCP monitor to trace the messages that are going to/from the device/browser is very handy.
3. There are newer tools to debug web application directly on the mobile device, but their discussion is out of scope of this document.

9. Testing

Apart from the internal testing of code, it is important to setup a staging system that will resemble the production environment and perform end-to-end system testing including any 3rd party integration system and the Money Mobiliser back-end system.

10. Localizing

Localization, or changing the language of the presentation layer, is very easy using the Mobiliser Smartphone application. It uses the jQuery module for localization, where the different text for each language is defined in "language" folder. Every file that starts with "strings_" holds the texts for all fields for that language. The file ends with the 2-letter language code for example "en" for English and "ba" for Bahasa (Indonesian).


Java - MobiliserSmartphone/assets/www/language/strings_ba.js - Ecli...
File Edit Refactor Source Refactor Run Navigate Search Project Window Help
index.html
strings_en.js
strings_ba.js
Writable
Smart Insert
11 : 28
Android SDK Content Loader

```
Str_greet = 'Selamat Datang';
Str_my = 'Saya';
Str_manage = 'Manage';
Str_account = 'Laporan';
Str_person = 'Orang';
Str_prepaid = 'Membayar Terlebih Dulu';
Str_recharge = 'Mengisi';
Str_bill = 'Bill';
Str_bills = 'Bills';
Str_pay = 'Membayar';
Str_payment = 'Pembayaran';
Str_credit = 'Kredit';
Str_cards = 'Kartu-kartu';
Str_bank = 'Bank';
Str_sva = 'SVA';
Str_login = 'Login';
Str_logout = 'Logout';
Str_back = 'Back';
Str_goback = 'Kembali';
Str_send = 'Mengirim';
Str_money = 'Uang';
Str_change = 'Berubah';
Str_pin = 'PIN'; //Fix for 3657
Str_username = 'Nama Pemakai';
Str_password = 'Kata Sandi';
Str_please = 'Silakan';
Str_select = 'Memilih';
Str_your = 'Anda';
Str_last = 'Terakhir';
Str_transactions = 'Transaksi';
Str_date = 'Tanggal';
```

Figure 12. Example file for Bahasa language

11. Securing the Application

The Smartphone mobile application does not store any data on the mobile device. All of the data will be removed from the mobile phone's memory as soon as the user finishes from using the application. The application is residing inside a browser container which takes care of the security through SSL.

Review the Money Mobiliser Server documentation for further discussion on how the back-end system handles sensitive data and how it stores it securely.

12. Authentication/Registration

Registration

A user can register an account from within the Smartphone application on the device after deployment. Click on the "Register" button on the menu at the bottom, then click "Continue" as shown below:

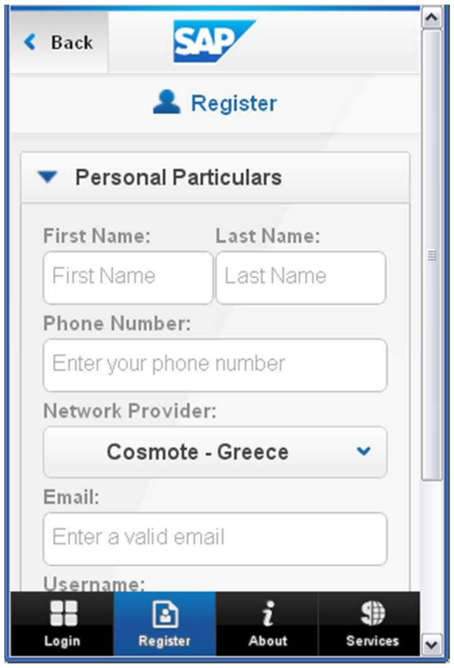


Figure 13. Personal information page

Then click “Authentication” to set a password and click “Continue”, then follow the instruction to finish the registration process. Depending on your back-end configuration, you will be sent a passcode via SMS or you can obtain it through the Channel Manager’s console output. Then, enter it on the screen and click “Confirm.”

Authentication

Once your registration has succeeded, then proceed to the login page and enter your credentials to start using your Smartphone Mobiliser application!

Note: In the mBanking implementation for the Smartphone application, users must have a service level that enables them to login to the application (for example, Platinum service) that can be added by the administrator/agent using the Customer Support Tool.

13. Running Application on the Device

Setting the Server Information

The only needed setup to run the application after it was installed on the device is perhaps the server information. Note that mostly finalized application should be provisioned with the server setup pre-built within the application to skip this step for the user.

At the login page, click “Mobiliser Settings” button and set the required information as is needed based on what your administrator sets for you.

Note: This button appears only for development purposes, and is not available on the hosted Mobile Web application.

14. API Reference

Class MobiliserClient

A thin JavaScript web service client that accesses the Mobiliser platform. It provides an abstraction layer to communicate with the system and returns XML documents as a result.
Defined in: SY_Mobiliser.js.

Class Summary

MobiliserClient()

Method Summary

assignCoupon(responseBack, couponTypeId)
AssignCoupon function

balanceInquiry(responseBack)
A function to get balance of SVA

cancelBill(responseBack, invoiceId)
Cancel bill function

changeCredential(responseBack, customerId, oldCredential, newCredential)
Change Credential function

checkCredential(responseBack, Credential, type)
checkCredential function

confirmVoucher(responseBack, id, ref)
confirmVoucher function

continuePayBill(responseBack, id, ref)
ContinuePayInvoice function

createBalanceAlert(responseBack, threshold, onlyTransition)
createBalanceAlert function

createFullCustomer(responseBack, reginfo, token)
Agent create full customer function

createIdentification(responseBack, customerId, type, identification)
Agent create identification function

createInvoice(responseBack, invoiceConfigurationId, ref, amount, date)
Create an invoice for a customer for a specific type of merchant bill

createInvoiceForInvoiceType(responseBack, invoiceTypeId, reference, amount)
Get types of invoices by group in the system

createNewAlert(responseBack, alertTypeId, alertDataListItems, contactPointsItems, frequencyVal, alertNotificationMsgId)
create a new alert for customer

createSmsToken(responseBack, phoneno)

Agent create sms token function

createWalletEntry(responseBack, paymentInstrument, paymentInstrumentType, nickname)
A function to create a wallet entry with paymentInstrument in the customer's mobile wallet

deleteBalanceAlert(responseBack, alertid)
deleteBalanceAlert function

deleteCoupon(responseBack, couponId)
DeleteCoupon function

deleteCustomerAlert(alert_id, responseBack)
delete an existing alert

deleteCustomerAlertByCustomerAndData(responseBack, pIId)
delete an alert if the account itself is deleted

deleteWalletEntry(responseBack, acctid)
A function to create a wallet entry with paymentInstrument in the customer's mobile wallet

demandForPayment(responseBack, username, password, payer, payee, txn)
DemandOnPayment function

findCouponTypesByTags(responseBack, tag, locale, mimeType)
FindCouponTypesByTags function

findTransactions(responseBack, customerId, maxRecords, paymentInstrumentId)
Get transaction history for a customer

getActiveAlertNotificationMessages(responseBack)
Service call to fetch the active alert notification message mapping, to be used in creating and updating alerts

getAlertDetailsForEdit(responseBack, alert_id)
get alert details for an existing alert

getAlertNotificationMsgId(alertTypeId, notification, notificationMsgTypeId)
Service call to fetch the alert notification msg type id based of alert type id and notification msg id

getBalanceAlert(responseBack)
getBalanceAlert function

getBillTypes(responseBack)
Get all types of invoices in the system

getCategoryTree(responseBack, locale, groupId)
GetCategoryTree function

getChildCategories(responseBack, parentCategoryId, locale)
GetChildCategories function

getCouponTypesForCategory(responseBack, categoryId, locale, mimeType)
getCouponTypesForCategory function

getExistingAlerts(responseBack)
get Existing Alerts function

getIdentifications(responseBack, customerId, type)
Agent get identifications function

getInvoiceTypesByGroup(responseBack)
Get types of invoices by group in the system

getLookups(responseBack, entity)
Function to fetch list of supported look up items like currencies networkproviders etc

getMyCoupons(responseBack, locale, mimeType)
GetMyCoupons function
getOpenInvoices(responseBack, customerId)
Get all active invoices for a customer
getOtherIdentifications(responseBack)
get Customer's other identifications
getRegisteredBills(responseBack, customerId)
Get all configured invoices for a customer
getRootCategories(responseBack, locale, groupId)
GetRootCategories function
getTxnDetails(responseBack, customerId, maxRecords, paymentInstrumentId)
Get details of a transaction
getWallet(responseBack, customerId)
A function to query all of the payment instruments in the customer's mobile wallet
load(responseBack, payerpI, txn)
load function
login(responseBack, username, password)
Agent login function
logout(responseBack)
Agent logout function
payBill(responseBack, invoiceId, payerPaymentInstrumentId)
Pay bill function
preAuthorisationContinue(responseBack, id, ref)
transfer function
purchaseCoupon(responseBack, paymentInstrumentId, paymentInstrumentId)
PurchaseCoupon function
registerSimpleBill(responseBack, customerId, alias, typeId, alias)
Configure an invoice for some merchant bill type for a customer
request(responseBack, payermsisdn, txn)
request function
setCredential(responseBack, customerId, Credential, type)
setCredential function
setPrimary(responseBack, acctid)
A function to set primary wallet in the customer's mobile wallet
startVoucher(responseBack, payercustomerId, payerpI, payeemsisdn, txn)
startVoucher function
topUp(responseBack, invoiceId, payerPaymentInstrumentId)
Top up function
transfer(responseBack, payercustomerId, payerpI, payeemsisdn, txn)
transfer function
unload(responseBack, payeepI, txn)
unload function
unregisterBill(responseBack, invoiceConfigurationId)
Remove a configured invoice for a customer

updateBalanceAlert(responseBack, threshold, onlyTransition, alertid)
updateBalanceAlert function
updateCustomerBlockAccount(responseBack)
Agent update customer function
updateCustomerNotification(responseBack, mode)
Agent update customer function
updateExistingAlert(responseBack, alertId, alertTypeId, alertDataList)
Updates an existing alert
updatePaymentInstrument(responseBack, paymentInstrument, paymentInstrumentType, acctid)
A function to update a paymentInstrument of a wallet entry in the customer's mobile wallet
updateWalletEntry(responseBack, nickname, acctid)
A function to update a wallet entry in the customer's mobile wallet

Class Detail

MobiliserClient()

Method Detail

assignCoupon(responseBack, couponTypeId)
AssignCoupon function
Parameters:
responseBack
Indicates which function to be called when a response is received.
couponTypeId
The id of an coupon

---

balanceInquiry(responseBack)
A function to get balance of SVA
Parameters:
responseBack
Indicates which function to be called when a response is received.

---

cancelBill(responseBack, invoiceId)
Cancel bill function
Parameters:
responseBack
Indicates which function to be called when a response is received.
invoiceId
The id of an invoice

---

changeCredential(responseBack, customerId, oldCredential, newCredential)
Change Credential function
Parameters:

responseBack

Indicates which function to be called when a response is received.

customerId

The customer id of the user

oldCredential

The old Credential

newCredential

The new Credential chosen by the user

---

checkCredential(responseBack, Credential, type)

checkCredential function

Parameters:

responseBack

Indicates which function to be called when a response is received.

Credential

The PIN or Password to set

type

The type of Credential

---

confirmVoucher(responseBack, id, ref)

confirmVoucher function

Parameters:

responseBack

Indicates which function to be called when a response is received.

id

The systemId of previous StartVoucher

ref

The Reference of previous StartVoucher

---

continuePayBill(responseBack, id, ref)

ContinuePayInvoice function

Parameters:

responseBack

Indicates which function to be called when a response is received.

id

The system id of checkPayInvoice transaction

ref

The reference of checkPayInvoice transaction

---

createBalanceAlert(responseBack, threshold, onlyTransition)

createBalanceAlert function

Parameters:

responseBack

Indicates which function to be called when a response is received.

threshold

onlyTransition

---

createFullCustomer(responseBack, reginfo, token)
Agent create full customer function
Parameters:
responseBack
  Indicates which function to be called when a response is received.
reginfo
token

---

createIdentification(responseBack, customerId, type, identification)
Agent create identification function
Parameters:
responseBack
  Indicates which function to be called when a response is received.
customerId
type
identification

---

createInvoice(responseBack, invoiceConfigurationId, ref, amount, date)
Create an invoice for a customer for a specific type of merchant bill
Parameters:
responseBack
  Indicates which function to be called when a response is received.
invoiceConfigurationId
  The id of an invoice configuration for a customer
ref
  The reference number of an invoice
amount
  The amount of money to pay in cents
date

---

createInvoiceForInvoiceType(responseBack, invoiceTypeId, reference, amount)
Get types of invoices by group in the system
Parameters:
responseBack
  Indicates which function to be called when a response is received.
invoiceTypeId
reference
amount

---

createNewAlert(responseBack, alertTypeId, alertDataListItems, contactPointsItems, frequencyVal, alertNotificationMsgId)
create a new alert for customer
Parameters:
responseBack
  Indicates which function to be called when a successful response is received.

alertTypeId
    Type of alert to be created
alertDataListItems
    alert data item objects list
contactPointsItems
    contact point objects list
frequencyVal
    Value of frequency Everytime or First time
alertNotificationMsgId
    text or conv

---

createSmsToken(responseBack, phoneno)
Agent create sms token function
Parameters:
responseBack
    Indicates which function to be called when a response is received.
phoneno

---

createWalletEntry(responseBack, paymentInstrument, paymentInstrumentType, nickname)
A function to create a wallet entry with paymentInstrument in the customer's mobile wallet
Parameters:
responseBack
    Indicates which function to be called when a response is received.
paymentInstrument
paymentInstrumentType
nickname

---

deleteBalanceAlert(responseBack, alertid)
deleteBalanceAlert function
Parameters:
responseBack
    Indicates which function to be called when a response is received.
alertid

---

deleteCoupon(responseBack, couponId)
DeleteCoupon function
Parameters:
responseBack
    Indicates which function to be called when a response is received.
couponId
    The id of an coupon

---

deleteCustomerAlert(alert_id, responseBack)
delete an existing alert
Parameters:
alert_id

id of an alert which needs to be deleted

responseBack

Indicate which function to be called when a response is received.

---

deleteCustomerAlertByCustomerAndData(responseBack, pIId)

delete an alert if the account itself is deleted

Parameters:

responseBack

function to be called in case of success

pIId

payment instrument id of the existing alert data key record

---

deleteWalletEntry(responseBack, acctid)

A function to create a wallet entry with paymentInstrument in the customer's mobile wallet

Parameters:

responseBack

Indicates which function to be called when a response is received.

acctid

---

demandForPayment(responseBack, username, password, payer, payee, txn)

DemandOnPayment function

Parameters:

responseBack

Indicates which function to be called when a response is received.

username

The username should normally be the msisdn in addition to its country code i.e. +18881234567

password

The user password

payer

The Customer object which will be making the payment

payee

The Customer object that will be receiving the payment

txn

The TxnData object that contains txn details

---

findCouponTypesByTags(responseBack, tag, locale, mimeType)

FindCouponTypesByTags function

Parameters:

responseBack

Indicates which function to be called when a response is received.

tag

The tag of the coupon

locale

The location of the coupon

mimeType

The mimetype of the coupon

findTransactions(responseBack, customerId, maxRecords, paymentInstrumentId)
Get transaction history for a customer
Parameters:
responseBack
Indicates which function to be called when a response is received.
customerId
The customer id of the user
maxRecords
The max number of transactions to return
paymentInstrumentId
The id of the payment instrument

---

getActiveAlertNotificationMessages(responseBack)
Service call to fetch the active alert notification message mapping, to be used in creating and updating alerts
Parameters:
responseBack
callback handler when the results returned.

---

getAlertDetailsForEdit(responseBack, alert_id)
get alert details for an existing alert
Parameters:
responseBack
function to be called in case of success
alert_id
id of the existing alert record

---

getAlertNotificationMsgId(alertTypeId, notification, notificationMsgTypeId)
Service call to fetch the alert notification msg type id based of alert type id and notification msg id
Parameters:
alertTypeId
alert type
notification
type
notificationMsgTypeId

---

getBalanceAlert(responseBack)
getBalanceAlert function
Parameters:
responseBack
Indicates which function to be called when a response is received.

---

getBillTypes(responseBack)
Get all types of invoices in the system
Parameters:

responseBack

Indicates which function to be called when a response is received.

---

getCategoryTree(responseBack, locale, groupId)

GetCategoryTree function

Parameters:

responseBack

Indicates which function to be called when a response is received.

locale

The location of the coupon

groupId

The group id of the coupon

---

getChildCategories(responseBack, parentCategoryId, locale)

GetChildCategories function

Parameters:

responseBack

Indicates which function to be called when a response is received.

parentCategoryId

The parent id of the coupon

locale

The location of the coupon

---

getCouponTypesForCategory(responseBack, categoryId, locale, mimeType)

getCouponTypesForCategory function

Parameters:

responseBack

Indicates which function to be called when a response is received.

categoryId

The category of the coupon

locale

The location of the coupon

mimeType

The mimetype of the coupon

---

getExistingAlerts(responseBack)

get Existing Alerts function

Parameters:

responseBack

Indicate which function to be called when a response is received.

---

getIdentifications(responseBack, customerId, type)

Agent get identifications function

Parameters:

responseBack

Indicates which function to be called when a response is received.

customerId
type

---

getInvoiceTypesByGroup(responseBack)
Get types of invoices by group in the system
Parameters:
responseBack
Indicates which function to be called when a response is received.

---

getLookups(responseBack, entity)
Function to fetch list of supported look up items like currencies networkproviders etc
Parameters:
responseBack
the callback handler
entity
to be looked up on the server

---

getMyCoupons(responseBack, locale, mimeType)
GetMyCoupons function
Parameters:
responseBack
Indicates which function to be called when a response is received.
locale
The location of the coupon
mimeType
The mimetype of the coupon

---

getOpenInvoices(responseBack, customerId)
Get all active invoices for a customer
Parameters:
responseBack
Indicates which function to be called when a response is received.
customerId
The customer id of the user

---

getOtherIdentifications(responseBack)
get Customer's other identifications
Parameters:
responseBack
Indicate which function to be called when a response is received.

---

getRegisteredBills(responseBack, customerId)
Get all configured invoices for a customer
Parameters:
responseBack
Indicates which function to be called when a response is received.

customerId
The customer id of the user

---

getRootCategories(responseBack, locale, groupId)
GetRootCategories function
Parameters:
responseBack
Indicates which function to be called when a response is received.
locale
The location of the coupon
groupId
The group id of the coupon

---

getTxnDetails(responseBack, customerId, maxRecords, paymentInstrumentId)
Get details of a transaction
Parameters:
responseBack
Indicates which function to be called when a response is received.
customerId
The customer id of the user
maxRecords
The max number of transactions to return
paymentInstrumentId
The id of the payment instrument

---

getWallet(responseBack, customerId)
A function to query all of the payment instruments in the customer's mobile wallet
Parameters:
responseBack
Indicates which function to be called when a response is received.
customerId
The customer id of the user

---

load(responseBack, payerpI, txn)
load function
Parameters:
responseBack
Indicates which function to be called when a response is received.
payerpI
The paymentInstrument Id of the Customer which will use to load fund to SVA
txn
The TxnData object that contains txn details

---

login(responseBack, username, password)
Agent login function
```
    var loginBack = function(r, xmlResponse) { ... // handle response };
    mc.login(loginBack, "user1", "pass2");
```

Parameters:
responseBack
Indicates which function to be called when a response is received.
username
The username should normally be the msisdn in addition to its country code i.e. +18881234567
password
The user password

---

logout(responseBack)
Agent logout function
Parameters:
responseBack
Indicates which function to be called when a response is received.

---

payBill(responseBack, invoiceId, payerPaymentInstrumentId)
Pay bill function
Parameters:
responseBack
Indicates which function to be called when a response is received.
invoiceId
The id of an invoice
payerPaymentInstrumentId
The payment instrument id for the payer

---

preAuthorisationContinue(responseBack, id, ref)
transfer function
Parameters:
responseBack
Indicates which function to be called when a response is received.
id
The systemId of previous PreAuthorisation
ref
The Reference of previous PreAuthorisation

---

purchaseCoupon(responseBack, paymentInstrumentId, paymentInstrumentId)
PurchaseCoupon function
Parameters:
responseBack
Indicates which function to be called when a response is received.
paymentInstrumentId
The instrument id of an coupon
paymentInstrumentId

---

registerSimpleBill(responseBack, customerId, alias, typeId, alias)
Configure an invoice for some merchant bill type for a customer
Parameters:

responseBack
Indicates which function to be called when a response is received.
customerId
The customer id of the user
alias
The name the customer gives for this invoice configuration
typeId
The id of the invoice type
alias

---

request(responseBack, payermsisdn, txn)
request function
Parameters:
responseBack
Indicates which function to be called when a response is received.
payermsisdn
The Customer msisdn that will receive the request
txn
The TxnData object that contains txn details

---

setCredential(responseBack, customerId, Credential, type)
setCredential function
Parameters:
responseBack
Indicates which function to be called when a response is received.
customerId
The customer id of the user
Credential
The PIN or Password to set
type
The type of Credential

---

setPrimary(responseBack, acctid)
A function to set primary wallet in the customer's mobile wallet
Parameters:
responseBack
Indicates which function to be called when a response is received.
acctid

---

startVoucher(responseBack, payercustomerId, payerpI, payeemsisdn, txn)
startVoucher function
Parameters:
responseBack
Indicates which function to be called when a response is received.
payercustomerId
The Customer customerId which will make the payment

payerpI
The paymentInstrument Id of the Customer which will use to make the payment
payeemsisdn
The Customer msisdn that will receive the payment
txn
The TxnData object that contains txn details

---

topUp(responseBack, invoiceId, payerPaymentInstrumentId)
Top up function
Parameters:
responseBack
Indicates which function to be called when a response is received.
invoiceId
The id of an invoice
payerPaymentInstrumentId
The payment instrument id for the payer

---

transfer(responseBack, payercustomerId, payerpI, payeemsisdn, txn)
transfer function
Parameters:
responseBack
Indicates which function to be called when a response is received.
payercustomerId
The customerId of payer
payerpI
The paymentInstrumentId of payer
payeemsisdn
The msisdn of payee receiving the payment
txn
The TxnData object that contains txn details

---

unload(responseBack, payeepI, txn)
unload function
Parameters:
responseBack
Indicates which function to be called when a response is received.
payeepI
The paymentInstrument Id of the Customer which will use to receive fund from SVA
txn
The TxnData object that contains txn details

---

unregisterBill(responseBack, invoiceConfigurationId)
Remove a configured invoice for a customer
Parameters:
responseBack
Indicates which function to be called when a response is received.

invoiceConfigurationId
    The id of an invoice configuration for a customer

---

updateBalanceAlert(responseBack, threshold, onlyTransition, alertid)
updateBalanceAlert function
Parameters:
responseBack
    Indicates which function to be called when a response is received.
threshold
onlyTransition
alertid

---

updateCustomerBlockAccount(responseBack)
Agent update customer function
Parameters:
responseBack
    Indicates which function to be called when a response is received.

---

updateCustomerNotification(responseBack, mode)
Agent update customer function
Parameters:
responseBack
    Indicates which function to be called when a response is received.
mode

---

updateExistingAlert(responseBack, alertId, alertTypeId, alertDataList)
Updates an existing alert
Parameters:
responseBack
    Response Handler
alertId
    Id of customer alert
alertTypeId
    alert type id
alertDataList
    alert data list

---

updatePaymentInstrument(responseBack, paymentInstrument, paymentInstrumentType, acctid)
A function to update a paymentInstrument of a wallet entry in the customer's mobile wallet
Parameters:
responseBack
    Indicates which function to be called when a response is received.
paymentInstrument
paymentInstrumentType
acctid

---

updateWalletEntry(responseBack, nickname, acctid)
A function to update a wallet entry in the customer's mobile wallet
Parameters:
responseBack
	Indicates which function to be called when a response is received.
nickname
acctid
