



Smartphone Mobiliser API Reference

Smartphone Mobiliser 5.1 SP03

DOCUMENT ID: DC01981-01-0513-01

LAST REVISED: August 2013

Copyright © 2013 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

| | |
|--|----------|
| Money Mobiliser Core API | 1 |
| Accounts class | 1 |
| AlertData class | 2 |
| BalanceAlert class | 2 |
| BillTypes class | 3 |
| Category class | 3 |
| ContactPoint class | 4 |
| Coupon class | 4 |
| CouponType class | 5 |
| CouponTypeDescription class | 5 |
| Customer class | 6 |
| CustomerAlert class | 7 |
| GeoLocation class | 7 |
| LoginSession class | 8 |
| MobiliserClient class | 10 |
| acceptTermAndConditions function | 10 |
| assignCoupon function | 11 |
| balanceInquiry function | 11 |
| cancelBill function | 12 |
| changeCredential function | 12 |
| checkCredential function | 13 |
| confirmVoucher function | 13 |
| continuePayBill function | 14 |
| createBalanceAlert function | 14 |
| createFullCustomer function | 14 |
| createIdentification function | 15 |
| createInvoice function | 15 |
| createInvoiceForInvoiceType function | 16 |
| createNewAlert function | 16 |
| createSmsToken function | 17 |
| createWalletEntry function | 17 |

| | |
|--|----|
| deleteBalanceAlert function | 18 |
| deleteCoupon function | 18 |
| deleteCustomerAlert function | 19 |
| deleteCustomerAlertByCustomerAndData function | 19 |
| deleteWalletEntry function | 20 |
| demandForPayment function | 20 |
| fbssologin function | 21 |
| findCouponTypesByTags function | 21 |
| findNextAddresses function | 22 |
| findNextCouponTypesByTags function | 22 |
| findNextCouponTypesForCategory function | 23 |
| findTransactions function | 23 |
| getActiveAlertNotificationMessages function | 24 |
| getAlertDetailsForEdit function | 24 |
| getAlertNotificationMsgId function | 24 |
| getBalanceAlert function | 25 |
| getBillTypes function | 25 |
| getCategoryTree function | 26 |
| getChildCategories function | 26 |
| getCouponTypesForCategory function | 27 |
| getDefaultTermAndConditions function | 27 |
| getDownloadPictureURL function | 27 |
| getExistingAlerts function | 28 |
| getIdentifications function | 28 |
| getInvoiceTypesByGroup function | 29 |
| getLookups function | 29 |
| getMyCoupons function | 29 |
| getOpenInvoices function | 30 |
| getOpenTermAndConditions function | 30 |
| getOtherIdentifications function | 31 |
| getRegisteredBills function | 31 |
| getRootCategories function | 32 |
| getRsaPublicKey function | 32 |
| getTxnDetails function | 33 |

| | |
|---|----|
| getUploadPictureURL function | 33 |
| getWallet function | 34 |
| load function | 34 |
| login function | 35 |
| logout function | 35 |
| payBill function | 36 |
| preAuthorisationContinue function | 36 |
| purchaseCoupon function | 37 |
| registerSimpleBill function | 37 |
| request function | 38 |
| setCredential function | 38 |
| setPrimary function | 39 |
| startVoucher function | 39 |
| topUp function | 40 |
| transfer function | 40 |
| unload function | 41 |
| unregisterBill function | 41 |
| updateBalanceAlert function | 42 |
| updateCustomerBlockAccount function | 42 |
| updateCustomerNotification function | 43 |
| updateExistingAlert function | 43 |
| updatePaymentInstrument function | 43 |
| updateWalletEntry function | 44 |
| OpenBill class | 44 |
| OtherIdentification class | 45 |
| Registeredbill class | 45 |
| Setting class | 46 |
| Timer class | 46 |
| Transaction class | 47 |
| TxnData class | 47 |
| Global | 48 |
| AlertNotificationMessage function | 48 |
| checkOtherIdentificationInSelectedAlert function | 48 |
| createNewAlertBack function | 49 |

| | |
|--|------------|
| deleteCustomerAlertBack function | 49 |
| findAlertDataInDataList function | 49 |
| getActiveAlertNotificationMessagesBack function | 50 |
| getAlertDetailsForEditBack function | 50 |
| getAlertNotificationMsgIdBack function | 50 |
| getExistingAlertsBack function | 51 |
| getTransactionTypesBack function | 51 |
| Identification function | 51 |
| listOtherIdentifications function | 52 |
| parseAlertNotificationMessages function | 52 |
| parseAlertNotificationMessageTypeId function | 52 |
| parseCustomerAlert function | 53 |
| parseExistingAlerts function | 53 |
| parseLookupEntities function | 53 |
| parseOtherIdentifications function | 54 |
| updateExistingAlertBack function | 54 |
| Source code | 54 |
| SY_Data_Objects.js file | 54 |
| SY_Mobiliser.js file | 70 |
| SY_Transactions.js file | 127 |
| Open Bank API | 219 |
| Account class | 219 |
| Amount class | 220 |
| Authentication class | 220 |
| Customer class | 220 |
| Denomination class | 221 |
| Favourites class | 222 |
| LoginSession class | 222 |
| MobiliserClient class | 223 |
| denominations function | 224 |
| favouriteInfo function | 224 |
| favouriteList function | 224 |
| favouriteTransfer function | 225 |
| fundTransfer function | 225 |

| | |
|--|-----|
| getAccountInfo function | 226 |
| getAccountList function | 226 |
| getBankDetailsList function | 227 |
| getChequeStatus function | 227 |
| getExchangeRate function | 227 |
| getLookups function | 228 |
| logout function | 228 |
| mBankingTopUp function | 228 |
| obLogin function | 229 |
| placeChequeBookRequest function | 229 |
| placeChequeStopPayment function | 230 |
| transactionList function | 230 |
| OrgUnit class | 231 |
| Setting class | 231 |
| Timer class | 232 |
| Transaction class | 232 |
| Global | 233 |
| alertXML function | 233 |
| denominationsBack function | 233 |
| favouriteInfoBack function | 233 |
| favouriteListBack function | 234 |
| favouriteTransferBack function | 234 |
| fundTransferBack function | 234 |
| getAccountInfoBack function | 235 |
| getAccountListBack function | 235 |
| getBankDetailsListBack function | 236 |
| getChequeStatusResponse function | 236 |
| getChequeStopResponse function | 236 |
| getCurrenciesBack function | 237 |
| getExchangeRateBack function | 237 |
| getTransactionListBack function | 237 |
| MBankingTopUpBack function | 238 |
| parseAccountInfo function | 238 |
| parseAccountList function | 238 |
| parseBankDetails function | 239 |

| | |
|--|------------|
| parseChequeStatus function | 239 |
| parseCurrencies function | 240 |
| parseDenomination function | 240 |
| parseExchangeRate function | 241 |
| parseFavouriteList function | 241 |
| parseLookupEntities function | 242 |
| parseTransactionList function | 242 |
| placeChequeBookRequestBack function | 243 |
| Source code | 243 |
| SY_OB_Data_Objects.js file | 243 |
| SY_OB_Mobiliser.js file | 251 |
| SY_OB_Transactions.js file | 267 |
| mBanking API | 297 |
| Global | 297 |
| acceptTermsAndConditionBack function | 297 |
| getCurrenciesBack function | 297 |
| getCustomerServicePackageRequestBack function | 298 |
| getExchangeRateBack function | 298 |
| getNetworkProvidersBack function | 298 |
| mbLoginBack function | 299 |
| NetworkProvider function | 299 |
| parseCurrencies function | 299 |
| parseCustomerServicePackage function | 300 |
| parseNetworkProviders function | 300 |
| sendTxnAsSMSBack function | 301 |
| ServicePackage function | 301 |
| Source code | 301 |
| SY_MB_Data_Objects.js file | 301 |
| SY_MB_Mobiliser.js file | 303 |
| SY_MB_Transactions.js file | 307 |

Money Mobiliser Core API

The Money Mobiliser Core API provides basic mobile wallet functionality, so that customers, for example, can pay bills, top up mobile devices, transfer money from person to person, receive offers and coupons, create mobile alerts, and find ATM and branch locations.

Accounts class

Account A class that stores information pertaining to certain account.

Syntax

```
new Accounts(pIID, type, no, info, acctHolderName, acctNumber, displayNumber, extra1, extra2, extra3, extra4)
```

Parameters

| Name | Type | Description |
|-----------------------|------|---|
| <i>pIID</i> | | The paymentInstrument id of an account |
| <i>type</i> | | The type of an account |
| <i>no</i> | | The number related to an account, such as card no. account no. |
| <i>info</i> | | The extra information of an account, alias name of bank account and credit card. balance of SVA |
| <i>acctHolderName</i> | | The name of bank account/ credit card Holder |
| <i>acctNumber</i> | | Bank account number/credit card number |
| <i>displayNumber</i> | | The number to be displayed for bank account/credit card. |

| | | |
|---------------|--|--|
| <i>extra1</i> | | The code of bank or The type of the credit card, master, visa or etc. |
| <i>extra2</i> | | The code of branch that bank account is opened or The security code of the credit card |
| <i>extra3</i> | | The name of bank or The year of the expiry date for the card |
| <i>extra4</i> | | The country of the bank or The month of the expiry date for the card |

Source

SY_Data_Objects.js line 145 on page 60.

AlertData class

AlertData this contains key value pair of alert data

Syntax

```
new AlertData()
```

Source

SY_Data_Objects.js line 443 on page 69.

BalanceAlert class

BalanceAlert is a class that stores information about a Balance Alert.

Syntax

```
new BalanceAlert( billConfigId, alias, billTypeId, billTypeName )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>billConfigId</i> | | The id of the bill configuration |
| <i>alias</i> | | The alias name given by the user for this configuration. |

| | | |
|---------------------|--|--|
| <i>billTypeId</i> | | The type id for the invoice. |
| <i>billTypeName</i> | | The name of the invoice/bill registered for. |

Source

SY_Data_Objects.js line 237 on page 63.

BillTypes class

BillTypes is a class that stores information about any invoice that the user has registered for.

Syntax

```
new BillTypes( billTypeId, billTypeName )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>billTypeId</i> | | The type id for the invoice. |
| <i>billTypeName</i> | | The name of the invoice/bill registered for. |

Source

SY_Data_Objects.js line 186 on page 61.

Category class

Contains basic information about a coupon category.

Syntax

```
new Category( id, internalName, caption, priority, noOfChildCategories, parent )
```

Parameters

| Name | Type | Description |
|---------------------|------|-------------|
| <i>id</i> | | |
| <i>internalName</i> | | |
| <i>caption</i> | | |

| | | |
|----------------------------|--|--|
| <i>priority</i> | | |
| <i>noOfChildCategories</i> | | |
| <i>parent</i> | | |

Source

SY_Data_Objects.js line 329 on page 66.

ContactPoint class

ContactPoint holds many to many mapping record between Customer Alert and Other Identification

Syntax

`new ContactPoint()`

Source

SY_Data_Objects.js line 433 on page 69.

Coupon class

Contains basic information about a coupon.

Syntax

`new Coupon(id, customerId, couponType, status, serialNumber, validTo, code, views, uses)`

Parameters

| Name | Type | Description |
|---------------------|------|---------------------------------|
| <i>id</i> | | |
| <i>customerId</i> | | |
| <i>couponType</i> | | the type of coupon. |
| <i>status</i> | | the coupon's status |
| <i>serialNumber</i> | | The serial number of the coupon |
| <i>validTo</i> | | valid date of the coupon |
| <i>code</i> | | The code of coupon |

| | | |
|--------------|--|---------------------------|
| <i>views</i> | | view times of the coupon |
| <i>uses</i> | | usage times of the coupon |

Source

SY_Data_Objects.js line 261 on page 63.

CouponType class

Contains basic information about a coupon type.

Syntax

```
new CouponType( id, name, purchasePrice, purchaseCurrency, max Views, maxUses,
isRestorableByUser, description )
```

Parameters

| Name | Type | Description |
|---------------------------|------|--------------------------|
| <i>id</i> | | |
| <i>name</i> | | |
| <i>purchasePrice</i> | | |
| <i>purchaseCurrency</i> | | |
| <i>max Views</i> | | |
| <i>maxUses</i> | | |
| <i>isRestorableByUser</i> | | |
| <i>description</i> | | Coupon type description. |

Source

SY_Data_Objects.js line 287 on page 64.

CouponTypeDescription class

Contains basic information about a coupon type description.

Syntax

```
new CouponTypeDescription(  mimeType, caption, content, thumbnailUrl, imageUrl )
```

Parameters

| Name | Type | Description |
|---------------------|------|-------------|
| <i>contentType</i> | | |
| <i>caption</i> | | |
| <i>content</i> | | |
| <i>thumbnailUrl</i> | | |
| <i>imageUrl</i> | | |

Source

SY_Data_Objects.js line 309 on page 65.

Customer class

Contains basic information about the customer.

Syntax

`new Customer(customerId, msisdn, blacklist, test, typeId, cancelId, modeId)`

Parameters

| Name | Type | Description |
|-------------------|--------|--|
| <i>customerId</i> | | |
| <i>msisdn</i> | string | The msisdn of the customer |
| <i>blacklist</i> | | An id for blacklistreason to indicate customer account status. |
| <i>test</i> | | test state of customer account. |
| <i>typeId</i> | | The id of customerType |
| <i>cancelId</i> | | The id of cancellationReason of customer account |
| <i>modeId</i> | | The id of mode of txn receipt mode of customer account |

Source*SY_Data_Objects.js* line 27 on page 55.

CustomerAlert class

CustomerAlert which contains alert data set for each alert

Syntax

```
new CustomerAlert( customerId, alertTypeId, alertNotificationMsgId, alertDataList,
contactPointList, blackoutList, created )
```

Parameters

| Name | Type | Description |
|-------------------------------|------|--|
| <i>customerId</i> | | to which this alert belongs |
| <i>alertTypeId</i> | | to which this alert belongs |
| <i>alertNotificationMsgId</i> | | to which notification type this alert is tied to |
| <i>alertDataList</i> | | array of AlertData objects |
| <i>contactPointList</i> | | array of ContactPoint objects |
| <i>blackoutList</i> | | array of Blackout objects |
| <i>created</i> | | |

Source*SY_Data_Objects.js* line 397 on page 68.

GeoLocation class

Contains basic information about geoLocation.

Syntax

```
new GeoLocation( id, internalName )
```

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|---------------------|--|--|
| <i>id</i> | | |
| <i>internalName</i> | | |

Source

SY_Data_Objects.js line 347 on page 66.

LoginSession class

LoginSession This is the session class which stores all of the user's data throughout a login session.

Syntax

new LoginSession(sessionId, customerId, displayName, msisdn, transaction, contacts, billTypes, registeredbill, openbill, openBillArray, openRequestArray, accounts, offlinesvas, totalaccounts, customer, svaalert, coupons, couponTypes, categories, category_grandparent, identification, customerAlert, alertNotificationMessages, otherIdentifications, transit_value, image, uploadUrl, downloads, locations, defaultTermAndConditions, openTermAndConditions, gcmregid)

Parameters

| Name | Type | Description |
|-----------------------|------|--|
| <i>sessionId</i> | | The session id as returned by the server |
| <i>customerId</i> | | The id of the customer |
| <i>displayName</i> | | The display name of the customer |
| <i>msisdn</i> | | The phone number of the customer |
| <i>transaction</i> | | Is an array of transactions |
| <i>contacts</i> | | Is an array of contacts from the address book of the user's smartphone |
| <i>billTypes</i> | | Is an array of available invoice types |
| <i>registeredbill</i> | | A list of all registered bills for the user |

| | | |
|----------------------------------|--|---|
| <i>openbill</i> | | The information of an open bill |
| <i>openBillArray</i> | | A list of all open and due bills for the user |
| <i>openRequestArray</i> | | A list of all open and due demand for payment for the user |
| <i>accounts</i> | | A list of money accounts registered by the user |
| <i>offlinesvas</i> | | A list of offline SVA accounts registered by the user |
| <i>totalaccounts</i> | | total number of money accounts registered by the user |
| <i>customer</i> | | particular information of the user |
| <i>svaalert</i> | | A list of balance alerts registered for SVA of the user |
| <i>coupons</i> | | A list of coupons available for the user |
| <i>couponTypes</i> | | A list of coupon types in system |
| <i>categories</i> | | A list of coupon categories in one level of category tree |
| <i>category_grandparent</i> | | the category node in up two level of current node |
| <i>identification</i> | | A list of identification registered by the user |
| <i>customerAlert</i> | | particular information of an alert for a user |
| <i>alertNotificationMessages</i> | | A list of alert Notification message |
| <i>otherIdentifications</i> | | A list of additional identification registered by the user |
| <i>transit_value</i> | | A variable to hold the temporary data between pages in navigation |

| | | |
|---------------------------------|--|--|
| <i>image</i> | | A handler to hold the image |
| <i>uploadUrl</i> | | The url address in system for uploading a image |
| <i>downloads</i> | | A list of handlers of images to download from system |
| <i>locations</i> | | A list of locations in system |
| <i>defaultTermAndConditions</i> | | A list of default term & conditions in registration |
| <i>openTermAndConditions</i> | | A list of open term & conditions after user login |
| <i>gcmregid</i> | | The GCM registration ID of this application |

Source

SY_Data_Objects.js line 94 on page 58.

MobiliserClient class

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.

Syntax

`new MobiliserClient()`

Source

SY_Mobiliser.js line 18 on page 71.

acceptTermAndConditions function

accept Terms and Conditions information

Syntax

`acceptTermAndConditions(responseBack)`

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|---------------------|--|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |
|---------------------|--|---|

Source

SY_Mobiliser.js line 1583 on page 125.

assignCoupon function

AssignCoupon function

Syntax

assignCoupon(*responseBack*, *couponTypeId*)

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>couponTypeId</i> | | The id of an coupon |

Source

SY_Mobiliser.js line 1072 on page 108.

balanceInquiry function

A function to get balance of SVA

Syntax

balanceInquiry(*responseBack*)

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 352 on page 83.

cancelBill function

Cancel bill function

Syntax

`cancelBill(responseBack, invoiceId)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>invoiceId</i> | | The id of an invoice |

Source

SY_Mobiliser.js line 1052 on page 107.

changeCredential function

Change Credential function

Syntax

`changeCredential(responseBack, customerId, oldCredential, newCredential)`

Parameters

| Name | Type | Description |
|----------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>customerId</i> | | The customer id of the user |
| <i>oldCredential</i> | | The old Credential |
| <i>newCredential</i> | | The new Credential chosen by the user |

Source

SY_Mobiliser.js line 302 on page 81.

checkCredential function

checkCredential function

SyntaxcheckCredential(*responseBack*, *Credential*, *type*)**Parameters**

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>Credential</i> | | The PIN or Password to set |
| <i>type</i> | | The type of Credential |

Source*SY_Mobiliser.js line 321* on page 81.**confirmVoucher function**

confirmVoucher function

SyntaxconfirmVoucher(*responseBack*, *id*, *ref*)**Parameters**

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>id</i> | | The systemId of previous StartVoucher |
| <i>ref</i> | | The Reference of previous StartVoucher |

Source*SY_Mobiliser.js line 813* on page 99.

continuePayBill function

ContinuePayInvoice function

Syntax

`continuePayBill(responseBack, id, ref)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>id</i> | | The system id of checkPayInvoice transaction |
| <i>ref</i> | | The reference of checkPayInvoice transaction |

Source

SY_Mobiliser.js line 1033 on page 106.

createBalanceAlert function

createBalanceAlert function

Syntax

`createBalanceAlert(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 926 on page 103.

createFullCustomer function

Agent create full customer function

Syntax

```
createFullCustomer( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 131 on page 75.

createIdentification function

Agent create identification function

Syntax

```
createIdentification( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 246 on page 79.

createInvoice function

Create an invoice for a customer for a specific type of merchant bill

Syntax

```
createInvoice( responseBack, invoiceConfigurationId, ref, amount, date )
```

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|-------------------------------|--|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>invoiceConfigurationId</i> | | The id of an invoice configuration for a customer |
| <i>ref</i> | | The reference number of an invoice |
| <i>amount</i> | | The amount of money to pay in cents |
| <i>date</i> | | |

Source

SY_Mobiliser.js line 644 on page 93.

createInvoiceForInvoiceType function

Get types of invoices by group in the system

Syntax

`createInvoiceForInvoiceType(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 534 on page 89.

createNewAlert function

create a new alert for customer

Syntax

`createNewAlert(responseBack, alertTypeId, alertDataListItems, contactPointsItems, frequencyVal, alertNotificationMsgId)`

Parameters

| Name | Type | Description |
|-------------------------------|------|---|
| <i>responseBack</i> | | Indicates which function to be called when a successful response is received. |
| <i>alertTypeId</i> | | Type of alert to be created |
| <i>alertDataListItems</i> | | alert data item objects list |
| <i>contactPointsItems</i> | | contact point objects list |
| <i>frequencyVal</i> | | Value of frequency Everytime or First time |
| <i>alertNotificationMsgId</i> | | text or conv |

Source

SY_Mobiliser.js line 1257 on page 114.

createSmsToken function

Agent create sms token function

Syntax

`createSmsToken(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 114 on page 74.

createWalletEntry function

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

Syntax

`createWalletEntry(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 366 on page 83.

deleteBalanceAlert function

deleteBalanceAlert function

Syntax

deleteBalanceAlert(*responseBack*)

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 984 on page 105.

deleteCoupon function

DeleteCoupon function

Syntax

deleteCoupon(*responseBack*, *couponId*)

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>couponId</i> | | The id of an coupon |

Source*SY_Mobiliser.js line 1225* on page 113.**deleteCustomerAlert function**

delete an existing alert

Syntax`deleteCustomerAlert(alert_id, responseBack)`**Parameters**

| Name | Type | Description |
|---------------------|------|---|
| <i>alert_id</i> | | id of an alert which needs to be deleted |
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source*SY_Mobiliser.js line 1323* on page 116.**deleteCustomerAlertByCustomerAndData function**

delete an alert if the account itself is deleted

Syntax`deleteCustomerAlertByCustomerAndData(responseBack, pIID)`**Parameters**

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | function to be called in case of success |
| <i>pIID</i> | | payment instrument id of the existing alert data key record |

Source*SY_Mobiliser.js line 1338* on page 117.

deleteWalletEntry function

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

Syntax

`deleteWalletEntry(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 468 on page 87.

demandForPayment function

DemandOnPayment function

Syntax

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

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>username</i> | | The username should normally be the msisdn in addition to its country code i.e. +18881234567 |
| <i>password</i> | | The user password |
| <i>payer</i> | | The Customer object which will be making the payment |
| <i>payee</i> | | The Customer object that will be receiving the payment |

| | | |
|------------|--|--|
| <i>txn</i> | | The TxnData object that contains txn details |
|------------|--|--|

Source

SY_Mobiliser.js line 836 on page 100.

fbssologin function

Agent security login function

Syntax

`fbssologin(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 69 on page 73.

findCouponTypesByTags function

FindCouponTypesByTags function

Syntax

`findCouponTypesByTags(responseBack, tag, locale, mimeType)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>tag</i> | | The tag of the coupon |
| <i>locale</i> | | The location of the coupon |
| <i>mimeType</i> | | The mimetype of the coupon |

Source

SY_Mobiliser.js line 1090 on page 108.

findNextAddresses function

find next address with location

Syntax

`findNextAddresses(responseBack, lat, lng)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |
| <i>lat</i> | | The latitude of current geolocation. |
| <i>lng</i> | | The longitude of current geolocation. |

Source

SY_Mobiliser.js line 1535 on page 124.

findNextCouponTypesByTags function

FindNextCouponTypesByTags function

Syntax

`findNextCouponTypesByTags(responseBack, tag, locale, mimeType)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>tag</i> | | The tag of the coupon |
| <i>locale</i> | | The location of the coupon |
| <i>mimeType</i> | | The mimetype of the coupon |

Source

SY_Mobiliser.js line 1600 on page 126.

findNextCouponTypesForCategory function

findNextCouponTypesForCategory function

Syntax

```
findNextCouponTypesForCategory( responseBack, categoryId, locale,  
mimeType )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>categoryId</i> | | The category of the coupon |
| <i>locale</i> | | The location of the coupon |
| <i>mimeType</i> | | The mimetype of the coupon |

Source

SY_Mobiliser.js line 1624 on page 127.

findTransactions function

Get transaction history for a customer

Syntax

```
findTransactions( responseBack, customerId, maxRecords, paymentInstrumentId )
```

Parameters

| Name | Type | Description |
|----------------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>customerId</i> | | The customer id of the user |
| <i>maxRecords</i> | | The max number of transactions to return |
| <i>paymentInstrumentId</i> | | The id of the payment instrument |

Source

SY_Mobiliser.js line 486 on page 87.

getActiveAlertNotificationMessages function

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

Syntax

`getActiveAlertNotificationMessages(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | callback handler when the results returned. |

Source

SY_Mobiliser.js line 1463 on page 121.

getAlertDetailsForEdit function

get alert details for an existing alert

Syntax

`getAlertDetailsForEdit(responseBack, alert_id)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | function to be called in case of success |
| <i>alert_id</i> | | id of the existing alert record |

Source

SY_Mobiliser.js line 1355 on page 118.

getAlertNotificationMsgId function

Service call to fetch the alert notification msg type id based of alert type id and notification msg id

Syntax

```
getAlertNotificationMsgId( alertTypeId, notification )
```

Parameters

| Name | Type | Description |
|---------------------|------|-------------|
| <i>alertTypeId</i> | | alert type |
| <i>notification</i> | | type |

Source

SY_Mobiliser.js line 1447 on page 121.

getBalanceAlert function

getBalanceAlert function

Syntax

```
getBalanceAlert( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 947 on page 103.

getBillTypes function

Get all types of invoices in the system

Syntax

```
getBillTypes( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 553 on page 90.

getCategoryTree function

GetCategoryTree function

Syntax

`getCategoryTree(responseBack, locale, groupId)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>locale</i> | | The location of the coupon |
| <i>groupId</i> | | The group id of the coupon |

Source

SY_Mobiliser.js line 1193 on page 112.

getChildCategories function

GetChildCategories function

Syntax

`getChildCategories(responseBack, parentCategoryId, locale)`

Parameters

| Name | Type | Description |
|-------------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>parentCategoryId</i> | | The parent id of the coupon |
| <i>locale</i> | | The location of the coupon |

Source

SY_Mobiliser.js line 1115 on page 109.

getCouponTypesForCategory function

getCouponTypesForCategory function

Syntax

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

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>categoryId</i> | | The category of the coupon |
| <i>locale</i> | | The location of the coupon |
| <i>mimeType</i> | | The mimetype of the coupon |

Source

SY_Mobiliser.js line 1134 on page 110.

getDefaultTermAndConditions function

get default Terms and Conditions information

Syntax

getDefaultTermAndConditions(*responseBack*)

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source

SY_Mobiliser.js line 1554 on page 124.

getDownloadPictureURL function

get picture download URL

Syntax

`getDownloadPictureURL(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source

SY_Mobiliser.js line 1520 on page 123.

getExistingAlerts function

get Existing Alerts function

Syntax

`getExistingAlerts(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source

SY_Mobiliser.js line 1239 on page 113.

getIdentifications function

Agent get identifications function

Syntax

`getIdentifications(responseBack)`

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|---------------------|--|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
|---------------------|--|--|

Source

SY_Mobiliser.js line 265 on page 80.

getInvoiceTypesByGroup function

Get types of invoices by group in the system

Syntax

`getInvoiceTypesByGroup(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 519 on page 89.

getLookups function

Function to fetch list of supported look up items like currencies networkproviders etc

Syntax

`getLookups(responseBack, entity)`

Parameters

| Name | Type | Description |
|---------------------|------|-------------------------------|
| <i>responseBack</i> | | the callback handler |
| <i>entity</i> | | to be looked up on the server |

Source

SY_Mobiliser.js line 1478 on page 122.

getMyCoupons function

GetMyCoupons function

Syntax

`getMyCoupons(responseBack, locale, mimeType)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>locale</i> | | The location of the coupon |
| <i>mimeType</i> | | The mimetype of the coupon |

Source

SY_Mobiliser.js line 1158 on page 111.

getOpenInvoices function

Get all active invoices for a customer

Syntax

`getOpenInvoices(responseBack, customerId)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>customerId</i> | | The customer id of the user |

Source

SY_Mobiliser.js line 625 on page 92.

getOpenTermAndConditions function

get open Terms and Conditions information

Syntax

`getOpenTermAndConditions(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source

SY_Mobiliser.js line 1569 on page 125.

getOtherIdentifications function

get Customer's other identifications

Syntax

getOtherIdentifications(*responseBack*)

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source

SY_Mobiliser.js line 1493 on page 122.

getRegisteredBills function

Get all configured invoices for a customer

Syntax

getRegisteredBills(*responseBack*, *customerId*)

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>customerId</i> | | The customer id of the user |

Source

SY_Mobiliser.js line 568 on page 90.

getRootCategories function

GetRootCategories function

Syntax

`getRootCategories(responseBack, locale, groupId)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>locale</i> | | The location of the coupon |
| <i>groupId</i> | | The group id of the coupon |

Source

SY_Mobiliser.js line 1176 on page 111.

getRsaPublicKey function

Agent request RSA public key from end point

Syntax

`getRsaPublicKey(responseBack, type)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>type</i> | | Specifies account type ("bank" or "card") |

Source

SY_Mobiliser.js line 83 on page 73.

getTxnDetails function

Get details of a transaction

Syntax

```
getTxnDetails( responseBack, customerId, maxRecords, paymentInstrumentId )
```

Parameters

| Name | Type | Description |
|----------------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>customerId</i> | | The customer id of the user |
| <i>maxRecords</i> | | The max number of transactions to return |
| <i>paymentInstrumentId</i> | | The id of the payment instrument |

Source

SY_Mobiliser.js line 505 on page 88.

getUploadPictureURL function

get picture upload URL

Syntax

```
getUploadPictureURL( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source

SY_Mobiliser.js line 1507 on page 123.

getWallet function

A function to query all of the payment instruments in the customer's mobile wallet

Syntax

`getWallet(responseBack, customerId)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>customerId</i> | | The customer id of the user |

Source

SY_Mobiliser.js line 338 on page 82.

load function

load function

Syntax

`load(responseBack, payerPI, txn)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>payerPI</i> | | The paymentInstrument Id of the Customer which will use to load fund to SVA |
| <i>txn</i> | | The TxnData object that contains txn details |

Source

SY_Mobiliser.js line 874 on page 101.

login function

Agent login function

Syntax

```
login( responseBack, username, password )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>username</i> | | The username should normally be the msisdn in addition to its country code i.e. +18881234567 |
| <i>password</i> | | The user password |

Example

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

Source

SY_Mobiliser.js line 45 on page 72.

logout function

Agent logout function

Syntax

```
logout( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 100 on page 74.

payBill function

Pay bill function

Syntax

`payBill(responseBack, invoiceId, payerPaymentInstrumentId)`

Parameters

| Name | Type | Description |
|---------------------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>invoiceId</i> | | The id of an invoice |
| <i>payerPaymentInstrumentId</i> | | The payment instrument id for the payer |

Source

SY_Mobiliser.js line 1016 on page 106.

preAuthorisationContinue function

preAuthorisationContinue function

Syntax

`preAuthorisationContinue(responseBack, id, ref)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>id</i> | | The systemId of previous PreAuthorisation |
| <i>ref</i> | | The Reference of previous PreAuthorisation |

Source

SY_Mobiliser.js line 710 on page 95.

purchaseCoupon function

PurchaseCoupon function

Syntax

```
purchaseCoupon( responseBack, paymentInstrumentId )
```

Parameters

| Name | Type | Description |
|----------------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>paymentInstrumentId</i> | | The instrument id of an coupon |

Source

SY_Mobiliser.js line 1209 on page 112.

registerSimpleBill function

Configure an invoice for some merchant bill type for a customer

Syntax

```
registerSimpleBill( responseBack, customerId, alias, typeId )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>customerId</i> | | The customer id of the user |
| <i>alias</i> | | The name the customer gives for this invoice configuration |
| <i>typeId</i> | | The id of the invoice type |

Source

SY_Mobiliser.js line 586 on page 91.

request function

request function

Syntax

```
request( responseBack, payermisdn, txn )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>payermisdn</i> | | The Customer misdn that will receive the request |
| <i>txn</i> | | The TxnData object that contains txn details |

Source

SY_Mobiliser.js line 729 on page 96.

setCredential function

setCredential function

Syntax

```
setCredential( responseBack, customerId, Credential, type )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>customerId</i> | | The customer id of the user |
| <i>Credential</i> | | The PIN or Password to set |
| <i>type</i> | | The type of Credential |

Source

SY_Mobiliser.js line 283 on page 80.

setPrimary function

A function to set primary wallet in the customer's mobile wallet

Syntax

```
setPrimary( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 420 on page 85.

startVoucher function

startVoucher function

Syntax

```
startVoucher( responseBack, payercustomerId, payerpI, payeemsisdn, txn )
```

Parameters

| Name | Type | Description |
|------------------------|------|---|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>payercustomerId</i> | | The Customer customerId which will make the payment |
| <i>payerpI</i> | | The paymentInstrument Id of the Customer which will use to make the payment |
| <i>payeemsisdn</i> | | The Customer msisdn that will receive the payment |
| <i>txn</i> | | The TxnData object that contains txn details |

Source

SY_Mobiliser.js line 772 on page 97.

topUp function

Top up function

Syntax

`topUp(responseBack, invoiceId, payerPaymentInstrumentId)`

Parameters

| Name | Type | Description |
|---------------------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>invoiceId</i> | | The id of an invoice |
| <i>payerPaymentInstrumentId</i> | | The payment instrument id for the payer |

Source

SY_Mobiliser.js line 1000 on page 105.

transfer function

transfer function

Syntax

`transfer(responseBack, payercustomerId, payerpl, payeemsisdn, txn)`

Parameters

| Name | Type | Description |
|------------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>payercustomerId</i> | | The customerId of payer |
| <i>payerpl</i> | | The paymentInstrumentId of payer |

| | | |
|--------------------|--|--|
| <i>payeemsisdn</i> | | The msisdn of payee receiving the payment |
| <i>txn</i> | | The TxnData object that contains txn details |

Source

SY_Mobiliser.js line 669 on page 94.

unload function

unload function

Syntax

unload(*responseBack*, *payeePI*, *txn*)

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>payeePI</i> | | The paymentInstrument Id of the Customer which will use to receive fund from SVA |
| <i>txn</i> | | The TxnData object that contains txn details |

Source

SY_Mobiliser.js line 901 on page 102.

unregisterBill function

Remove a configured invoice for a customer

Syntax

unregisterBill(*responseBack*, *invoiceConfigurationId*)

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|-------------------------------|--|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |
| <i>invoiceConfigurationId</i> | | The id of an invoice configuration for a customer |

Source

SY_Mobiliser.js line 610 on page 92.

updateBalanceAlert function

updateBalanceAlert function

Syntax

updateBalanceAlert(*responseBack*)

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 962 on page 104.

updateCustomerBlockAccount function

Agent update customer function

Syntax

updateCustomerBlockAccount(*responseBack*)

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 220 on page 78.

updateCustomerNotification function

Agent update customer function

Syntax

```
updateCustomerNotification(responseBack)
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 194 on page 77.

updateExistingAlert function

Updates an existing alert

Syntax

```
updateExistingAlert(responseBack, alertId, alertTypeId, alertDataList)
```

Parameters

| Name | Type | Description |
|----------------------|------|----------------------|
| <i>responseBack</i> | | Response Handler |
| <i>alertId</i> | | Id of customer alert |
| <i>alertTypeId</i> | | alert type id |
| <i>alertDataList</i> | | alert data list |

Source

SY_Mobiliser.js line 1372 on page 118.

updatePaymentInstrument function

A function to update a paymentInstrument of a wallet entry in the customer's mobile wallet

Syntax

```
updatePaymentInstrument(responseBack)
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 442 on page 86.

updateWalletEntry function

A function to update a wallet entry in the customer's mobile wallet

Syntax

`updateWalletEntry(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_Mobiliser.js line 400 on page 84.

OpenBill class

OpenBill A class representing an open invoice which the user has to pay for a certain merchant.

Syntax

`new OpenBill(billId, billTypeName, dueDate, amount)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>billId</i> | | The id of this invoice/bill |
| <i>billTypeName</i> | | The name of the bill type for this invoice |

| | | |
|----------------|--|------------------------------|
| <i>dueDate</i> | | The due date of this bill |
| <i>amount</i> | | The amount due for this bill |

Source

SY_Data_Objects.js line 201 on page 61.

OtherIdentification class

OtherIdentifications this contains other identifications for a customer

Syntax

```
new OtherIdentification( customerId, type )
```

Parameters

| Name | Type | Description |
|-------------------|------|--|
| <i>customerId</i> | | to which this Other Identification belongs |
| <i>type</i> | | of identification fax, email, phone etc |

Source

SY_Data_Objects.js line 418 on page 69.

Registeredbill class

Registeredbill is a class that stores information about any invoice that the user has registered for.

Syntax

```
new Registeredbill( billConfigId, alias, billTypeId, billTypeName )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>billConfigId</i> | | The id of the bill configuration |
| <i>alias</i> | | The alias name given by the user for this configuration. |

| | | |
|---------------------|--|--|
| <i>billTypeId</i> | | The type id for the invoice. |
| <i>billTypeName</i> | | The name of the invoice/bill registered for. |

Source

SY_Data_Objects.js line 219 on page 62.

Setting class

Setting is a connection configurations class to connect to the back-end server.

Syntax

`new Setting(protocol, ipaddress, port, wsname)`

Parameters

| Name | Type | Description |
|------------------|------|---|
| <i>protocol</i> | | String representing the protocol type. Either <code>"http://"</code> or <code>"https://"</code> . |
| <i>ipaddress</i> | | The IP address of the money mobiliser server |
| <i>port</i> | | The port number of the money mobiliser server |
| <i>wsname</i> | | The name of the WS_Core web service; |

Source

SY_Data_Objects.js line 370 on page 67.

Timer class

Timer

Syntax

`new Timer()`

*Source**SY_Data_Objects.js line 357 on page 66.*

Transaction class

Transaction A class that stores information pertaining to certain transaction.

Syntax

```
new Transaction( transactionId, transactionDate, amount, usecase )
```

Parameters

| Name | Type | Description |
|------------------------|------|-------------------------------|
| <i>transactionId</i> | | The id of a transaction |
| <i>transactionDate</i> | | The date of the transaction |
| <i>amount</i> | | The amount of the transaction |
| <i>usecase</i> | | The usecase of transaction |

*Source**SY_Data_Objects.js line 172 on page 60.*

TxnData class

TxnData

Syntax

```
new TxnData( usecase, amount, orderId, text )
```

Parameters

| Name | Type | Description |
|----------------|------|---|
| <i>usecase</i> | | The usecase id for a transaction |
| <i>amount</i> | | The amount of a transaction |
| <i>orderId</i> | | The orderId for a transaction. Holds specific information for certain usecases and txns. |

| | | |
|-------------|--|---|
| <i>text</i> | | Any text which reminds the user of the transaction. |
|-------------|--|---|

Source

SY_Data_Objects.js line 50 on page 56.

Global

AlertNotificationMessage function

Alert notification message mapping class

Syntax

`AlertNotificationMessage(id, alertTypeId, notificationMsgTypeId)`

Parameters

| Name | Type | Description |
|------------------------------|------|-----------------------------|
| <i>id</i> | | of the mapping record |
| <i>alertTypeId</i> | | id of alert type |
| <i>notificationMsgTypeId</i> | | if of notification msg type |

Source

SY_Data_Objects.js line 470 on page 70.

checkOtherIdentificationInSelectedAlert function

finds out if this OI is there in the alert

Syntax

`checkOtherIdentificationInSelectedAlert(oIID)`

Parameters

| Name | Type | Description |
|-------------|------|----------------------|
| <i>oIID</i> | | for an contact point |

Returns

boolean depicting true if exists else false

*Source**SY_Transactions.js line 2097 on page 204.***createNewAlertBack function**

Callback handler for Adding Alert

Syntax`createNewAlertBack()`*Source**SY_Transactions.js line 2118 on page 205.***deleteCustomerAlertBack function**

Call back handler for Deleting customer alert

Syntax`deleteCustomerAlertBack()`*Source**SY_Transactions.js line 2053 on page 202.***findAlertDataInDataList function**

Iterate and find the key in the list and returns the object.

Syntax`findAlertDataInDataList(alertDataList, field)`*Parameters*

| Name | Type | Description |
|----------------------|------|----------------------------|
| <i>alertDataList</i> | | list of alert data objects |
| <i>field</i> | | fieldName to be searched |

Returns

AlertData or null

*Source**SY_Transactions.js line 2203 on page 208.*

getActiveAlertNotificationMessagesBack function

callback handler for the active alert notification message mapping web service call

Syntax

```
getActiveAlertNotificationMessagesBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

Source

SY_Transactions.js line 2350 on page 213.

getAlertDetailsForEditBack function

Callback handler for Get Alert Detail service

Syntax

```
getAlertDetailsForEditBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

Source

SY_Transactions.js line 2137 on page 205.

getAlertNotificationMsgIdBack function

callback handler of getting the alert notification message id back from server

Syntax

```
getAlertNotificationMsgIdBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

*Source**SY_Transactions.js line 2310 on page 212.***getExistingAlertsBack function**

callback handler for receiving existing alerts

Syntax`getExistingAlertsBack()`*Source**SY_Transactions.js line 1973 on page 199.***getTransactionTypesBack function**

Callback handler for getting the transaction types back

Syntax`getTransactionTypesBack(r)`*Parameters*

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

*Source**SY_Transactions.js line 2396 on page 215.***Identification function**

Identification of customer

Syntax`Identification(id, customerId, type, identification)`*Parameters*

| Name | Type | Description |
|-------------------|------|-------------------------------------|
| <i>id</i> | | of the Identification record |
| <i>customerId</i> | | to whom this identification belongs |

| | | |
|-----------------------|--|-------------------|
| <i>type</i> | | of identification |
| <i>identification</i> | | actual value |

Source

SY_Data_Objects.js line 457 on page 70.

listOtherIdentifications function

This function layout the UI for other identifications

Syntax

```
listOtherIdentifications()
```

Source

SY_Transactions.js line 2069 on page 203.

parseAlertNotificationMessages function

Parser for alert notification messages response

Syntax

```
parseAlertNotificationMessages()
```

Returns

the array of AlertNotificationMessage objects

Source

SY_Transactions.js line 2371 on page 214.

parseAlertNotificationMessageTypeId function

parsing logic for alertNotificationMessageTypeId

Syntax

```
parseAlertNotificationMessageTypeId(r)
```

Parameters

| Name | Type | Description |
|----------|------|-----------------------------|
| <i>r</i> | | response data to be parsed. |

Returns

parsed alert notification message type id

*Source**SY_Transactions.js line 2338 on page 213.***parseCustomerAlert function**

parse customer alert data from the response

*Syntax*parseCustomerAlert(*r*)*Parameters*

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

Returns

CustomerAlert object

*Source**SY_Transactions.js line 2223 on page 208.***parseExistingAlerts function**

Parsing the Alerts for Alert List

Syntax

parseExistingAlerts()

*Source**SY_Transactions.js line 1991 on page 199.***parseLookupEntities function**

parser for currencies response

*Syntax*parseLookupEntities(*r*, *filter*) { Array }*Parameters*

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

| | | |
|---------------|--|--|
| <i>filter</i> | | an optional array which maps a new string for name |
|---------------|--|--|

Returns
of parsed currency objects with id and name properties

Type:

Array

Source
SY_Transactions.js line 2422 on page 216.

parseOtherIdentifications function

Parser for other identification web service

Syntax
`parseOtherIdentifications()`

Source
SY_Transactions.js line 2470 on page 217.

updateExistingAlertBack function

Callback handler for Updating Alert

Syntax
`updateExistingAlertBack()`

Source
SY_Transactions.js line 2290 on page 211.

Source code

SY_Data_Objects.js

A set of data objects utilized by the Mobiliser Smartphone application.

```
1  /**
2  * @overview
3  * A set of data objects utilized by the Mobiliser Smartphone
  application.
```

```

4      *
5      * @name SY_Data_Objects.js
6      * @author SAP AG
7      * @version 1.0
8      *
9      */
10
11
12     // TODO: check document against implementation (see
13     parameters)
14
15     /**
16
17     @class Contains basic information about the customer.
18
19     @param customerId
20     @param {string} msisdn The msisdn of the customer
21     @param blacklist An id for blacklistreason to indicate
22     customer account status.
23     @param test test state of customer account.
24     @param typeId The id of customerType
25     @param cancelId The id of cancellationReason of customer
26     account
27     @param modeId The id of mode of txn receipt mode of customer
28     account
29     */
30
31     function Customer() {
32         this.customerId = '';
33         this.orgUnitId = '';
34         this.blacklist = '';
35         this.active = '';
36         this.test = '';
37         this.displayName = '';

```

Money Mobiliser Core API

```
34         this.riskCategoryId = '';
35         this.typeId = '';
36         this.cancelId = '';
37         this.modeId = '';
38     }
39
40     /**
41     @class TxnData
42
43     @constructor
44
45     @param usecase The usecase id for a transaction
46     @param amount The amount of a transaction
47     @param orderId The orderId for a transaction. Holds
specific information for certain usecases and txns.
48     @param text Any text which reminds the user of the
transaction.
49     */
50     function TxnData(usecase, amount, orderId, text) {
51         this.usecase = usecase;
52         this.amount = amount;
53         this.orderId = orderId;
54         this.text = text;
55     }
56
57     /**
58     @class LoginSession This is the session class which stores
all of the user's data throughtout a login session.
59
60     @constructor
61     @param sessionId The session id as returned by the server
62     @param customerId The id of the customer
63     @param displayName The display name of the customer
```


| | | |
|----|----------------------------------|--|
| 64 | @param msisdn | The phone number of the customer |
| 65 | @param transaction | Is an array of transactions |
| 66 | @param contacts | Is an array of contacts from the address book of the user's smartphone |
| 67 | @param billTypes | Is an array of available invoice types |
| 68 | @param registeredbill | A list of all registered bills for the user |
| 69 | @param openbill | The information of an open bill |
| 70 | @param openBillArray | A list of all open and due bills for the user |
| 71 | @param openRequestArray | A list of all open and due demand for payment for the user |
| 72 | @param accounts | A list of money accounts registered by the user |
| 73 | @param offlinesvas | A list of offline SVA accounts registered by the user |
| 74 | @param totalaccounts | total number of money accounts registered by the user |
| 75 | @param customer | particular information of the user |
| 76 | @param svaalert | A list of balance alerts registered for SVA of the user |
| 77 | @param coupons | A list of coupons available for the user |
| 78 | @param couponTypes | A list of coupon types in system |
| 79 | @param categories | A list of coupon categories in one level of category tree |
| 80 | @param category_grandparent | the category node in up two level of current node |
| 81 | @param identification | A list of identification registered by the user |
| 82 | @param customerAlert | particular information of an alert for a user |
| 83 | @param alertNotificationMessages | A list of alert Notification message |
| 84 | @param otherIdentifications | A list of additional identification registered by the user |
| 85 | @param transit_value | A variable to hold the temporary data between pages in navigation |
| 86 | @param image | A handler to hold the image |

Money Mobiliser Core API

```
87      @param uploadUrl The url address in system for uploading a
image
88      @param downloads A list of handlers of images to download
from system
89      @param locations A list of locations in system
90      @param defaultTermAndConditions A list of default term &
conditions in registration
91      @param openTermAndConditions A list of open term &
conditions after user login
92      @param gcmregid The GCM registration ID of this
application
93      */
94      function LoginSession() {
95          this.sessionId = '';
96          this.sessionTimeoutWindow = MAX_SESSION_TIME;
97          this.msisdn = '';
98          this.transaction = [];
99          this.contacts = [];
100         this.billTypes = [];
101         this.registeredbill = [];
102         this.openbill = '';
103         this.openBillArray = [];
104         this.openRequestArray = [];
105         this.accounts = [];
106         this.offlinesvas = [];
107         this.totalaccounts = '';
108         this.customer = '';
109         this.svaalert = [];
110         this.coupons = [];
111         this.couponTypes = [];
112         this.categories = [];
113         this.category_grandparent = '';
114         this.identification = '';
115         this.customerAlert = new CustomerAlert();
```

```

116     this.alertNotificationMessages = [];
117     this.otherIdentifications = [];
118     this.transit_value = '';
119     this.image = '';
120     this.uploadUrl = '';
121     this.downloads = [];
122     this.locations = '';
123     this.defaultTermAndConditions = '';
124     this.openTermAndConditions = '';
125     this.gcmregid = '';
126     this.pibalance = '';
127 }
128
129 /**
130  @class Account A class that stores information pertaining to
131  certain account.
132  @constructor
133  @param pIID The paymentInstrument id of an account
134  @param type The type of an account
135  @param no The number related to an account, such as card no.
136  account no.
137  @param info The extra information of an account, alias name of
138  bank account and credit card. balance of SVA
139  @param acctHolderName The name of bank account/credit card
140  Holder
141  @param acctNumber Bank account number/credit card number
142  @param displayNumber The number to be displayed for bank
143  account/credit card.
144  @param extral The code of bank or The type of the credit card,
145  master, visa or etc.
146  @param extra2 The code of branch that bank account is opened
147  or The security code of the credit card
148  @param extra3 The name of bank or The year of the expiry date
149  for the card

```

Money Mobiliser Core API

```
143     @param extra4 The country of the bank or The month of the
expiry date for the card
144     */
145     function Accounts() {
146         this.walletId = '';
147         this.creditPriority = '';
148         this.debitPriority = '';
149         this.pIID = '';
150         this.pIClass = '';
151         this.type = '';
152         this.no = '';
153         this.info = '';
154         this.acctHolderName = '';
155         this.acctNumber = '';
156         this.displayNumber = '';
157         this.extra1 = '';           //bankCode/cardType
158         this.extra2 = '';           //branchCode/securityNumber
159         this.extra3 = '';           //bankName/yearExpiry
160         this.extra4 = '';           //bankCountry/monthExpiry
161     }
162
163     /**
164     @class Transaction A class that stores information
pertaining to certain transcation.
165
166     @constructor
167     @param transactionId The id of a transaction
168     @param transactionDate The date of the transaction
169     @param amount The amount of the transaction
170     @param usecase The usecase of transaction
171     */
172     function Transaction() {
173         this.transactionId = '';
```

```

174         this.transactionDate = '';
175         this.amount = '';
176         this.usecase = '';
177     }
178
179     /**
180     @class BillTypes is a class that stores information about
181     any invoice that the user has registered for.
182
183     @constructor
184     @param billTypeId The type id for the invoice.
185     @param billTypeName The name of the invoice/bill registered
186     for.
187     */
188     function BillTypes() {
189         this.billTypeId = '';
190         this.billTypeName = '';
191         this.billTypeGroupId= '';
192     }
193
194     /**
195     @class OpenBill A class represnting an open invoice which
196     the user has to pay for a certain merchant.
197
198     @constructor
199     @param billId The id of this invoice/bill
200     @param billTypeName The name of the bill type for this
201     invoice
202     @param dueDate The due date of this bill
203     @param amount The amount due for this bill
204     */
205     function OpenBill() {
206         this.billId = '';
207         this.billTypeName = '';

```

Money Mobiliser Core API

```
204         this.dueDate = '';
205         this.amount = '';
206         this.reference = '';
207         this.billreference = '';
208     }
209
210     /**
211     @class Registeredbill is a class that stores information
about any invoice that the user has registered for.
212
213     @constructor
214     @param billConfigId The id of the bill configuration
215     @param alias The alias name given by the user for this
configuration.
216     @param billTypeId The type id for the invoice.
217     @param billTypeName The name of the invoice/bill registered
for.
218     */
219     function Registeredbill() {
220         this.billConfigId = '';
221         this.alias = '';
222         this.billTypeId = '';
223         this.reference = '';
224         this.billTypeName = '';
225         this.billTypeGroupId = '';
226     }
227
228     /**
229     @class BalanceAlert is a class that stores information about a
Balance Alert.
230
231     @constructor
232     @param billConfigId The id of the bill configuration
```

```
233     @param alias The alias name given by the user for this
configuration.
234     @param billTypeId The type id for the invoice.
235     @param billTypeName The name of the invoice/bill registered
for.
236     */
237     function BalanceAlert() {
238         this.id = '';
239         this.paymentInstrumentId = '';
240         this.threshold = '';
241         this.active = '';
242         this.onlyTransition = '';
243         this.templateName = '';
244     }
245
246     /**
247     @class Contains basic information about a coupon.
248
249     @constructor
250
251     @param id
252     @param customerId
253     @param couponType the type of coupon.
254     @param status the coupon's status
255     @param serialNumber The serial number of the coupon
256     @param validTo valid date of the coupon
257     @param code The code of coupon
258     @param views view times of the coupon
259     @param uses usage times of the coupon
260     */
261     function Coupon() {
262         this.id = '';
263         this.customerId = '';
```

Money Mobiliser Core API

```
264     this.couponType = new CouponType();
265     this.status = '';
266     this.serialNumber = '';
267     this.validTo = '';
268     this.code = '';
269     this.views = '';
270     this.uses = '';
271 }
272
273 /**
274  * @class Contains basic information about a coupon type.
275  *
276  * @constructor
277  *
278  * @param id
279  * @param name
280  * @param purchasePrice
281  * @param purchaseCurrency
282  * @param maxViews
283  * @param maxUses
284  * @param isRestorableByUser
285  * @param description Coupon type description.
286  */
287 function CouponType() {
288     this.id = '';
289     this.name = '';
290     this.purchasePrice = '';
291     this.purchaseCurrency = '';
292     this.maxViews = '';
293     this.maxUses = '';
294     this.isRestorableByUser = '';
295     this.description = new CouponTypeDescription();
```



```
296     }
297
298     /**
299     @class Contains basic information about a coupon type
description.
300
301     @constructor
302
303     @param mimeType
304     @param caption
305     @param content
306     @param thumbnailUrl
307     @param imageUrl
308     */
309     function CouponTypeDescription() {
310         this.mimeType = '';
311         this.caption = '';
312         this.content = '';
313         this.thumbnailUrl = '';
314         this.imageUrl = '';
315     }
316
317     /**
318     @class Contains basic information about a coupon category.
319
320     @constructor
321
322     @param id
323     @param internalName
324     @param caption
325     @param priority
326     @param noOfChildCategories
```

Money Mobiliser Core API

```
327     @param parent
328     */
329     function Category() {
330         this.id = '';
331         this.internalName = '';
332         this.caption = '';
333         this.priority = '';
334         this.noOfChildCategories = '';
335         this.parent = '';
336         this.valid = '';
337     }
338
339     /**
340     @class Contains basic information about geoLocation.
341
342     @constructor
343
344     @param id
345     @param internalName
346     */
347     function GeoLocation() {
348         this.latitude = '';
349         this.longitude = '';
350     }
351
352     /**
353     @class Timer
354
355     @constructor
356     */
357     function Timer() {
358         this.handle = '';
```

```

359         this.on_flag = 0;
360     }
361
362     /**
363         @class Setting is a connection configurations class to
connect to the back-end server.
364         @constructor
365         @param protocol String representing the protocol type.
Either "http://" or "https://".
366         @param ipaddress The IP address of the money mobiliser
server
367         @param port The port number of the money mobiliser server
368         @param wsname The name of the WS_Core web service;
369         */
370     function Setting() {
371         this.protocol = '${ws.protocol}://';
372         this.ipaddress = '${ws.ipaddress}';
373         this.port = '${ws.port}';
374         this.wsname = '${ws.uri}';
375
376         this.origin = "MAPP";
377         this.appinfo = {};
378         this.appinfo.device = '';
379         this.appinfo.deviceId = '';
380         this.appinfo.otherDeviceId = '';
381         this.appinfo.application = "MAPP"; // TODO: change app
name and add version number (svn revision ? build number ?)
382                                     // Currently, appid used
for AuditData expects "MAPP", otherwise user cannot login.
383         this.appinfo.applicationVersion = "5.1.3";
384     }
385
386     /*Alert specific classes*/
387     /**

```

Money Mobiliser Core API

```
388      * @class CustomerAlert which contains alert data set for each
alert
389      * @param customerId to which this alert belongs
390      * @param alertTypeId to which this alert belongs
391      * @param alertNotificationMsgId to which notification type
this alert is tied to
392      * @param alertDataList array of AlertData objects
393      * @param contactPointList array of ContactPoint objects
394      * @param blackoutList array of Blackout objects
395      * @param created
396      */
397      function CustomerAlert() {
398          this.id = '';
399          this.customerId = '';
400          this.alertTypeId = '';
401          this.active = '';
402          this.alertNotificationMsgId = '';
403          this.notifMaxCnt = ''; //For frequency - 0 for Every time,
1 for First time
404          this.notifMaxRecur = ''; //For frequency - D for Day, W
for Week, F - Fortnight, M - Month, Q - Quarter, Y - Year
405          this.alertDataList = [];
406          this.contactPointList = [];
407          this.blackoutList = [];
408          this.created = '';
409      }
410
411
412      /**
413      * @class OtherIdentifications this contains other
identifications for a
414      *      customer
415      * @param customerId to which this Other Identification
belongs
```

```
416     * @param type of identification fax, email, phone etc
417     */
418     function OtherIdentification() {
419         this.id = '';
420         this.customerId = '';
421         this.type = '';
422         this.identification = '';
423         this.identificationType = '';
424         this.nickname = '';
425         this.provider = '';
426         this.status = '';
427         this.active = '';
428     }
429
430     /**
431     * @class ContactPoint holds many to many mapping record
432     between Customer Alert and Other Identification
433     */
434     function ContactPoint() {
435         this.id = '';
436         this.customerAlertId = '';
437         this.identification = new Identification();
438         this.otherIdentification = new OtherIdentification();
439     }
440     /**
441     * @class AlertData this contains key value pair of alert
442     data
443     */
444     function AlertData() {
445         this.id = '';
446         this.customerAlertId = '';
447         this.key = '';
```

Money Mobiliser Core API

```
447         this.value = '';
448     }
449
450     /**
451     * @description Identification of customer
452     * @param id of the Identification record
453     * @param customerId to whom this identification belongs
454     * @param type of identification
455     * @param identification actual value
456     */
457     function Identification() {
458         this.id = '';
459         this.customerId = '';
460         this.type = '';
461         this.identification = '';
462     }
463
464     /**
465     * @description Alert notification message mapping class
466     * @param id of the mapping record
467     * @param alertTypeId id of alert type
468     * @param notificationMsgTypeId if of notification msg type
469     */
470     function AlertNotificationMessage() {
471         this.id = '';
472         this.alertTypeId = '';
473         this.notificationMsgTypeId = '';
474     }
```

SY_Mobiliser.js

A thin JavaScript web service client that accesses the Mobiliser platform.

```
1     /**
```

```

2      * @fileOverview A thin JavaScript web service client that
  accesses the Mobiliser platform.
3      * It provides an abstraction layer to communicate with the
  system and returns
4      * XML documents as a result.
5      *
6      * @name SY_Mobiliser.js
7      * @author SAP AG
8      * @version 1.0
9      *
10     */
11
12     /**
13      * @class 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.
14
15      * @constructor
16
17     */
18     function MobiliserClient() {
19         this.url = setting.protocol + setting.ipaddress;
20
21         if(setting.port)
22             this.url += ":" + setting.port;
23
24         // it's used by file upload/download
25         smartphoneService.urlWithoutWS = this.url;
26
27         if(typeof setting.wsname !== 'undefined')
28             this.url += "/" + setting.wsname;
29
30         smartphoneService.url = this.url;
31         this.network_access = true;

```

Money Mobiliser Core API

```
32     }
33
34     /* web service calls */
35
36     /**
37     @description Agent login function
38     @param responseBack Indicates which function to be called
when a response is received.
39     @param username The username should normally be the msisdn in
addition to its country code i.e. +18881234567
40     @param password The user password
41     @example
42     var loginBack = function(r, xmlResponse) { ... // handle
response };
43     mc.login(loginBack, "user1", "pass2");
44     */
45     MobiliserClient.prototype.login = function(responseBack,
username, password, authtype) {
46     //TODO : optimize and reuse for each call ?
47     var pl = new Object();
48     pl.origin = setting.origin;
49     pl.traceNo = UUIDv4();
50     pl.AuditData = setting.appinfo;
51
52     pl.identification = username;
53     pl.credential = password;
54     if (authtype == '0') {
55         pl.identificationType = 5;
56         pl.credentialType = 1;
57     } else {
58         pl.identificationType = 0;
59         pl.credentialType = 0;
60     }
61
```



```

62     smartphoneService.post(pl, "loginCustomer",
responseBack);
63     };
64
65     /**
66     @description Agent security login function
67     @param responseBack Indicates which function to be called when
a response is received.
68     */
69     MobiliserClient.prototype.fbssologin = function(responseBack,
accessToken) {
70         var pl = new Object();
71         pl.origin = setting.origin;
72         pl.traceNo = UUIDv4();
73         pl.AuditData = setting.appinfo;
74
75         smartphoneService.post(pl, "whoami" + "?
create_mobiliser_remember_me=yes&facebook_code="+accessToken,
responseBack);
76     };
77
78     /**
79     @description Agent request RSA public key from end point
80     @param responseBack Indicates which function to be called when
a response is received.
81     @param type Specifies account type ("bank" or "card")
82     */
83     MobiliserClient.prototype.getRsaPublicKey =
function(responseBack, type) {
84         var pl = new Object();
85
86         pl.origin = setting.origin;
87         pl.traceNo = UUIDv4();
88         pl.AuditData = setting.appinfo;
89         type = (!type || "card" !== type ? "bank" : "card");

```

Money Mobiliser Core API

```
90     session.keyType = type;
91     pl.keyName = ("bank" === type ? "mobiliser_bank" :
92 "mobiliser_card");
93
94     smartphoneService.post(pl, "getRsaPublicKey",
95 responseBack);
96 };
97
98 /**
99     @description Agent logout function
100     @param responseBack Indicates which function to be called
101 when a response is received.
102 */
103 MobiliserClient.prototype.logout = function(responseBack) {
104     var pl = new Object();
105     pl.origin = setting.origin;
106     pl.traceNo = UUIDv4();
107     pl.AuditData = setting.appinfo;
108     pl.sessionId = session.sessionId;
109
110     smartphoneService.post(pl, "logout", responseBack);
111 };
112
113 /**
114     @description Agent create sms token function
115     @param responseBack Indicates which function to be called
116 when a response is received.
117 */
118 MobiliserClient.prototype.createSmsToken =
119 function(responseBack, phoneno) {
120     var pl = new Object();
121     pl.origin = setting.origin;
122     pl.traceNo = UUIDv4();
123     pl.AuditData = setting.appinfo;
```

```
119         pl.msisdn = phoneno;
120         pl.language = "en";
121
122         smartphoneService.user = null;
123         smartphoneService.pass = null;
124         smartphoneService.post(pl, "createSmsToken",
responseBack);
125     };
126
127     /**
128     @description Agent create full customer function
129     @param responseBack Indicates which function to be called
when a response is received.
130     */
131     MobiliserClient.prototype.createFullCustomer =
function(responseBack, reginfo, token) {
132         var pl_customer = new Object();
133         pl_customer.orgUnitId = "0000";
134         pl_customer.blackListReason = "0";
135         pl_customer.active = "true";
136         pl_customer.test = "false";
137         pl_customer.displayName = reginfo.FirstName + " " +
reginfo.LastName;
138         pl_customer.riskCategoryId = "0";
139         pl_customer.customerTypeId = reginfo.CustomerType;
140         pl_customer.cancellationReasonId = "0";
141         pl_customer.txnReceiptModeId = "3";
142         pl_customer.language = "en";
143         pl_customer.country = "US";
144
145         var pl_ident1 = new Object();
146         pl_ident1.type = "0";
147         pl_ident1.identification = reginfo.Phoneno;
148         if (reginfo.CustomerType == "102")
```

Money Mobiliser Core API

```
149         pl_ident1.provider = reginfo.Provider;
150
151         var pl_ident2 = new Object();
152         pl_ident2.type = "5";
153         pl_ident2.identification = reginfo.usrname;
154
155         var pl_cred1 = new Object();
156         pl_cred1.type = "0";
157         pl_cred1.credential = reginfo.Pin;
158
159         var pl_cred2 = new Object();
160         pl_cred2.type = "1";
161         pl_cred2.credential = reginfo.Password;
162
163         var pl_addr = new Object();
164         pl_addr.addressType = "0";
165         pl_addr.addressStatus = "0";
166         pl_addr.firstName = reginfo.FirstName;
167         pl_addr.lastName = reginfo.LastName;
168         pl_addr.email = reginfo.Email;
169
170         var pl = new Object();
171         pl.origin = setting.origin;
172         pl.traceNo = UUIDv4();
173         pl.AuditData = setting.appinfo;
174         pl.customer = pl_customer;
175         pl.identifications = [ pl_ident1, pl_ident2 ];
176         pl.credentials = [ pl_cred1, pl_cred2 ];
177         pl.addresses = [ pl_addr ];
178         pl.smsToken = token;
179         pl.acceptedTncVersionIds =
reginfo.acceptedTncVersionIds;
```

```
180
181     if (reginfo.CustomerType == "102") {
182         var mbankingService = {};
183         jQuery.extend(mbankingService, smartphoneService);
184         mbankingService.url = mc.mbururl();
185         mbankingService.post(pl, "createFullCustomer",
responseBack);
186     } else
187         smartphoneService.post(pl, "createFullCustomer",
responseBack);
188 };
189
190 /**
191     @description Agent update customer function
192     @param responseBack Indicates which function to be called
when a response is received.
193     */
194     MobiliserClient.prototype.updateCustomerNotification =
function(responseBack, mode) {
195         var pl_customer = new Object();
196         pl_customer.id = session.customer.id;
197         pl_customer.orgUnitId = session.customer.orgUnitId;
198         pl_customer.blackListReason =
session.customer.blackListReason;
199         pl_customer.active = session.customer.active;
200         pl_customer.test = session.customer.test;
201         pl_customer.displayName = session.customer.displayName;
202         pl_customer.riskCategoryId =
session.customer.riskCategoryId;
203         pl_customer.customerTypeId =
session.customer.customerTypeId;
204         pl_customer.cancellationReasonId =
session.customer.cancellationReasonId;
205         pl_customer.txnReceiptModeId = mode;
206
```

Money Mobiliser Core API

```
207         var pl = new Object();
208         pl.origin = setting.origin;
209         pl.traceNo = UUIDv4();
210         pl.AuditData = setting.appinfo;
211         pl.customer = pl_customer;
212
213         smartphoneService.post(pl, "updateCustomer",
responseBack);
214     };
215
216     /**
217     @description Agent update customer function
218     @param responseBack Indicates which function to be called
when a response is received.
219     */
220     MobiliserClient.prototype.updateCustomerBlockAccount =
function(responseBack) {
221         var pl_customer = new Object();
222         pl_customer.id = session.customer.id;
223         pl_customer.orgUnitId = session.customer.orgUnitId;
224         pl_customer.blackListReason = "2";
225         pl_customer.active = session.customer.active;
226         pl_customer.test = session.customer.test;
227         pl_customer.displayName = session.customer.displayName;
228         pl_customer.riskCategoryId =
session.customer.riskCategoryId;
229         pl_customer.customerTypeId =
session.customer.customerTypeId;
230         pl_customer.cancellationReasonId =
session.customer.cancellationReasonId;
231         pl_customer.txnReceiptModeId =
session.customer.txnReceiptModeId;
232
233         var pl = new Object();
234         pl.origin = setting.origin;
```

```

235         pl.traceNo = UUIDv4();
236         pl.AuditData = setting.appinfo;
237         pl.customer = pl_customer;
238
239         smartphoneService.post(pl, "updateCustomer",
responseBack);
240     };
241
242     /**
243     @description Agent create identification function
244     @param responseBack Indicates which function to be called
when a response is received.
245     */
246     MobiliserClient.prototype.createIdentification =
function(responseBack, customerId, type, identification) {
247         var pl_ident = new Object();
248         pl_ident.customerId = customerId;
249         pl_ident.type = type;
250         pl_ident.identification = identification;
251
252         var pl = new Object();
253         pl.origin = setting.origin;
254         pl.traceNo = UUIDv4();
255         pl.AuditData = setting.appinfo;
256         pl.identification = pl_ident;
257
258         smartphoneService.post(pl, "createIdentification",
responseBack);
259     };
260
261     /**
262     @description Agent get identifications function
263     @param responseBack Indicates which function to be called
when a response is received.

```

Money Mobiliser Core API

```
264     */
265     MobiliserClient.prototype.getIdentifications =
function(responseBack, customerId, type) {
266         var pl = new Object();
267         pl.origin = setting.origin;
268         pl.traceNo = UUIDv4();
269         pl.AuditData = setting.appinfo;
270         pl.customerId = customerId;
271         pl.identificationTypeId = type;
272
273         smartphoneService.post(pl, "getIdentifications",
responseBack);
274     };
275
276     /**
277     @description setCredential function
278     @param responseBack Indicates which function to be called
when a response is received.
279     @param customerId The customer id of the user
280     @param Credential The PIN or Password to set
281     @param type The type of Credential
282     */
283     MobiliserClient.prototype.setCredential =
function(responseBack, Credential, type) {
284         var pl = new Object();
285         pl.origin = setting.origin;
286         pl.traceNo = UUIDv4();
287         pl.AuditData = setting.appinfo;
288         pl.customerId = session.customer.id;
289         pl.credential = Credential;
290         pl.credentialType = type;
291
292         smartphoneService.post(pl, "setCredential",
responseBack);
```



```

293     };
294
295     /**
296     @description Change Credential function
297     @param responseBack Indicates which function to be called
when a response is received.
298     @param customerId The customer id of the user
299     @param oldCredential The old Credential
300     @param newCredential The new Credential chosen by the user
301     */
302     MobiliserClient.prototype.changeCredential =
function(responseBack, oldCredential, newCredential, type) {
303         var pl = new Object();
304         pl.origin = setting.origin;
305         pl.traceNo = UUIDv4();
306         pl.AuditData = setting.appinfo;
307         pl.customerId = session.customer.id;
308         pl.oldCredential = oldCredential;
309         pl.newCredential = newCredential;
310         pl.credentialType = type;
311
312         smartphoneService.post(pl, "changeCredential",
responseBack);
313     };
314
315     /**
316     @description checkCredential function
317     @param responseBack Indicates which function to be called
when a response is received.
318     @param Credential The PIN or Password to set
319     @param type The type of Credential
320     */
321     MobiliserClient.prototype.checkCredential =
function(responseBack, Credential, type) {

```

Money Mobiliser Core API

```
322     var pl = new Object();
323     pl.origin = setting.origin;
324     pl.traceNo = UUIDv4();
325     pl.AuditData = setting.appinfo;
326     pl.customerId = session.customer.id;
327     pl.credential = Credential;
328     pl.credentialType = type;
329
330     smartphoneService.post(pl, "checkCredential",
responseBack);
331 };
332
333 /**
334     @description A function to query all of the payment
instruments in the customer's mobile wallet
335     @param responseBack Indicates which function to be called
when a response is received.
336     @param customerId The customer id of the user
337     */
338     MobiliserClient.prototype.getWallet = function(responseBack,
customerId) {
339     var pl = new Object();
340     pl.origin = setting.origin;
341     pl.traceNo = UUIDv4();
342     pl.AuditData = setting.appinfo;
343     pl.customerId = customerId;
344
345     smartphoneService.post(pl, "getWalletEntriesByCustomer",
responseBack);
346 };
347
348 /**
349     @description A function to get balance of SVA
```

```

350     @param responseBack Indicates which function to be called
when a response is received.
351     */
352     MobiliserClient.prototype.balanceInquiry =
function(responseBack, piid) {
353         var pl = new Object();
354         pl.origin = setting.origin;
355         pl.traceNo = UUIDv4();
356         pl.AuditData = setting.appinfo;
357         pl.paymentInstrumentId = piid;
358
359         smartphoneService.post(pl, "getPaymentInstrumentBalance",
responseBack);
360     };
361
362     /**
363     @description A function to create a wallet entry with
paymentInstrument in the customer's mobile wallet
364     @param responseBack Indicates which function to be called
when a response is received.
365     */
366     MobiliserClient.prototype.createWalletEntry =
function(responseBack, paymentInstrument,
367         paymentInstrumentType, nickname) {
368         var pl_pI = new Object();
369         pl_pI.pseudo_type = paymentInstrumentType ==
"bankAccount" ? "BankAccount" : "CreditCard";
370         pl_pI.customerId = session.customer.id;
371         pl_pI.active = "true";
372         pl_pI.status = "0";
373         pl_pI.currency = "EUR";
374         pl_pI.multiCurrency = "false";
375         for(var p in paymentInstrument)
376         pl_pI[p] = paymentInstrument[p];
377

```

Money Mobiliser Core API

```
378     var pl_entry = new Object();
379     pl_entry.customerId = session.customer.id;
380     pl_entry.debitPriority = "1";
381     pl_entry.creditPriority = "1";
382     pl_entry.alias = nickname;
383     pl_entry[paymentInstrumentType] = pl_pI;
384
385     var pl = new Object();
386     pl.origin = setting.origin;
387     pl.traceNo = UUIDv4();
388     pl.AuditData = setting.appinfo;
389     pl.walletEntry = pl_entry;
390     pl.primaryDebit = "true";
391     pl.primaryCredit = "true";
392
393     smartphoneService.post(pl, "createWalletEntry",
responseBack);
394     };
395
396     /**
397     @description A function to update a wallet entry in the
customer's mobile wallet
398     @param responseBack Indicates which function to be called
when a response is received.
399     */
400     MobiliserClient.prototype.updateWalletEntry =
function(responseBack, nickname, acctid) {
401     var pl_entry = new Object();
402     pl_entry.id = session.accounts[acctid].walletId;
403     pl_entry.customerId = session.customer.id;
404     pl_entry.paymentInstrumentId =
session.accounts[acctid].pIID;
405     pl_entry.alias = nickname;
406
```

```

407     var pl = new Object();
408     pl.origin = setting.origin;
409     pl.traceNo = UUIDv4();
410     pl.AuditData = setting.appinfo;
411     pl.walletEntry = pl_entry;
412
413     smartphoneService.post(pl, "updateWalletEntry",
responseBack);
414 };
415
416 /**
417     @description A function to set primary wallet in the
customer's mobile wallet
418     @param responseBack Indicates which function to be called
when a response is received.
419     */
420     MobiliserClient.prototype.setPrimary =
function(responseBack, acctid) {
421     var pl_entry = new Object();
422     pl_entry.id = session.accounts[acctid].walletId;
423     pl_entry.customerId = session.customer.id;
424     pl_entry.paymentInstrumentId =
session.accounts[acctid].pIID;
425     pl_entry.alias = session.accounts[acctid].info;
426
427     var pl = new Object();
428     pl.origin = setting.origin;
429     pl.traceNo = UUIDv4();
430     pl.AuditData = setting.appinfo;
431     pl.walletEntry = pl_entry;
432     pl.primaryDebit = "true";
433     pl.primaryCredit = "true";
434

```

Money Mobiliser Core API

```
435     smartphoneService.post(pl, "updateWalletEntry",
responseBack);
436     };
437
438     /**
439     @description A function to update a paymentInstrument of a
wallet entry in the customer's mobile wallet
440     @param responseBack Indicates which function to be called
when a response is received.
441     */
442     MobiliserClient.prototype.updatePaymentInstrument =
function(responseBack, paymentInstrument,
443         paymentInstrumentType, acctid) {
444         var pl_pI = new Object();
445         pl_pI.pseudo_type = paymentInstrumentType;
446         pl_pI.id = session.accounts[acctid].pIID;
447         pl_pI.customerId = session.customer.id;
448         pl_pI.active = "true";
449         pl_pI.status = "0";
450         pl_pI.currency = "EUR";
451         pl_pI.multiCurrency = "false";
452         for(var p in paymentInstrument)
453             pl_pI[p] = paymentInstrument[p];
454
455         var pl = new Object();
456         pl.origin = setting.origin;
457         pl.traceNo = UUIDv4();
458         pl.AuditData = setting.appinfo;
459         pl.PaymentInstrument = pl_pI;
460
461         smartphoneService.post(pl, "updatePaymentInstrument",
responseBack);
462     };
463
```

```

464     /**
465     @description A function to create a wallet entry with
paymentInstrument in the customer's mobile wallet
466     @param responseBack Indicates which function to be called
when a response is received.
467     */
468     MobiliserClient.prototype.deleteWalletEntry =
function(responseBack, acctid) {
469         var pl = new Object();
470         pl.origin = setting.origin;
471         pl.traceNo = UUIDv4();
472         pl.AuditData = setting.appinfo;
473         pl.walletEntryId = session.accounts[acctid].walletId;
474
475         session.accountPiIdForDeletion =
session.accounts[acctid].piId;
476         smartphoneService.post(pl, "deleteWalletEntry",
responseBack);
477     };
478
479     /**
480     @description Get transaction history for a customer
481     @param responseBack Indicates which function to be called
when a response is received.
482     @param customerId The customer id of the user
483     @param maxRecords The max number of transactions to return
484     @param paymentInstrumentId The id of the payment
instrument
485     */
486     MobiliserClient.prototype.findTransactions =
function(responseBack, customerId, maxRecords) {
487         var pl = new Object();
488         pl.origin = setting.origin;
489         pl.traceNo = UUIDv4();
490         pl.AuditData = setting.appinfo;

```

Money Mobiliser Core API

```
491         pl.customerId = customerId;
492         pl.statusFilter = "30";
493         pl.maxRecords = maxRecords;
494
495         smartphoneService.post(pl, "findTransactions",
responseBack);
496     };
497
498     /**
499     @description Get details of a transaction
500     @param responseBack Indicates which function to be called
when a response is received.
501     @param customerId The customer id of the user
502     @param maxRecords The max number of transactions to return
503     @param paymentInstrumentId The id of the payment instrument
504     */
505     MobiliserClient.prototype.getTxnDetails =
function(responseBack, id) {
506         var pl = new Object();
507         pl.origin = setting.origin;
508         pl.traceNo = UUIDv4();
509         pl.AuditData = setting.appinfo;
510         pl.txnId = id;
511
512         smartphoneService.post(pl, "getTransactionDetails",
responseBack);
513     };
514
515     /**
516     @description Get types of invoices by group in the system
517     @param responseBack Indicates which function to be called
when a response is received.
518     */
```



```

519     MobiliserClient.prototype.getInvoiceTypesByGroup =
function(responseBack) {
520         var pl = new Object();
521         pl.origin = setting.origin;
522         pl.traceNo = UUIDv4();
523         pl.AuditData = setting.appinfo;
524         pl.groupId = "2";
525         pl.onlyActive = "true";
526
527         smartphoneService.post(pl, "getInvoiceTypesByGroup",
responseBack);
528     };
529
530     /**
531     @description Get types of invoices by group in the system
532     @param responseBack Indicates which function to be called
when a response is received.
533     */
534     MobiliserClient.prototype.createInvoiceForInvoiceType =
function(responseBack, invoiceTypeId,
535         reference, amount ) {
536         var pl = new Object();
537         pl.origin = setting.origin;
538         pl.traceNo = UUIDv4();
539         pl.AuditData = setting.appinfo;
540         pl.invoiceTypeId = invoiceTypeId;
541         pl.customerId = session.customer.id;
542         pl.reference = reference;
543         pl.amount = amount;
544         pl.currency = "EUR";
545
546         smartphoneService.post(pl, "createInvoiceForInvoiceType",
responseBack);
547     };

```

Money Mobiliser Core API

```
548
549     /**
550     @description Get all types of invoices in the system
551     @param responseBack Indicates which function to be called
when a response is received.
552     */
553     MobiliserClient.prototype.getBillTypes =
function(responseBack) {
554         var pl = new Object();
555         pl.origin = setting.origin;
556         pl.traceNo = UUIDv4();
557         pl.AuditData = setting.appinfo;
558         pl.onlyActive = "true";
559
560         smartphoneService.post(pl, "getAllInvoiceTypes",
responseBack);
561     };
562
563     /**
564     @description Get all configured invoices for a customer
565     @param responseBack Indicates which function to be called
when a response is received.
566     @param customerId The customer id of the user
567     */
568     MobiliserClient.prototype.getRegisteredBills =
function(responseBack, customerId) {
569         var pl = new Object();
570         pl.origin = setting.origin;
571         pl.traceNo = UUIDv4();
572         pl.AuditData = setting.appinfo;
573         pl.customerId = customerId;
574         pl.onlyActive = "true";
575
```

```

576     smartphoneService.post(pl,
"getInvoiceConfigurationsByCustomer", responseBack);
577     };
578
579     /**
580     @description Configure an invoice for some merchant bill type
for a customer
581     @param responseBack Indicates which function to be called
when a response is received.
582     @param customerId The customer id of the user
583     @param alias The name the customer gives for this invoice
configuration
584     @param typeId The id of the invoice type
585     */
586     MobiliserClient.prototype.registerSimpleBill =
function(responseBack, customerId, typeId, ref, alias) {
587         var pl_invconfig = new Object();
588         pl_invconfig.invoiceTypeId = typeId;
589         pl_invconfig.customerId = customerId;
590         pl_invconfig.reference = ref;
591         pl_invconfig.status = "0";
592         pl_invconfig.active = "true";
593         pl_invconfig.alias = alias;
594
595
596         var pl = new Object();
597         pl.origin = setting.origin;
598         pl.traceNo = UUIDv4();
599         pl.AuditData = setting.appinfo;
600         pl.invoiceConfiguration = pl_invconfig;
601
602         smartphoneService.post(pl, "createInvoiceConfiguration",
responseBack);
603     };

```

Money Mobiliser Core API

```
604
605     /**
606     @description Remove a configured invoice for a customer
607     @param responseBack Indicates which function to be called
when a response is received.
608     @param invoiceConfigurationId The id of an invoice
configuration for a customer
609     */
610     MobiliserClient.prototype.unregisterBill =
function(responseBack, invoiceConfigurationId) {
611         var pl = new Object();
612         pl.origin = setting.origin;
613         pl.traceNo = UUIDv4();
614         pl.AuditData = setting.appinfo;
615         pl.invoiceConfigurationId = invoiceConfigurationId;
616
617         smartphoneService.post(pl, "deleteInvoiceConfiguration",
responseBack);
618     };
619
620     /**
621     @description Get all active invoices for a customer
622     @param responseBack Indicates which function to be called
when a response is received.
623     @param customerId The customer id of the user
624     */
625     MobiliserClient.prototype.getOpenInvoices =
function(responseBack, customerId) {
626         var pl = new Object();
627         pl.origin = setting.origin;
628         pl.traceNo = UUIDv4();
629         pl.AuditData = setting.appinfo;
630         pl.customerId = customerId;
631         pl.invoiceStatus = "1";
```

```
632
633     smartphoneService.post(pl, "getInvoicesByCustomer",
responseBack);
634     };
635
636     /**
637     @description Create an invoice for a customer for a specific
type of merchant bill
638     @param responseBack Indicates which function to be called
when a response is received.
639     @param invoiceConfigurationId The id of an invoice
configuration for a customer
640     @param ref The reference number of an invoice
641     @param amount The amount of money to pay in cents
642     @param date
643     */
644     MobiliserClient.prototype.createInvoice =
function(responseBack, invoiceConfigurationId, amount, date) {
645         var inv = new Object();
646         inv.invoiceConfigurationId = invoiceConfigurationId;
647         inv.status = "1";
648         inv.amount = amount;
649         inv.currency = "EUR";
650         inv.date = date+"T00:00:00.000Z";
651
652         var pl = new Object();
653         pl.origin = setting.origin;
654         pl.traceNo = UUIDv4();
655         pl.AuditData = setting.appinfo;;
656         pl.invoice = inv;
657
658         smartphoneService.post(pl, "createInvoice",
responseBack);
659     };
```

Money Mobiliser Core API

```
660
661  /**
662    @description transfer function
663    @param responseBack Indicates which function to be called
when a response is received.
664    @param payercustomerId The customerId of payer
665    @param payerpI The paymentInstrumentId of payer
666    @param payeemsisdn The msisdn of payee receiving the
payment
667    @param txn The TxnData object that contains txn details
668  */
669  MobiliserClient.prototype.transfer = function(responseBack,
payercustomerId, payeemsisdn, txn) {
670    var payerident = new Object();
671    payerident.type = "1";
672    payerident.value = payercustomerId;
673
674    var pl_payer = new Object();
675    pl_payer.identifier = payerident;
676
677    var payeeident = new Object();
678    payeeident.type = "0";
679    payeeident.value = payeemsisdn;
680
681    var pl_payee = new Object();
682    pl_payee.identifier = payeeident;
683
684    var pl_amount = new Object();
685    pl_amount.currency = "EUR";
686    pl_amount.vat = "0";
687    pl_amount.value = txn.amount;
688
689    var pl = new Object();
```

```

690         pl.origin = setting.origin;
691         pl.traceNo = UUIDv4();
692         pl.AuditData = setting.appinfo;
693         pl.autoCapture = "true";
694         pl.orderChannel = "0";
695         pl.usecase = txn.usecase;
696         pl.Payer = pl_payer;
697         pl.Payee = pl_payee;
698         pl.Amount = pl_amount;
699         pl.Text = txn.text;
700
701         smartphoneService.post(pl, "preAuthorisation",
responseBack);
702     };
703
704     /**
705     @description preAuthorisationContinue function
706     @param responseBack Indicates which function to be called
when a response is received.
707     @param id The systemId of previous PreAuthorisation
708     @param ref The Reference of previous PreAuthorisation
709     */
710     MobiliserClient.prototype.preAuthorisationContinue =
function(responseBack, id) {
711         var pl_ref = new Object();
712         pl_ref.systemId = id;
713
714         var pl = new Object();
715         pl.origin = setting.origin;
716         pl.traceNo = UUIDv4();
717         pl.AuditData = setting.appinfo;
718         pl.ReferenceTransaction = pl_ref;
719

```

Money Mobiliser Core API

```
720     smartphoneService.post(pl, "preAuthorisationContinue",
721     responseBack);
722
723     /**
724     @description request function
725     @param responseBack Indicates which function to be called
726     when a response is received.
727     @param payermsisdn The Customer msisdn that will receive the
728     request
729     @param txn The TxnData object that contains txn details
730     */
731     MobiliserClient.prototype.request = function(responseBack,
732     payermsisdn, txn) {
733
734         var payerident = new Object();
735         payerident.type = "0";
736         payerident.value = payermsisdn;
737
738         var pl_payer = new Object();
739         pl_payer.identifier = payerident;
740
741         var payeeident = new Object();
742         payeeident.type = "1";
743         payeeident.value = session.customer.id;
744
745         var pl_payee = new Object();
746         pl_payee.identifier = payeeident;
747
748         var pl_amount = new Object();
749         pl_amount.currency = "EUR";
750         pl_amount.vat = "0";
751         pl_amount.value = txn.amount;
752
753         var pl = new Object();
```



```

750         pl.origin = setting.origin;
751         pl.traceNo = UUIDv4();
752         pl.AuditData = setting.appinfo;
753         pl.autoCapture = "true";
754         pl.orderChannel = "0";
755         pl.usecase = txn.usecase;
756         pl.Payer = pl_payer;
757         pl.Payee = pl_payee;
758         pl.Amount = pl_amount;
759         pl.Text = txn.text;
760
761         smartphoneService.post(pl, "preAuthorisation",
responseBack);
762     };
763
764     /**
765     @description startVoucher function
766     @param responseBack Indicates which function to be called
when a response is received.
767     @param payercustomerId The Customer customerId which will
make the payment
768     @param payerpI The paymentInstrument Id of the Customer which
will use to make the payment
769     @param payeemsisdn The Customer msisdn that will receive the
payment
770     @param txn The TxnData object that contains txn details
771     */
772     MobiliserClient.prototype.startVoucher =
function(responseBack, payercustomerId, payeemsisdn, txn) {
773         var payerident = new Object();
774         payerident.type = "1";
775         payerident.value = payercustomerId;
776
777         var pl_payer = new Object();

```

Money Mobiliser Core API

```
778         pl_payer.identifier = payerident;
779
780         var payeeident = new Object();
781         payeeident.type = "0";
782         payeeident.value = payeemsisdn;
783
784         var pl_payee = new Object();
785         pl_payee.identifier = payeeident;
786
787         var pl_amount = new Object();
788         pl_amount.currency = "EUR";
789         pl_amount.vat = "0";
790         pl_amount.value = txn.amount;
791
792         var pl = new Object();
793         pl.origin = setting.origin;
794         pl.traceNo = UUIDv4();
795         pl.AuditData = setting.appinfo;
796         pl.autoCapture = "true";
797         pl.orderChannel = "0";
798         pl.usecase = txn.usecase;
799         pl.Payer = pl_payer;
800         pl.Payee = pl_payee;
801         pl.Amount = pl_amount;
802         pl.Text = txn.text;
803
804         smartphoneService.post(pl, "startVoucher",
responseBack);
805     };
806
807     /**
808     @description confirmVoucher function
```

```

809     @param responseBack Indicates which function to be called
when a response is received.
810     @param id The systemId of previous StartVoucher
811     @param ref The Reference of previous StartVoucher
812     */
813     MobiliserClient.prototype.confirmVoucher =
function(responseBack, id, ref) {
814         var pl_ref = new Object();
815         pl_ref.systemId = id;
816         pl_ref.value = ref;
817
818         var pl = new Object();
819         pl.origin = setting.origin;
820         pl.traceNo = UUIDv4();
821         pl.AuditData = setting.appinfo;
822         pl.ReferenceTransaction = pl_ref;
823
824         smartphoneService.post(pl, "confirmvoucher",
responseBack);
825     };
826
827     /**
828     @description DemandOnPayment function
829     @param responseBack Indicates which function to be called
when a response is received.
830     @param username The username should normally be the msisdn in
addition to its country code i.e. +18881234567
831     @param password The user password
832     @param payer The Customer object which will be making the
payment
833     @param payee The Customer object that will be receiving the
payment
834     @param txn The TxnData object that contains txn details
835     */

```

Money Mobiliser Core API

```
836 MobiliserClient.prototype.demandForPayment =
function(responseBack, payermsisdn, payeecustomerId, amount, msg) {
837     var payerident = new Object();
838     payerident.type = "0";
839     payerident.value = payermsisdn;
840
841     var pl_payer = new Object();
842     pl_payer.identifier = payerident;
843
844     var payeeident = new Object();
845     payeeident.type = "1";
846     payeeident.value = payeecustomerId;
847
848     var pl_payee = new Object();
849     pl_payee.identifier = payeeident;
850
851     var pl_amount = new Object();
852     pl_amount.currency = "EUR";
853     pl_amount.vat = "0";
854     pl_amount.value = amount;
855
856     var pl = new Object();
857     pl.origin = setting.origin;
858     pl.traceNo = UUIDv4();
859     pl.AuditData = setting.appinfo;
860     pl.Payer = pl_payer;
861     pl.Payee = pl_payee;
862     pl.Amount = pl_amount;
863     pl.Text = msg;
864
865     smartphoneService.post(pl, "demandForPayment",
responseBack);
866     };
```

```
867
868     /**
869     @description load function
870     @param responseBack Indicates which function to be called
when a response is received.
871     @param payerpI The paymentInstrument Id of the Customer which
will use to load fund to SVA
872     @param txn The TxnData object that contains txn details
873     */
874     MobiliserClient.prototype.load = function(responseBack,
payer, payee, txn) {
875         var pl_amount = new Object();
876         pl_amount.currency = "EUR";
877         pl_amount.vat = "0";
878         pl_amount.value = txn.amount;
879
880         var pl = new Object();
881         pl.origin = setting.origin;
882         pl.traceNo = UUIDv4();
883         pl.AuditData = setting.appinfo;
884         pl.autoCapture = "true";
885         pl.orderChannel = "0";
886         pl.usecase = txn.usecase;
887         pl.Payer = payer;
888         pl.Payee = payee;
889         pl.Amount = pl_amount;
890         pl.Text = txn.text;
891
892         smartphoneService.post(pl, "preAuthorisation",
responseBack);
893     };
894
895     /**
896     @description unload function
```

Money Mobiliser Core API

```
897     @param responseBack Indicates which function to be called
when a response is received.
898     @param payeeId The paymentInstrument Id of the Customer which
will use to receive fund from SVA
899     @param txn The TxnData object that contains txn details
900     */
901     MobiliserClient.prototype.unload = function(responseBack,
payer, payee, txn, autocap) {
902         var pl_amount = new Object();
903         pl_amount.currency = "EUR";
904         pl_amount.vat = "0";
905         pl_amount.value = txn.amount;
906
907         var pl = new Object();
908         pl.origin = setting.origin;
909         pl.traceNo = UUIDv4();
910         pl.AuditData = setting.appinfo;
911         pl.autoCapture = autocap;
912         pl.orderChannel = "0";
913         pl.usecase = txn.usecase;
914         pl.Payer = payer;
915         pl.Payee = payee;
916         pl.Amount = pl_amount;
917         pl.Text = txn.text;
918
919         smartphoneService.post(pl, "preAuthorisation",
responseBack);
920     };
921
922     /**
923     @description createBalanceAlert function
924     @param responseBack Indicates which function to be called
when a response is received.
925     */
```

```

926     MobiliserClient.prototype.createBalanceAlert =
function(responseBack, threshold, onlyTransition) {
927         var pl_alert = new Object();
928         pl_alert.paymentInstrumentId =
session.accounts[0].pIID;
929         pl_alert.threshold = threshold;
930         pl_alert.active = "true";
931         pl_alert.onlyTransition = onlyTransition;
932         pl_alert.templateName = "sva balance alert";
933
934         var pl = new Object();
935         pl.origin = setting.origin;
936         pl.traceNo = UUIDv4();
937         pl.AuditData = setting.appinfo;
938         pl.BalanceAlert = pl_alert;
939
940         smartphoneService.post(pl, "createBalanceAlert",
responseBack);
941     };
942
943     /**
944     @description getBalanceAlert function
945     @param responseBack Indicates which function to be called
when a response is received.
946     */
947     MobiliserClient.prototype.getBalanceAlert =
function(responseBack) {
948         var pl = new Object();
949         pl.origin = setting.origin;
950         pl.traceNo = UUIDv4();
951         pl.AuditData = setting.appinfo;
952         pl.paymentInstrumentId = session.accounts[0].pIID;
953         pl.includeInactive = "false";
954

```

Money Mobiliser Core API

```
955     smartphoneService.post(pl,
956     "getBalanceAlertByPaymentInstrument", responseBack);
957
958     /**
959     @description updateBalanceAlert function
960     @param responseBack Indicates which function to be called
961     when a response is received.
962     */
963     MobiliserClient.prototype.updateBalanceAlert =
964     function(responseBack, threshold, onlyTransition, alertid) {
965         var pl_alert = new Object();
966         pl_alert.id = session.svaalert[alertid].id;
967         pl_alert.paymentInstrumentId =
968         session.accounts[0].pIID;
969         pl_alert.threshold = threshold;
970         pl_alert.active = "true";
971         pl_alert.onlyTransition = onlyTransition;
972         pl_alert.templateName = "sva balance alert";
973
974         var pl = new Object();
975         pl.origin = setting.origin;
976         pl.traceNo = UUIDv4();
977         pl.AuditData = setting.appinfo;
978         pl.BalanceAlert = pl_alert;
979
980         smartphoneService.post(pl, "updateBalanceAlert",
981         responseBack);
982     };
983
984     /**
985     @description deleteBalanceAlert function
986     @param responseBack Indicates which function to be called
987     when a response is received.
```



```
983     */
984     MobiliserClient.prototype.deleteBalanceAlert =
function(responseBack, alertid) {
985         var pl = new Object();
986         pl.origin = setting.origin;
987         pl.traceNo = UUIDv4();
988         pl.AuditData = setting.appinfo;
989         pl.balanceAlertId = session.svaalert[alertid].id;
990
991         smartphoneService.post(pl, "deleteBalanceAlert",
responseBack);
992     };
993
994     /**
995     @description Top up function
996     @param responseBack Indicates which function to be called
when a response is received.
997     @param invoiceId The id of an invoice
998     @param payerPaymentInstrumentId The payment instrument id
for the payer
999     */
1000    MobiliserClient.prototype.topUp = function(responseBack,
invoiceId) {
1001        var pl = new Object();
1002        pl.origin = setting.origin;
1003        pl.traceNo = UUIDv4();
1004        pl.AuditData = setting.appinfo;
1005        pl.invoiceId = invoiceId;
1006
1007        smartphoneService.post(pl, "checkInvoice",
responseBack);
1008    };
1009
1010    /**
```

Money Mobiliser Core API

```
1011     @description Pay bill function
1012     @param responseBack Indicates which function to be called
when a response is received.
1013     @param invoiceId The id of an invoice
1014     @param payerPaymentInstrumentId The payment instrument id
for the payer
1015     */
1016     MobiliserClient.prototype.payBill = function(responseBack,
invoiceId, payerPaymentInstrumentId) {
1017         var pl = new Object();
1018         pl.origin = setting.origin;
1019         pl.traceNo = UUIDv4();
1020         pl.AuditData = setting.appinfo;
1021         pl.invoiceId = invoiceId;
1022         pl.payerPaymentInstrumentId = payerPaymentInstrumentId;
1023
1024         smartphoneService.post(pl, "checkInvoice",
responseBack);
1025     };
1026
1027     /**
1028     @description ContinuePayInvoice function
1029     @param responseBack Indicates which function to be called
when a response is received.
1030     @param id The system id of checkPayInvoice transaction
1031     @param ref The reference of checkPayInvoice transaction
1032     */
1033     MobiliserClient.prototype.continuePayBill =
function(responseBack, id, ref) {
1034         var pl_ref = new Object();
1035         pl_ref.systemId = id;
1036         pl_ref.value = ref;
1037
1038         var pl = new Object();
```

```
1039     pl.origin = setting.origin;
1040     pl.traceNo = UUIDv4();
1041     pl.AuditData = setting.appinfo;
1042     pl.ReferenceTransaction = pl_ref;
1043
1044     smartphoneService.post(pl, "continueInvoice",
responseBack);
1045 };
1046
1047 /**
1048  @description Cancel bill function
1049  @param responseBack Indicates which function to be called
when a response is received.
1050  @param invoiceId The id of an invoice
1051  */
1052  MobiliserClient.prototype.cancelBill =
function(responseBack, invoiceId) {
1053     var pl = new Object();
1054     pl.origin = setting.origin;
1055     pl.traceNo = UUIDv4();
1056     pl.AuditData = setting.appinfo;
1057     pl.invoiceId = invoiceId;
1058     pl.cancelByCustomer = "false";
1059
1060     smartphoneService.post(pl, "cancelInvoice",
responseBack);
1061 };
1062
1063
1064  //Coupon Feature
1065
1066  //Assign Coupon
1067  /**
1068  @description AssignCoupon function
```

Money Mobiliser Core API

```
1069     @param responseBack Indicates which function to be called
when a response is received.
1070     @param couponTypeId The id of an coupon
1071     */
1072     MobiliserClient.prototype.assignCoupon =
function(responseBack, couponTypeId) {
1073         var pl = new Object();
1074         pl.origin = setting.origin;
1075         pl.traceNo = UUIDv4();
1076         pl.AuditData = setting.appinfo;
1077         pl.couponTypeId = couponTypeId;
1078
1079         smartphoneService.post(pl, "assignCoupon",
responseBack);
1080     };
1081
1082     //Find Coupon Types By Tags
1083     /**
1084     @description FindCouponTypesByTags function
1085     @param responseBack Indicates which function to be called
when a response is received.
1086     @param tag The tag of the coupon
1087     @param locale The location of the coupon
1088     @param mimeType The mimetype of the coupon
1089     */
1090     MobiliserClient.prototype.findCouponTypesByTags =
function(responseBack, tags) {
1091         var tagarray = tags.split(" ");
1092         var pl = new Object();
1093         pl.origin = setting.origin;
1094         pl.traceNo = UUIDv4();
1095         pl.AuditData = setting.appinfo;
1096         pl.tag = tagarray;
1097         pl.locale = locale;
```

```
1098     pl.mimeType = mimeType;
1099     if (geoLocation != null) {
1100         pl.location = new Object();
1101         pl.location.latitude = geoLocation.latitude;
1102         pl.location.longitude = geoLocation.longitude;
1103     }
1104
1105     smartphoneService.post(pl, "findCouponTypesByTags",
responseBack);
1106 };
1107
1108 //Get Child Categories
1109 /**
1110  @description GetChildCategories function
1111  @param responseBack Indicates which function to be called
when a response is received.
1112  @param parentCategoryId The parent id of the coupon
1113  @param locale The location of the coupon
1114  */
1115  MobiliserClient.prototype.getChildCategories =
function(responseBack, parentCategoryId) {
1116     var pl = new Object();
1117     pl.origin = setting.origin;
1118     pl.traceNo = UUIDv4();
1119     pl.AuditData = setting.appinfo;
1120     pl.locale = locale;
1121     pl.parentCategoryId = parentCategoryId;
1122
1123     smartphoneService.post(pl, "getChildCategories",
responseBack);
1124 };
1125
1126 //Get Coupon Types For Category
1127 /**
```

Money Mobiliser Core API

```
1128     @description getCouponTypesForCategory function
1129     @param responseBack Indicates which function to be called
when a response is received.
1130     @param categoryId The category of the coupon
1131     @param locale The location of the coupon
1132     @param mimeType The mimetype of the coupon
1133     */
1134     MobiliserClient.prototype.getCouponTypesForCategory =
function(responseBack, categoryId) {
1135         var pl = new Object();
1136         pl.origin = setting.origin;
1137         pl.traceNo = UUIDv4();
1138         pl.AuditData = setting.appinfo;
1139         pl.categoryId = categoryId;
1140         pl.locale = locale;
1141         pl.mimeType = mimeType;
1142         if (geoLocation != null) {
1143             pl.location = new Object();
1144             pl.location.latitude = geoLocation.latitude;
1145             pl.location.longitude = geoLocation.longitude;
1146         }
1147
1148         smartphoneService.post(pl, "getCouponTypesForCategory",
responseBack);
1149     };
1150
1151     //Get My Coupons
1152     /**
1153     @description GetMyCoupons function
1154     @param responseBack Indicates which function to be called
when a response is received.
1155     @param locale The location of the coupon
1156     @param mimeType The mimetype of the coupon
1157     */
```

```

1158     MobiliserClient.prototype.getMyCoupons =
function(responseBack) {
1159         var pl = new Object();
1160         pl.origin = setting.origin;
1161         pl.traceNo = UUIDv4();
1162         pl.AuditData = setting.appinfo;
1163         pl.locale = locale;
1164         pl.mimeType = mimeType;
1165
1166         smartphoneService.post(pl, "getMyCoupons",
responseBack);
1167     };
1168
1169     //Get Root Categories
1170     /**
1171     @description GetRootCategories function
1172     @param responseBack Indicates which function to be called
when a response is received.
1173     @param locale The location of the coupon
1174     @param groupId The group id of the coupon
1175     */
1176     MobiliserClient.prototype.getRootCategories =
function(responseBack, groupId) {
1177         var pl = new Object();
1178         pl.origin = setting.origin;
1179         pl.traceNo = UUIDv4();
1180         pl.AuditData = setting.appinfo;
1181         pl.groupId = groupId;
1182         pl.locale = locale;
1183
1184         smartphoneService.post(pl, "getRootCategories",
responseBack);
1185     };
1186

```

Money Mobiliser Core API

```
1187     /**
1188     @description GetCategoryTree function
1189     @param responseBack Indicates which function to be called
when a response is received.
1190     @param locale The location of the coupon
1191     @param groupId The group id of the coupon
1192     */
1193     MobiliserClient.prototype.getCategoryTree =
function(responseBack, groupId) {
1194         var pl = new Object();
1195         pl.origin = setting.origin;
1196         pl.traceNo = UUIDv4();
1197         pl.AuditData = setting.appinfo;
1198         pl.groupId = groupId;
1199         pl.locale = locale;
1200
1201         smartphoneService.post(pl, "getCategoryTree",
responseBack);
1202     };
1203
1204     /**
1205     @description PurchaseCoupon function
1206     @param responseBack Indicates which function to be called
when a response is received.
1207     @param paymentInstrumentId The instrument id of an coupon
1208     */
1209     MobiliserClient.prototype.purchaseCoupon =
function(responseBack, couponInstanceId, paymentInstrumentId) {
1210         var pl = new Object();
1211         pl.origin = setting.origin;
1212         pl.traceNo = UUIDv4();
1213         pl.AuditData = setting.appinfo;
1214         pl.couponInstanceId = couponInstanceId;
1215         pl.paymentInstrumentId = paymentInstrumentId;
```



```
1216
1217     smartphoneService.post(pl, "purchaseCoupon",
1218     responseBack);
1219 };
1220 /**
1221     @description DeleteCoupon function
1222     @param responseBack Indicates which function to be called
1223     when a response is received.
1224     @param couponId The id of an coupon
1225     */
1226     MobiliserClient.prototype.deleteCoupon =
1227     function(responseBack, couponId) {
1228         var pl = new Object();
1229         pl.origin = setting.origin;
1230         pl.traceNo = UUIDv4();
1231         pl.AuditData = setting.appinfo;
1232         pl.couponInstanceId = couponId;
1233         smartphoneService.post(pl, "deleteCoupon",
1234         responseBack);
1235     };
1236 /**
1237     @description get Existing Alerts function
1238     @param responseBack Indicate which function to be called when
1239     a response is received.
1240     */
1241     MobiliserClient.prototype.getExistingAlerts =
1242     function(responseBack) {
1243         var pl = new Object();
1244         pl.origin = setting.origin;
1245         pl.customerId = session.customer.id;
1246         pl.traceNo = UUIDv4();
```

Money Mobiliser Core API

```
1244         pl.AuditData = setting.appinfo;
1245         smartphoneService.post(pl, "findCustomerAlertByCustomer",
responseBack);
1246     };
1247
1248     /**
1249     * @description create a new alert for customer
1250     * @param responseBack Indicates which function to be called
when a successful response is received.
1251     * @param alertTypeId Type of alert to be created
1252     * @param alertDataListItems alert data item objects list
1253     * @param contactPointsItems contact point objects list
1254     * @param frequencyVal Value of frequency Everytime or First
time
1255     * @param alertNotificationMsgId text or conv
1256     */
1257     MobiliserClient.prototype.createNewAlert =
function(responseBack, alertTypeId, alertDataListItems,
contactPointsItems, frequencyVal, alertNotificationMsdId) {
1258         var alertDataListArray = [];
1259         for(var i = 0;i < alertDataListItems.length;i++) {
1260             var ad = new Object();
1261             ad.key = alertDataListItems[i].key;
1262             ad.value = alertDataListItems[i].value;
1263             alertDataListArray.push(ad);
1264         }
1265
1266         var contactPointsArray = [];
1267         for(var i = 0;i < contactPointsItems.length;i++) {
1268             var oi = new Object();
1269             oi.id = contactPointsItems[i].id;
1270             oi.customerId = contactPointsItems[i].customerId;
1271             oi.type = contactPointsItems[i].type;
```

```
1272         oi.identification =
contactPointsItems[i].identification;
1273         var cp = new Object();
1274         if(contactPointsItems[i] instanceof Identification)
{
1275             cp.identification = oi;
1276         } else {
1277             oi.nickname = contactPointsItems[i].nickname;
1278             cp.otherIdentification = oi;
1279         }
1280         contactPointsArray.push(cp);
1281     }
1282
1283     var pl = new Object();
1284     pl.origin = setting.origin;
1285     var customerAlert = new Object();
1286     customerAlert.customerId = session.customer.id;
1287     customerAlert.alertTypeId = alertTypeId;
1288
1289     //Only add frequency if this alert type supports it else
do not add this parameter
1290     if(frequencyVal.notifMax > -1) {
1291         customerAlert.notifMaxCnt = frequencyVal.notifMax;
1292         if(frequencyVal.notifMax != 0) {
1293             customerAlert.notifMaxRecur =
frequencyVal.maxRecur;
1294         }
1295     }
1296
1297     customerAlert.alertNotificationMsgId =
alertNotificationMsdId;
1298
1299     if(alertDataListItems != null &&
alertDataListItems.length > 0) {
```

Money Mobiliser Core API

```
1300         customerAlert.alertDataList = new Object();
1301         customerAlert.alertDataList.alertData =
alertDataListArray;
1302     } else {
1303         delete customerAlert.alertDataList;
1304     }
1305
1306     if(contactPointsArray != null &&
contactPointsArray.length > 0) {
1307         customerAlert.contactPointList = new Object();
1308         customerAlert.contactPointList.contactPoint =
contactPointsArray;
1309     }
1310
1311     pl.customerAlert = customerAlert;
1312     pl.traceNo = UUIDv4();
1313     pl.AuditData = setting.appinfo;
1314
1315     smartphoneService.post(pl, "createCustomerAlert",
responseBack);
1316 };
1317
1318 /**
1319  @description delete an existing alert
1320  @param alert_id id of an alert which needs to be deleted
1321  @param responseBack Indicate which function to be called when
a response is received.
1322  */
1323  MobiliserClient.prototype.deleteCustomerAlert =
function(alertId, responseBack) {
1324      var pl = new Object();
1325      pl.origin = setting.origin;
1326      pl.customerAlertId = alertId;
1327      pl.traceNo = UUIDv4();
```

```
1328         pl.AuditData = setting.appinfo;
1329
1330         smartphoneService.post(pl, "deleteCustomerAlert",
responseBack);
1331     };
1332
1333     /**
1334     * @description delete an alert if the account itself is
deleted
1335     * @param responseBack function to be called in case of
success
1336     * @param pIId payment instrument id of the existing alert
data key record
1337     */
1338     MobiliserClient.prototype.deleteCustomerAlertByCustomerAndData =
function(responseBack, pIId) {
1339         var pl = new Object();
1340         pl.origin = setting.origin;
1341         pl.traceNo = UUIDv4();
1342         pl.AuditData = setting.appinfo;
1343         pl.customerId = session.customer.id;
1344         pl.key = "customerPiId";
1345         pl.data = session.accountPiIdForDeletion;
1346
1347         smartphoneService.post(pl,
"deleteCustomerAlertByCustomerAndData", responseBack);
1348     };
1349
1350     /**
1351     * @description get alert details for an existing alert
1352     * @param responseBack function to be called in case of
success
1353     * @param alert_id id of the existing alert record
1354     */
```

Money Mobiliser Core API

```
1355 MobiliserClient.prototype.getAlertDetailsForEdit =  
function(responseBack, alertId) {  
1356     var pl = new Object();  
1357     pl.origin = setting.origin;  
1358     pl.customerAlertId = alertId;  
1359     pl.traceNo = UUIDv4();  
1360     pl.AuditData = setting.appinfo;  
1361  
1362     smartphoneService.post(pl, "getCustomerAlert",  
responseBack);  
1363 };  
1364  
1365 /**  
1366  * @description Updates an existing alert  
1367  * @param responseBack Response Handler  
1368  * @param alertId Id of customer alert  
1369  * @param alertTypeId alert type id  
1370  * @param alertDataList alert data list  
1371  */  
1372 MobiliserClient.prototype.updateExistingAlert =  
function(responseBack, customerAlert) {  
1373     //     var alertData = customerAlert.alertDataList;  
1374     //     var alertDataListArray = [];  
1375     //     for(var i = 0;i < alertData.length;i++) {  
1376     //         var ad = new Object();  
1377     //         ad.id = alertData[i].id;  
1378     //         ad.customerAlertId =  
alertData[i].customerAlertId;  
1379     //         ad.key = alertData[i].key;  
1380     //         ad.value = alertData[i].value;  
1381     //         alertDataListArray.push(ad);  
1382     //     }  
1383     //  
1384     //     var contactPoint = customerAlert.contactPointList;
```

```
1385 // var contactPointsArray = [];
1386 // for(var i = 0;i < contactPoint.length;i++) {
1387 //     var cp = new Object();
1388 //     if(contactPoint[i].id) // It may be a new addition
and hence may not have an id
1389 //         cp.id = contactPoint[i].id;
1390 //
1391 //     if(contactPoint[i].customerAlertId) // It may be a
new addition and hence may not have an id
1392 //         cp.customerAlertId =
contactPoint[i].customerAlertId;
1393 //
1394 //     var oi = new Object();
1395 //     var tmpObj = contactPoint[i].otherIdentification
1396 //     if(!tmpObj) {
1397 //         tmpObj = contactPoint[i].identification;
1398 //     }
1399 //
1400 //     oi.id = tmpObj.id;
1401 //     oi.customerId = tmpObj.customerId;
1402 //     oi.type = tmpObj.type;
1403 //     oi.identification = tmpObj.identification;
1404 //
1405 //     if(contactPoint[i].otherIdentification) {
1406 //         oi.nickname =
contactPoint[i].otherIdentification.nickname;
1407 //         cp.otherIdentification = oi
1408 //     } else {
1409 //         cp.identification = oi;
1410 //     }
1411 //
1412 //     contactPointsArray.push(cp);
1413 // }
1414
```

Money Mobiliser Core API

```
1415     var pl = new Object();
1416     pl.origin = setting.origin;
1417     pl.traceNo = UUIDv4();
1418     pl.AuditData = setting.appinfo;
1419     pl.customerAlert = customerAlert;
1420
1421     //Restructure Alert Data
1422     var alertData = pl.customerAlert.alertDataList;
1423     pl.customerAlert.alertDataList = new Object();
1424     if(alertData != null && alertData.length > 0) {
1425         pl.customerAlert.alertDataList.alertData =
alertData;
1426     } else {
1427         delete pl.customerAlert.alertDataList;
1428     }
1429
1430     //Restructure Contact Point
1431     var contactPoint = pl.customerAlert.contactPointList;
1432     pl.customerAlert.contactPointList = new Object();
1433     if(contactPoint != null && contactPoint.length > 0) {
1434         pl.customerAlert.contactPointList.contactPoint =
contactPoint;
1435     } else {
1436         delete
pl.customerAlert.contactPointList.contactPoint;
1437     }
1438
1439     smartphoneService.post(pl, "updateCustomerAlert",
responseBack);
1440 };
1441
1442 /**
1443  * @description Service call to fetch the alert notification
msg type id based of alert type id and notification msg id
```



```
1444     * @param alertTypeId alert type
1445     * @param notification type
1446     */
1447     MobiliserClient.prototype.getAlertNotificationMsgId =
function(responseBack, alertTypeId, notificationMsgTypeId) {
1448         var pl = new Object();
1449         pl.origin = setting.origin;
1450         pl.traceNo = UUIDv4();
1451         pl.AuditData = setting.appinfo;
1452         pl.alertTypeId = alertTypeId;
1453         pl.notificationMsgTypeId = notificationMsgTypeId;
1454         smartphoneService.post(pl,
"getActiveAlertNotifMsgByAlertTypeAndNotifMsgType", responseBack);
1455     };
1456
1457
1458     /**
1459     * @description Service call to fetch the active alert
notification message mapping, to be used in
1460     * creating and updating alerts
1461     * @param responseBack callback handler when the results
returned.
1462     */
1463     MobiliserClient.prototype.getActiveAlertNotificationMessages
= function(responseBack) {
1464         var pl = new Object();
1465         pl.origin = setting.origin;
1466         pl.traceNo = UUIDv4();
1467         pl.AuditData = setting.appinfo;
1468
1469         smartphoneService.post(pl,
"getActiveAlertNotificationMessages", responseBack);
1470     };
1471
```

Money Mobiliser Core API

```
1472     /**
1473     * @description Function to fetch list of supported look up
1474     * items like currencies
1475     * @param responseBack the callback handler
1476     * @param entity to be looked up on the server
1477     */
1478     MobiliserClient.prototype.getLookups =
1479     function(responseBack, entity) {
1480         var pl = new Object();
1481         pl.origin = setting.origin;
1482         pl.traceNo = UUIDv4();
1483         pl.AuditData = setting.appinfo;
1484         pl.entityName = entity;
1485         smartphoneService.post(pl, "getLookups", responseBack);
1486     };
1487
1488
1489     /**
1490     * @description get Customer's other identifications
1491     * @param responseBack Indicate which function to be called
1492     * when a response is received.
1493     */
1494     MobiliserClient.prototype.getOtherIdentifications =
1495     function(responseBack) {
1496         var pl = new Object();
1497         pl.origin = setting.origin;
1498         pl.customerId = session.customer.id;
1499         pl.traceNo = UUIDv4();
1500         pl.AuditData = setting.appinfo;
1501         smartphoneService.post(pl,
1502         "getCustomerOtherIdentificationByCustomer", responseBack);
```

```
1501     };
1502
1503     /**
1504     * @description get picture upload URL
1505     * @param responseBack Indicate which function to be called
1506     * when a response is received.
1507     */
1508     MobiliserClient.prototype.getUploadPictureURL =
1509     function(responseBack) {
1510         var pl = {};
1511         pl.origin = setting.origin;
1512         pl.traceNo = UUIDv4();
1513         pl.AuditData = setting.appinfo;
1514
1515         smartphoneService.post(pl, "uploadPicture",
1516         responseBack);
1517     };
1518
1519     /**
1520     * @description get picture download URL
1521     * @param responseBack Indicate which function to be called
1522     * when a response is received.
1523     */
1524     MobiliserClient.prototype.getDownloadPictureURL =
1525     function(responseBack) {
1526         var pl = {};
1527         pl.origin = setting.origin;
1528         pl.traceNo = UUIDv4();
1529         pl.AuditData = setting.appinfo;
1530
1531         smartphoneService.post(pl, "getMyPictures",
1532         responseBack);
1533     };
1534
```

Money Mobiliser Core API

```
1529     /**
1530     * @description find next address with location
1531     * @param responseBack Indicate which function to be called
when a response is received.
1532     * @param lat The latitude of current geolocation.
1533     * @param lng The longitude of current geolocation.
1534     */
1535     MobiliserClient.prototype.findNextAddresses =
function(responseBack, lat, lng) {
1536         var pl_loc = {};
1537         pl_loc.latitude = lat;
1538         pl_loc.longitude = lng;
1539         pl_loc.radius = "300000";
1540
1541         var pl = {};
1542         pl.origin = setting.origin;
1543         pl.traceNo = UUIDv4();
1544         pl.AuditData = setting.appinfo;
1545         pl.location = pl_loc;
1546
1547         smartphoneService.post(pl, "findNextAddresses",
responseBack);
1548     };
1549
1550     /**
1551     * @description get default Terms and Conditions
information
1552     * @param responseBack Indicate which function to be called
when a response is received.
1553     */
1554     MobiliserClient.prototype.getDefaultTermAndConditions =
function(responseBack, customerTypeId) {
1555         var pl = {};
1556         pl.origin = setting.origin;
```

```
1557     pl.traceNo = UUIDv4();
1558     pl.AuditData = setting.appinfo;
1559     pl.customerTypeId = customerTypeId;
1560     pl.orgunitId = "0000";
1561
1562     smartphoneService.post(pl, "getDefaultTermAndConditions",
responseBack);
1563 };
1564
1565 /**
1566  * @description get open Terms and Conditions information
1567  * @param responseBack Indicate which function to be called
when a response is received.
1568  */
1569     MobiliserClient.prototype.getOpenTermAndConditions =
function(responseBack) {
1570         var pl = {};
1571         pl.origin = setting.origin;
1572         pl.traceNo = UUIDv4();
1573         pl.AuditData = setting.appinfo;
1574         pl.customerId = session.customer.id;
1575
1576         smartphoneService.post(pl, "getOpenTermAndConditions",
responseBack);
1577     };
1578
1579 /**
1580  * @description accept Terms and Conditions information
1581  * @param responseBack Indicate which function to be called
when a response is received.
1582  */
1583     MobiliserClient.prototype.acceptTermAndConditions =
function(responseBack, tncs) {
1584         var pl = {};
```

Money Mobiliser Core API

```
1585     pl.origin = setting.origin;
1586     pl.traceNo = UUIDv4();
1587     pl.AuditData = setting.appinfo;
1588     pl.tncVersionId = tncs;
1589
1590     smartphoneService.post(pl, "acceptTermAndConditions",
responseBack);
1591 };
1592
1593 /**
1594  @description FindNextCouponTypesByTags function
1595  @param responseBack Indicates which function to be called
when a response is received.
1596  @param tag The tag of the coupon
1597  @param locale The location of the coupon
1598  @param mimeType The mimetype of the coupon
1599  */
1600  MobiliserClient.prototype.findNextCouponTypesByTags =
function(responseBack, tags) {
1601     var tagarray = tags.split(" ");
1602     var pl = new Object();
1603     pl.origin = setting.origin;
1604     pl.traceNo = UUIDv4();
1605     pl.AuditData = setting.appinfo;
1606     pl.tag = tagarray;
1607     pl.locale = locale;
1608     pl.mimeType = mimeType;
1609     pl.location = new Object();
1610     pl.location.latitude = geoLocation.latitude;
1611     pl.location.longitude = geoLocation.longitude;
1612     pl.location.radius = "3000";
1613
1614     smartphoneService.post(pl, "findNextCouponTypes",
responseBack);
```

```

1615     };
1616
1617     /**
1618     @description findNextCouponTypesForCategory function
1619     @param responseBack Indicates which function to be called
when a response is received.
1620     @param categoryId The category of the coupon
1621     @param locale The location of the coupon
1622     @param mimeType The mimetype of the coupon
1623     */
1624     MobiliserClient.prototype.findNextCouponTypesForCategory =
function(responseBack, categoryId) {
1625         var pl = new Object();
1626         pl.origin = setting.origin;
1627         pl.traceNo = UUIDv4();
1628         pl.AuditData = setting.appinfo;
1629         pl.categoryId = categoryId;
1630         pl.locale = locale;
1631         pl.mimeType = mimeType;
1632         pl.location = new Object();
1633         pl.location.latitude = geoLocation.latitude;
1634         pl.location.longitude = geoLocation.longitude;
1635         pl.location.radius = "3000";
1636
1637         smartphoneService.post(pl, "findNextCouponTypes",
responseBack);
1638     };

```

SY_Transactions.js

This is the actual response handling and processing of the web service calls that are made by the MobiliserClient class.

```

1     /**
2     * @overview
3     * This is the actual response handling and processing of the
web service calls

```

Money Mobiliser Core API

```
4      * that are made by the MobiliserClient class. Any
manipulation of the response,
5      * extraction of needed data and the making of Data Objects
(DO) out of the XML
6      * document would be inserted here.
7      *
8      * @name SY_Transactions.js
9      * @author SAP AG
10     * @version 1.0
11     *
12     */
13
14     /**
15     @function
16
17     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
18     */
19     var loginBack = function(r) {
20         var rCode = getStatusCode(r);
21         if (appstate == trans.TRAN_LOGIN) {
22             if(rCode == '0') {
23                 loginParseXML(r);
24                 return true;
25             } else {
26                 _alertBox(Str_alert_loignfailed +
getErrorMessage(r));
27                 return false;
28             }
29         }
30     };
31
32     /**
33     @function
```



```
34
35     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
36     */
37     var fbssologinBack = function(r) {
38         var rCode = getStatusCode(r);
39         if(rCode == '0') {
40             fbssloginParseXML(r);
41             return true;
42         } else {
43             _alertBox(Str_msg_failfacebookssso +
getErrorMessage(r));
44             return false;
45         }
46     };
47
48     /**
49     @function
50
51     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
52     */
53     var getRsaPublicKeyBack = function(r) {
54         var rCode = getStatusCode(r);
55         if("0" === rCode) {
56             if (!session.keys) {
57                 session.keys = [];
58             }
59             session.keys[session.keyType] = r.key;
60             return true;
61         } else {
62             _alertBox(Str_msg_failgetpublickey +
getErrorMessage(r));
63             return false;
```

Money Mobiliser Core API

```
64     }
65 };
66
67 /**
68     @function
69
70     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
71     */
72     var logoutBack = function(r) {
73         var rCode = getStatusCode(r);
74         if(rCode == '0') {
75             $.mobile.changePage("login.html", page_opt);
76             session = new LoginSession();
77             return true;
78         } else {
79             _alertBox(Str_alert_logoutfailed +
getErrorMessage(r));
80             return false;
81         }
82     };
83
84 /**
85     @function
86
87     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
88     */
89     var changeCredentialBack = function(r) {
90         hideProgress();
91
92         var rCode = getStatusCode(r);
93         if (appstate == trans.TRAN_CHANGEPIN) {
```

```

94         if (rCode == '0') {
95             // if succeeded then...
96             $.mobile.changePage("changebindone.html",
page_opt);
97         } else {
98             // if failed then...
99             _alertBox(Str_alert_changepinfailed +
getErrorMessage(r));
100         }
101     }
102     appstate = trans.TRAN_IDLE;
103 };
104
105 /**
106  @function
107
108  @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
109  */
110  var checkCredentialBack = function(r) {
111      var rCode = getStatusCode(r);
112      if (rCode == '0') {
113          if (appstate == trans.TRAN_BLOCKACCOUNT) {
114              mc.updateCustomerBlockAccount(updateCustomerBack);
115          } else {
116              hideProgress();
117              appstate = trans.TRAN_IDLE;
118              _alertBox(Str_alert_correctpin);
119              $.mobile.changePage("manage.html", page_opt);
120          }
121      } else {
122          hideProgress();
123          appstate = trans.TRAN_IDLE;

```

Money Mobiliser Core API

```
124         _alertBox(getErrorMessage(r));
125         $("#blockpin").val('');
126     }
127 };
128
129 /**
130     @function
131
132     @param r This is the XML document/JSON object that is
    returned by the AJAX call made by the MobiliserClient.
133     */
134     var balanceInquiryBack = function(r) {
135         var rCode = getStatusCode(r);
136         if (rCode == '0') {
137             session.pibalance= parseReturnEntry(r, 'Balance');
138             return true;
139         } else {
140             _alertBox(Str_alert_balanceenqfailed +
    getErrorMessage(r));
141             return false;
142         }
143     };
144
145 /**
146     @function
147
148     @param r This is the XML document/JSON object that is
    returned by the AJAX call made by the MobiliserClient.
149     */
150     var getWalletBack = function(r) {
151         //console.log("getWallet:" + JSON.stringify(r));
152         var rCode = getStatusCode(r);
153         if (rCode == '0') {
```

```
154         walletParseXML(r);
155         return true;
156     } else {
157         _alertBox(Str_alert_gettingwalletfailed +
158         getErrorMessage(r));
159         return false;
160     }
161 };
162 /**
163     @function
164
165     @param xmlHttpRequest
166     */
167     var updateBankAccountBack = function(r) {
168         var rCode = getStatusCode(r);
169         if (rCode == '0') {
170             return true;
171         } else {
172             _alertBox(Str_alert_updatepayfailed +
173             getErrorMessage(r));
174             return false;
175         }
176     };
177 /**
178     @function
179
180     @param xmlHttpRequest
181     */
182     var setPrimaryBack = function(r) {
183         var rCode = getStatusCode(r);
184         if (rCode == '0') {
```

Money Mobiliser Core API

```
185         return true;
186     } else {
187         _alertBox(Str_alert_primarywalletfail +
getErrorMessage(r));
188         return false;
189     }
190 };
191
192 /**
193     @function
194
195     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
196 */
197     var createWalletEntryBack = function(r) {
198         var rCode = getStatusCode(r);
199         if (rCode == '0') {
200             return true;
201         } else {
202             _alertBox(Str_alert_addwalletentry +
Str_alert_isfailed + getErrorMessage(r));
203             return false;
204         }
205     };
206
207 /**
208     @function
209
210     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
211 */
212     var updateWalletEntryBack = function(r) {
213         var rCode = getStatusCode(r);
214         if (rCode == '0') {
```

```
215         return true;
216     } else {
217         _alertBox(Str_alert_editwallet + Str_alert_isfailed +
getErrorMessage(r));
218         return false;
219     }
220 };
221
222 /**
223     @function
224
225     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
226     */
227     var deleteBankWalletEntryBack = function(r) {
228         var rCode = getStatusCode(r);
229         if (rCode == '0') {
230             return true;
231         } else {
232             _alertBox(Str_alert_removewallet + Str_alert_isfailed
+ getErrorMessage(r));
233             return false;
234         }
235     };
236
237 /**
238     @function
239
240     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
241     */
242     var deleteCardWalletEntryBack = function(r) {
243         var rCode = getStatusCode(r);
244         if (rCode == '0') {
```

Money Mobiliser Core API

```
245         return true;
246     } else {
247         _alertBox(Str_alert_removecreditcard +
Str_alert_isfailed + getErrorMessage(r));
248         return false;
249     }
250 };
251
252     var deleteCustomerAlertByCustomerAndDataBack = function(r)
{
253         var rCode = getStatusCode(r);
254         if (rCode == '0') {
255             return true;
256         } else {
257             _alertBox(Str_alert_assoalertfailed +
getErrorMessage(r));
258             return false;
259         }
260     };
261
262     /**
263     @function
264
265     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
266     */
267     var createSmsTokenBack = function(r) {
268         hideProgress();
269
270         var rCode = getStatusCode(r);
271         if (appstate == trans.TRAN_CREATESMSTOKEN) {
272             if (rCode == '0') {
273                 $.mobile.changePage("regpin.html", page_opt);
274             } else {
```



```
275     _alertBox(Str_alert_smstokenfailed +  
getErrorMessage(r));  
276     }  
277     }  
278     appstate = trans.TRAN_IDLE;  
279     };  
280  
281     /**  
282     @function  
283  
284     @param r This is the XML document/JSON object that is  
returned by the AJAX call made by the MobiliserClient.  
285     */  
286     var createFullCustomerBack = function(r) {  
287         hideProgress();  
288  
289         var rCode = getStatusCode(r);  
290         if (appstate == trans.TRAN_CREATECUSTOMER) {  
291             if (rCode == '0') {  
292                 var customerId = parseReturnEntry(r,  
'customerId');  
293                 $.mobile.changePage("regdone.html", page_opt);  
294             } else {  
295                 _alertBox(Str_alert_createcustomerfailed +  
getErrorMessage(r));  
296             }  
297         }  
298         appstate = trans.TRAN_IDLE;  
299     };  
300  
301     /**  
302     @function  
303
```

Money Mobiliser Core API

```
304     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
305     */
306     var updateCustomerBack = function(r) {
307         hideProgress();
308
309         var rCode = getStatusCode(r);
310         if (appstate == trans.TRAN_BLOCKACCOUNT) {
311             if (rCode == '0') {
312                 _alertBox(Str_alert_accountblocked);
313                 $("#username").val('');
314                 $.mobile.changePage("login.html", page_opt);
315             } else {
316                 hideProgress();
317                 _alertBox(Str_alert_blockfailed +
getErrorMessage(r));
318             }
319         } else if (appstate == trans.TRAN_NOTIFICATION) {
320             if (rCode == '0') {
321                 var mode = 0;
322                 if($("#checkboxbox-sms").is(':checked') && $
('#marketpref').val() == "on")
323                     mode = mode | 1;
324                 if($("#checkboxbox-email").is(':checked') && $
('#marketpref').val() == "on")
325                     mode = mode | 2;
326                 session.customer.modeId = mode;
327
328                 _alertBox(Str_alert_notificationsaved);
329                 $.mobile.changePage("manage.html", page_opt);
330             } else {
331                 _alertBox(Str_alert_notificationfailed +
getErrorMessage(r));
332             }
```

```
333     }
334     appstate = trans.TRAN_IDLE;
335 };
336
337 /**
338     @function
339
340     @param r This is the XML document/JSON object that is
341     returned by the AJAX call made by the MobiliserClient.
342     */
343     var createIdentificationBack = function(r) {
344         var rCode = getStatusCode(r);
345         if (appstate == trans.TRAN_CREATECUSTOMER) {
346             if (rCode == '0') {
347                 mc.setCredential(registerPinBack,
348                 parseReturnEntry(r, 'customerId'), $('#mobilepin').val(), "0");
349             } else {
350                 hideProgress();
351                 appstate = trans.TRAN_IDLE;
352                 _alertBox(Str_alert_identificationfailed +
353                 getErrorMessage(r));
354             }
355         }
356     };
357
358     /**
359     @function
360
361     @param r This is the XML document/JSON object that is
362     returned by the AJAX call made by the MobiliserClient.
363     */
364     var getIdentificationsBack = function(r) {
365         var rCode = getStatusCode(r);
```

Money Mobiliser Core API

```
362         //console.log("getIdentifications:" +
JSON.stringify(r));
363         if (rCode == '0') {
364             session.msisdn = parseIdentification(r);
365             return true;
366         } else {
367             _alertBox(Str_alert_getidentificationfailed +
getErrorMessage(r));
368             return false;
369         }
370     };
371
372     /**
373     @function
374
375     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
376     */
377     var registerPinBack = function(r) {
378         var rCode = getStatusCode(r);
379         if (appstate == trans.TRAN_CREATECUSTOMER) {
380             if (rCode == '0') {
381                 mc.setCredential(registerPasswordBack,
session.customer.id, $('#webpwd').val(), "1");
382             } else {
383                 hideProgress();
384                 appstate = trans.TRAN_IDLE;
385                 _alertBox(Str_alert_registercustomerpinfailed +
getErrorMessage(r));
386             }
387         }
388     };
389
390     /**
```

```

391     @function
392
393     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
394     */
395     var registerPasswordBack = function(r) {
396         var rCode = getStatusCode(r);
397         if (appstate == trans.TRAN_CREATECUSTOMER) {
398             if (rCode == '0') {
399                 mc.createIdentification(registerBack,
session.customer.id, msisdnformat($('#phoneno').val()), "0");
400             } else {
401                 hideProgress();
402                 appstate = trans.TRAN_IDLE;
403                 _alertBox(Str_alert_customerpasswordfailed +
getErrorMessage(r));
404             }
405         }
406     };
407
408     /**
409     @function
410
411     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
412     */
413     var registerBack = function(r) {
414         hideProgress();
415
416         var rCode = getStatusCode(r);
417         if (appstate == trans.TRAN_CREATECUSTOMER) {
418             if (rCode == '0') {
419                 $.mobile.changePage("regpin.html", page_opt);
420             } else {

```

Money Mobiliser Core API

```
421         _alertBox(Str_alert_createcustomerfailed +
422         getErrorMessage(r));
423     }
424     appstate = trans.TRAN_IDLE;
425 };
426
427 /**
428     @function
429
430     @param r This is the XML document/JSON object that is
431     returned by the AJAX call made by the MobiliserClient.
432     */
432     var findTxnsBack = function(r) {
433         var rCode = getStatusCode(r);
434         if (appstate == trans.TRAN_TXN) {
435             if (rCode == '0') {
436                 if (txnsParseXML(r))
437                     return true;
438                 else {
439                     _alertBox(Str_alert_nottransactionhistory);
440                     return false;
441                 }
442             } else {
443                 _alertBox(Str_alert_gettransactionhistoryfailed +
444                 getErrorMessage(r));
445                 return false;
446             }
447         };
448
449     /**
450     @function
```

```

451
452     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
453     */
454     var getTxnDetailsBack = function(r) {
455         var rCode = getStatusCode(r);
456         if (appstate == trans.TRAN_TXN) {
457             if (rCode == '0') {
458                 session.transit_value = txnDetailParse(r);
459                 return true;
460             } else {
461                 _alertBox(Str_alert_transactiondetailfailed +
getErrorMessage(r));
462                 return false;
463             }
464         }
465     };
466
467     /**
468     @function
469
470     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
471     */
472     var getInvoiceTypesByGroupBack = function(r) {
473         hideProgress();
474
475         var rCode = getStatusCode(r);
476         if (appstate == trans.TRAN_TOPUP) {
477             if (rCode == '0') {
478                 if (parseGroupInvoiceTypes(r))
479                     $.mobile.changePage("prepaid.html",
page_opt);
480             } else {

```

Money Mobiliser Core API

```
481         appstate = trans.TRAN_IDLE;
482         _alertBox(Str_alert_nooperatorregistered);
483     }
484     } else {
485         // if failed then...
486         appstate = trans.TRAN_IDLE;
487         _alertBox(Str_alert_getinvoicetypefailed +
488         getErrorMessage(r));
489     }
490 };
491
492 /**
493     @function
494
495     @param r This is the XML document/JSON object that is
496     returned by the AJAX call made by the MobiliserClient.
497     */
498     var createInvoiceForInvoiceTypeBack = function(r) {
499         var rCode = getStatusCode(r);
500         if (appstate == trans.TRAN_TOPUP) {
501             if (rCode == '0') {
502                 mc.topUp(topUpCallBack, parseReturnEntry(r,
503                 'invoiceId'));
504             } else {
505                 hideProgress();
506                 appstate = trans.TRAN_IDLE;
507                 _alertBox(Str_alert_getinvoicetypefailed +
508                 getErrorMessage(r));
509             }
510         }
511     };
512 /**
```



```
511     @function
512
513     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
514     */
515     var getBillTypeBack = function(r) {
516         var rCode = getStatusCode(r);
517         if (appstate == trans.TRAN_ENTERPAYBILL) {
518             if (rCode == '0') {
519                 session.billTypes = [];
520                 parseInvoiceTypes(r);
521                 mc.getRegisteredBills(registeredBillsBack,
session.customer.id);
522             } else {
523                 hideProgress();
524                 appstate = trans.TRAN_IDLE;
525                 _alertBox(Str_alert_getbilltypefailed +
getErrorMessage(r));
526             }
527         }
528     };
529
530     /**
531     @function
532
533     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
534     */
535     var registeredBillsBack = function(r) {
536         var rCode = getStatusCode(r);
537         if (rCode == '0') {
538             session.registeredbill = [];
539             userbills = parseRegisteredBills(r);
540
```

Money Mobiliser Core API

```
541         if( appstate == trans.TRAN_ENTERPAYBILL ) {
542             mc.getOpenInvoices(getOpenInvoicesBack,
543 session.customer.id);
544         } else if ( appstate == trans.TRAN_PAYANEWBILL ) {
545             hideProgress();
546             appstate = trans.TRAN_IDLE;
547         }
548         if (userbills) {
549             $.mobile.changePage("paybilldetail.html",
550 page_opt);
551         } else
552             _alertBox(Str_alert_registerbill);
553     }
554     } else {
555         hideProgress();
556         appstate = trans.TRAN_IDLE;
557         _alertBox(Str_alert_registerbillfailed +
558 getErrorMessage(r));
559     }
560 };
561
562 /**
563 @function
564 @param r This is the XML document/JSON object that is
565 returned by the AJAX call made by the MobiliserClient.
566 */
567 var registerSimpleBack = function(r) {
568     hideProgress();
569     var rCode = getStatusCode(r);
570     if (appstate == trans.TRAN_REGISTERBILL) {
571         if (rCode == '0') {
572             _alertBox(Str_alert_registerbillsuccess);
```

```

571         $.mobile.changePage("paybill.html", page_opt);
572     } else {
573         _alertBox(Str_alert_registersimplebackfailed +
574         getErrorMessage(r));
575     }
576     appstate = trans.TRAN_IDLE;
577 };
578
579 /**
580  @function
581
582  @param r This is the XML document/JSON object that is
583  returned by the AJAX call made by the MobiliserClient.
584  */
585  var getOpenInvoicesBack = function(r) {
586
587      hideProgress();
588
589      var rCode = getStatusCode(r);
590      if (rCode == '0') {
591          var h1 = 0;
592          var h2 = 0;
593          session.openRequestArray = [];
594          session.openBillArray = [];
595          parseOpenInvoices(r);
596
597          if (appstate == trans.TRAN_ENTERPAYBILL)
598              $.mobile.changePage("paybill.html", page_opt);
599          else if (appstate == trans.TRAN_SENDBILL)
600              $.mobile.changePage("paybilldone.html",
601              page_opt);
602      } else {
603          _alertBox(Str_alert_openinvoicefailed +
604          getErrorMessage(r));

```

Money Mobiliser Core API

```
601     }
602     appstate = trans.TRAN_IDLE;
603 };
604
605 /**
606     @function
607
608     @param r This is the XML document/JSON object that is
        returned by the AJAX call made by the MobiliserClient.
609 */
610     var createInvoiceBack = function(r) {
611         var rCode = getStatusCode(r);
612         if (appstate == trans.TRAN_SENDBILL) {
613             if (rCode == '0') {
614                 session.transit_value.invoiceId =
        parseReturnEntry(r, 'invoiceId');
615                 mc.payBill(payBillOrCancelBack,
        session.transit_value.invoiceId, $('#billfromaccount').val());
616             } else {
617                 appstate = trans.TRAN_IDLE;
618                 hideProgress();
619                 _alertBox(Str_alert_createinvoicefailed +
        getErrorMessage(r));
620             }
621         }
622     };
623
624 /**
625     @function
626
627     @param r This is the XML document/JSON object that is
        returned by the AJAX call made by the MobiliserClient.
628 */
629     var preAuthoriseBack = function(r) {
```

```

630         var rCode = getStatusCode(r);
631         if (rCode == '0') {
632             session.transit_value.sysid =
parseTransactionSystemId(r);
633             session.transit_value.referno =
parseTransactionValue(r);
634             session.transit_value.payerFee =
parseTransactionFee(r, "false");
635             session.transit_value.payeeFee =
parseTransactionFee(r, "true");
636             return true;
637         } else if (rCode == '2501' && appstate ==
trans.TRAN_SENDMONEY) {
638             return rCode;
639         } else {
640             _alertBox(Str_alert_failedpreauthorise +
getErrorMessage(r));
641             return false;
642         }
643     };
644
645     /**
646     @function
647
648     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
649     */
650     var startVoucherBack = function(r) {
651         var rCode = getStatusCode(r);
652         if (appstate == trans.TRAN_SENDMONEY) {
653             if (rCode == '0') {
654                 session.transit_value.sysid =
parseTransactionSystemId(r);
655                 session.transit_value.referno =
parseTransactionValue(r);

```

Money Mobiliser Core API

```
656         session.transit_value.payerFee =
parseTransactionFee(r, "false");

657         session.transit_value.payeeFee =
parseTransactionFee(r, "true");

658         return true;

659     } else {

660         _alertBox(Str_alert_startvoucherfailed +
getErrorMessage(r));

661         return false;

662     }

663 }

664 };

665

666 /**

667     @function

668

669     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.

670     */

671     var confirmVoucherBack = function(r) {

672         var rCode = getStatusCode(r);

673         if (rCode == '0') {

674             session.transit_value.sysid =
parseTransactionSystemId(r);

675             session.transit_value.referno = parsePickupCode(r);

676             return true;

677         } else {

678             _alertBox(Str_alert_failedtoconfirmvoucher +
getErrorMessage(r));

679             return false;

680         }

681     };

682

683 /**
```

```
684     @function
685
686     @param r This is the XML document/JSON object that is
        returned by the AJAX call made by the MobiliserClient.
687     */
688     var authoriseBack = function(r) {
689         var failureMsg;
690
691         var rCode = getStatusCode(r);
692         if (appstate == trans.TRAN_SENDMONEY) {
693             failureMsg = Str_alert_sendmoneyfailed;
694         } else if (appstate == trans.TRAN_SENDSREQUEST) {
695             failureMsg = Str_alert_sendrequestfailed;
696         } else if (appstate == trans.TRAN_LOADSVA) {
697             failureMsg = Str_alert_loadSVAFailed;
698         } else if (appstate == trans.TRAN_UNLOADSVA) {
699             failureMsg = Str_alert_unloadsvafailed;
700         }
701
702         if (rCode == '0') {
703             session.transit_value.sysid =
        parseTransactionSystemId(r);
704             return true;
705         } else {
706             _alertBox(failureMsg + getErrorMessage(r));
707             return false;
708         }
709     };
710
711     /**
712     @function
713
```

Money Mobiliser Core API

```
714     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
715     */
716     var demandForPaymentBack = function(r) {
717         var rCode = getStatusCode(r);
718         if (appstate == trans.TRAN_SENDRREQUEST) {
719             if (rCode == '0') {
720                 session.transit_value.sysid = parseReturnEntry(r,
'invoiceId');
721                 return true;
722             } else {
723                 _alertBox(Str_alert_sendrequestfailed +
getErrorMessage(r));
724                 return false;
725             }
726         }
727     };
728
729     /**
730     @function
731
732     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
733     */
734     var topUpCallBack = function(r) {
735         _payBillOrCancelBack(r, true);
736     };
737
738     /**
739     @function
740
741     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
742     */
```



```
743     var payBillOrCancelBack = function(r) {
744         _payBillOrCancelBack(r, true);
745     };
746
747     /**
748     @function
749
750     @param r This is the XML document/JSON object that is
751     returned by the AJAX call made by the MobiliserClient.
752     */
753     var payBillBack = function(r) {
754         _payBillOrCancelBack(r, false);
755     };
756
757     /**
758     @function
759
760     @param r This is the XML document/JSON object that is
761     returned by the AJAX call made by the MobiliserClient.
762     @param doCancel The flag to trigger cancel action.
763     */
764     function _payBillOrCancelBack(r, doCancel) {
765         var nextPage;
766         var errmsg = null;
767         if (appstate == trans.TRAN_SENDBILL) {
768             nextPage = "paybillconfirm";
769             errmsg = Str_alert_paybillfailed;
770         } else if (appstate == trans.TRAN_TOPUP) {
771             nextPage = "prepaidconfirm";
772             errmsg = Str_alert_topupfailed;
773         }
774
775         var rCode = getStatusCode(r);
```

Money Mobiliser Core API

```
774         if (rCode == '0') {
775             session.transit_value.sysid =
parseTransactionSystemId(r);
776             session.transit_value.referno =
parseTransactionValue(r);
777             session.transit_value.payerFee =
parseTransactionFee(r, "false");
778             session.transit_value.payeeFee =
parseTransactionFee(r, "true");
779             $.mobile.changePage(nextPage + ".html", page_opt);
780             hideProgress();
781         } else {
782             if (trans.TRAN_TOPUP != appstate && doCancel) { //
openbill TRAN_SENDBILL state
783                 _alertBox(errmsg + getErrorMessage(r) + '\n'+
Str_alert_cancelinvoiceno + session.transit_value.invoiceId);
784                 mc.cancelBill(cancelBillBack,
session.transit_value.invoiceId);
785             } else {
786                 // if request fails to execute
787                 _alertBox(errmsg + getErrorMessage(r));
788                 hideProgress();
789                 appstate = trans.TRAN_IDLE;
790             }
791         }
792     }
793
794     /**
795     @function
796
797     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
798     */
799     var continuePayOrCancelBack = function(r) {
800         _continuePayInvoiceOrCancelBack(r, true);
```

```
801     };
802
803     /**
804     @function
805
806     @param r This is the XML document/JSON object that is
      returned by the AJAX call made by the MobiliserClient.
807     */
808     var continuePayBack = function(r) {
809         _continuePayInvoiceOrCancelBack(r, false);
810     };
811
812     /**
813     @function
814
815     @param r This is the XML document/JSON object that is
      returned by the AJAX call made by the MobiliserClient.
816     @param doCancel The flag to trigger cancel action.
817     */
818     function _continuePayInvoiceOrCancelBack(r, doCancel) {
819         var nextPage;
820         var errmsg = null;
821         if (appstate == trans.TRAN_SENDBILL) {
822             nextPage = "paybilldone";
823             errmsg = Str_alert_paybillfailed;
824         } else if (appstate == trans.TRAN_TOPUP) {
825             nextPage = "prepaiddone";
826             errmsg = Str_alert_topupfailed;
827         }
828
829         var rCode = getStatusCode(r);
830         if (rCode == '0') {
831             if (doCancel) {
```

Money Mobiliser Core API

```
832         session.transit_value.sysid =
parseTransactionSystemId(r);

833         $.mobile.changePage(nextPage + ".html",
page_opt);

834         hideProgress();

835         appstate = trans.TRAN_IDLE;

836     } else { //openbill TRAN_SENDBILL state

837         mc.getOpenInvoices(getOpenInvoicesBack,
session.customer.id);

838     }

839     } else {

840         if (doCancel) {

841             _alertBox( errmsg + getErrorMessage(r) +'\n' +
Str_alert_cancelinvoiceno + session.transit_value.invoiceId);

842             mc.cancelBill(cancelBillBack,
session.transit_value.invoiceId);

843         } else { // if request fails to execute

844             _alertBox(errmsg + getErrorMessage(r));

845             hideProgress();

846             appstate = trans.TRAN_IDLE;

847         }

848     }

849 }

850

851 /**

852     @function

853

854     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.

855     */

856     var cancelBillBack = function(r) {

857         hideProgress();

858

859         var rCode = getStatusCode(r);
```

```
860         if (rCode != '0') {
861             _alertBox(Str_alert_calncelinvoicefailed +
getErrorMessage(r));
862         }
863         appstate = trans.TRAN_IDLE;
864     };
865
866     /**
867     @function
868
869     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
870     */
871     var getBalanceAlertBack = function(r) {
872         var rCode = getStatusCode(r);
873         if (rCode == '0') {
874             parseBalanceAlert(r);
875             return true;
876         } else {
877             _alertBox(Str_alert_balancealertfailed +
getErrorMessage(r));
878             return false;
879         }
880     };
881
882     /**
883     @function
884
885     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
886     */
887     var createBalanceAlertBack = function(r) {
888         var rCode = getStatusCode(r);
889         if (rCode == '0') {
```

Money Mobiliser Core API

```
890         return true;
891     } else {
892         _alertBox(Str_alert_addbalancealert +
Str_alert_isfailed + getErrorMessage(r));
893         return false;
894     }
895 };
896
897 /**
898     @function
899
900     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
901     */
902     var updateBalanceAlertBack = function(r) {
903         var rCode = getStatusCode(r);
904         if (rCode == '0') {
905             return true;
906         } else {
907             _alertBox(Str_alert_editbalancealert +
Str_alert_isfailed + getErrorMessage(r));
908             return false;
909         }
910     };
911
912 /**
913     @function
914
915     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
916     */
917     var deleteBalanceAlertBack = function(r) {
918         var rCode = getStatusCode(r);
919         if (rCode == '0') {
```

```
920         return true;
921     } else {
922         _alertBox(Str_alert_removebalancealert +
Str_alert_isfailed + getErrorMessage(r));
923         return false;
924     }
925 };
926
927 /**
928     @ResponseBack Coupon Features
929     */
930
931 /**
932     @function
933
934     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
935     */
936     var getMyCouponsBack = function(r) {
937         var rCode = getStatusCode(r);
938         if (rCode == '0') {
939             couponparseXML(r);
940             return true;
941         } else {
942             _alertBox(Str_alert_getcouponfailed +
getErrorMessage(r));
943             return false;
944         }
945     };
946
947 /**
948     @function
949
```

Money Mobiliser Core API

```
950     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
951     */
952     var deleteCouponBack = function(r) {
953         var rCode = getStatusCode(r);
954         if (rCode == '0') {
955             _alertBox(Str_alert_coupondeletedsuccussfully);
956             return true;
957         } else {
958             _alertBox(Str_alert_deletedcouponfailed +
getErrorMessage(r));
959             return false;
960         }
961     };
962
963     /**
964     @function
965
966     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
967     */
968     var purchaseCouponBack = function(r) {
969         var rCode = getStatusCode(r);
970         if (rCode == '0') {
971             _alertBox(Str_alert_couponpurchasesuccess);
972             return true;
973         } else {
974             _alertBox(Str_alert_couponpurchasefailed +
getErrorMessage(r));
975             return false;
976         }
977     };
978
979     /**
```



```

980     @function
981
982     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
983     */
984     var getRootCategoriesBack = function(r) {
985         var rCode = getStatusCode(r);
986         if (rCode == '0') {
987             getRootCategoriesParseXML(r);
988             return true;
989         } else {
990             _alertBox(Str_alert_rootcategoryfailed +
getErrorMessage(r));
991             return false;
992         }
993     };
994
995     /**
996     @function
997
998     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
999     */
1000    var getCategoryTreeBack = function(r) {
1001        var rCode = getStatusCode(r);
1002        if (rCode == '0') {
1003            if (!getCategoryTreeBackParseXML(r))
1004                _alertBox(Str_alert_nocategoryavailable);
1005            return true;
1006        } else {
1007            _alertBox(Str_alert_rootcategoryfailed +
getErrorMessage(r));
1008            return false;
1009        }

```

Money Mobiliser Core API

```
1010     };
1011
1012     /**
1013     @function
1014
1015     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1016     */
1017     var getChildCategoriesBack = function(r) {
1018         var rCode = getStatusCode(r);
1019         if (rCode == '0') {
1020             getChildCategoriesParseXML(r);
1021             return true;
1022         } else {
1023             _alertBox(Str_alert_childcategoryfailed +
getErrorMessage(r));
1024             return false;
1025         }
1026     };
1027
1028     /**
1029     @function
1030
1031     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1032     */
1033     var getCouponTypesForCategoryBack = function(r) {
1034         var rCode = getStatusCode(r);
1035         if (rCode == '0') {
1036             getCouponTypesofCategoryParseXML(r);
1037             return true;
1038         } else {
1039             _alertBox(Str_alert_coupontypesfaield +
getErrorMessage(r));
```

```
1040         return false;
1041     }
1042 };
1043
1044 /**
1045  @function
1046
1047  @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1048  */
1049  var assignCouponBack = function(r) {
1050      var rCode = getStatusCode(r);
1051      if (rCode == '0') {
1052          _alertBox(Str_alert_couponaddedsuccess);
1053          return true;
1054      } else {
1055          _alertBox(Str_alert_assigncouponfailed +
getErrorMessage(r));
1056          return false;
1057      }
1058  };
1059
1060 /**
1061  @function
1062
1063  @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1064  */
1065  var findCouponTypesByTagsBack = function(r) {
1066      var rCode = getStatusCode(r);
1067      if (rCode == '0') {
1068          couponTypesByTagsparseXML(r);
1069          return true;
```

Money Mobiliser Core API

```
1070     } else {
1071         _alertBox(Str_alert_failedtogetcoupon +
1072         getErrorMessage(r));
1073     }
1074 };
1075
1076
1077 /**
1078     @function retrieve upload URL
1079
1080     @param r This is the XML document/JSON object that is
1081     returned by the AJAX call made by the MobiliserClient.
1082     */
1083     var getUploadPictureURLBack = function(r) {
1084         var rCode = getStatuscode(r);
1085         if (rCode == '0') {
1086             session.uploadUrl = parseReturnEntry(r,
1087             'uploadUrl');
1088         } else {
1089             hideProgress();
1090             _alertBox(Str_alert_failgetuploadurl +
1091             getErrorMessage(r));
1092             appstate = trans.TRAN_IDLE;
1093         }
1094     };
1095
1096 /**
1097     @function retrieve download URL
1098
1099     @param r This is the XML document/JSON object that is
1100     returned by the AJAX call made by the MobiliserClient.
1101     */
1102     var downloadPictureURLBack = function(r) {
```

```
1099     var rCode = getStatusCode(r);
1100     if (rCode == '0') {
1101         session.downloads = [];
1102         if (r.pictures) {
1103             for (var i = 0, len = r.pictures.length, picture; i
1104 < len; i++) {
1105                 picture = r.pictures[i];
1106                 session.downloads.push({url: picture.url,
1107 contentype: picture.contentType});
1108             }
1109         } else
1110             _alertBox(Str_alert_failgetdownloadurl +
1111 getErrorMessage(r));
1112     };
1113
1114     /**
1115     @function retrieve locations
1116     @param r This is the XML document/JSON object that is
1117 returned by the AJAX call made by the MobiliserClient.
1118     */
1119     var findNextAddressesBack = function(r) {
1120         var rCode = getStatusCode(r);
1121         if (rCode == '0') {
1122             session.locations = r.location;
1123         } else {
1124             hideProgress();
1125             _alertBox(Str_alert_failfindnextaddrs +
1126 getErrorMessage(r));
1127             appstate = trans.TRAN_IDLE;
1128         }
1129     };
1130 }
```

Money Mobiliser Core API

```
1128     /**
1129     @function get Default TermAndConditions
1130
1131     @param r This is the XML document/JSON object that is
1132     returned by the AJAX call made by the MobiliserClient.
1133     */
1134     var getDefaultTermAndConditionsBack = function(r) {
1135         var rCode = getStatusCode(r);
1136         if (rCode == '0') {
1137             session.defaultTermAndConditions =
1138             r.termAndCondition;
1139             return true;
1140         } else {
1141             _alertBox(Str_alert_failgetdefaulttandc +
1142             getErrorMessage(r));
1143             return false;
1144         }
1145     };
1146
1147     /**
1148     @function get Open TermAndConditions
1149
1150     @param r This is the XML document/JSON object that is
1151     returned by the AJAX call made by the MobiliserClient.
1152     */
1153     var getOpenTermAndConditionsBack = function(r) {
1154         var rCode = getStatusCode(r);
1155         if (rCode == '0') {
1156             session.openTermAndConditions = r.termAndCondition;
1157             return true;
1158         } else {
1159             _alertBox(Str_alert_failgetopentandc +
1160             getErrorMessage(r));
1161             return false;
1162         }
1163     };
1164 }
```

```
1157     }
1158   };
1159
1160   /**
1161     @function accept TermAndConditions
1162
1163     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1164   */
1165   var acceptTermAndConditionsBack = function(r) {
1166     var rCode = getStatusCode(r);
1167     if (rCode == '0') {
1168       if (r.privileges)
1169         for (var i=0; i<r.privileges.length; i++)
1170 session.customer.privileges.push(r.privileges[i]);
1171       return true;
1172     } else {
1173       _alertBox(Str_alert_failaccepttandc +
getErrorMessage(r));
1174       return false;
1175     }
1176   };
1177
1178   /**
1179     @function
1180
1181     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1182   */
1183   var findNextCouponTypesBack = function(r) {
1184     var rCode = getStatusCode(r);
1185     if (rCode == '0') {
1186       session.couponTypes = r.result;
```

Money Mobiliser Core API

```
1187         return true;
1188     } else {
1189         _alertBox(Str_alert_failedtogetcoupontypes +
1190         getErrorMessage(r));
1191         return false;
1192     }
1193 };
1194
1195 /
1196 *****
1197 *****
1198 * Functions
1199 */
1200
1201 @function
1202
1203 @param r This is the XML document/JSON object that is
1204 returned by the AJAX call made by the MobiliserClient.
1205 */
1206 function getStatusCode(r) {
1207     return r.Status.code.toString();
1208 }
1209
1210 @function
1211
1212 @param r This is the XML document/JSON object that is
1213 returned by the AJAX call made by the MobiliserClient.
1214 @param entryName This is string of the entry node name in
1215 XML document/JSON object.
1216 */
1217 function parseReturnEntry(r, entryName) {
1218     return r[entryName];
1219 }
```



```
1215     }
1216
1217     /**
1218         @function
1219
1220         @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1221         @param entryName This is string of the entry node name in
XML document/JSON object.
1222         @param attrName This is string of attribute name of the
entry node in XML document/JSON object.
1223     */
1224     function parseReturnEntryAttr(r, entryName, attrName) {
1225         return r[entryName][attrName].toString();
1226     }
1227
1228     /**
1229         @function
1230
1231         @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1232     */
1233     function parseIdentification(r) {
1234         var msisdn = null;
1235         if (r.identifications && r.identifications.length) {
1236             for(var i = 0;i < r.identifications.length;i++) {
1237                 if (r.identifications[i].type == 0) {
1238                     session.identification =
r.identifications[i];
1239                     msisdn =
r.identifications[i].identification;
1240                     break;
1241                 }
1242             }

```

Money Mobiliser Core API

```
1243     }
1244     return msisdn;
1245 }
1246
1247 /**
1248     @function
1249
1250     @param r This is the XML document/JSON object that is
1251     returned by the AJAX call made by the MobiliserClient.
1252 */
1253 function parseTransactionSystemId(r) {
1254     if (r.Transaction && r.Transaction.systemId)
1255         return r.Transaction.systemId;
1256     else
1257         return null;
1258 }
1259 /**
1260     @function
1261
1262     @param r This is the XML document/JSON object that is
1263     returned by the AJAX call made by the MobiliserClient.
1264 */
1265 function parseTransactionValue(r) {
1266     if (r.Transaction && r.Transaction.value)
1267         return r.Transaction.value;
1268     else
1269         return null;
1270 }
1271 /**
1272     @function
1273
```

```
1274     @param r This is the XML document/JSON object that is
1275     returned by the AJAX call made by the MobiliserClient.
1276     */
1277     function parseTransactionFee(r, isPayee) {
1278         var transactionfee = 0;
1279         if (r.MoneyFee)
1280             for (var i=0; i<r.MoneyFee.length; i++)
1281                 if (r.MoneyFee[i].payee == isPayee)
1282                     transactionfee +=
1283                     parseFloat(r.MoneyFee[i].value);
1284         return transactionfee;
1285     }
1286
1287     /**
1288     @function
1289
1290     @param r This is the XML document/JSON object that is
1291     returned by the AJAX call made by the MobiliserClient.
1292     */
1293     function parseGroupInvoiceTypes(r) {
1294         // Callback to application processing
1295         if (r.invoiceType !== null) {
1296             session.transit_value = r.invoiceType;
1297             return r.invoiceType.length;
1298         } else
1299             return 0;
1300     }
1301     /**
1302     @function
1303
```

Money Mobiliser Core API

```
1304     @param r This is the XML document/JSON object that is
1305     returned by the AJAX call made by the MobiliserClient.
1306     */
1307     function parseInvoiceTypes(r) {
1308         if (r.invoiceType !== null) {
1309             for (var i=0; i<r.invoiceType.length; i++) {
1310                 session.billTypes[i] = new BillTypes();
1311                 session.billTypes[i].billTypeId =
1312                 r.invoiceType[i].id;
1313                 session.billTypes[i].billTypeName =
1314                 r.invoiceType[i].name;
1315                 session.billTypes[i].billTypeGroupId =
1316                 r.invoiceType[i].groupId;
1317             }
1318         }
1319     }
1320     /**
1321     @function
1322     @param r This is the XML document/JSON object that is
1323     returned by the AJAX call made by the MobiliserClient.
1324     */
1325     function parseRegisteredBills(r) {
1326         var userbills_number = 0;
1327         if (r.invoiceConfiguration !== null) {
1328             for (var i=0; i<r.invoiceConfiguration.length; i++)
1329             {
1330                 session.registeredbill[i] = new
1331                 Registeredbill();
1332                 session.registeredbill[i].billConfigId =
1333                 r.invoiceConfiguration[i].id;
1334                 session.registeredbill[i].billTypeId =
1335                 r.invoiceConfiguration[i].invoiceTypeId;
1336                 session.registeredbill[i].alias =
```

```

1331         (r.invoiceConfiguration.alias === null) ? "" :
r.invoiceConfiguration[i].alias;
1332         session.registeredbill[i].reference =
r.invoiceConfiguration[i].reference;
1333         session.registeredbill[i].billTypeName = "";
1334         session.registeredbill[i].billTypeGroupId = "";
1335         for (var j=0; j<session.billTypes.length; j++)
{
1336             if (session.billTypes[j].billTypeId ==
session.registeredbill[i].billTypeId) {
1337                 session.registeredbill[i].billTypeName =
session.billTypes[j].billTypeName;
1338                 session.registeredbill[i].billTypeGroupId =
session.billTypes[j].billTypeGroupId;
1339                 if (session.billTypes[j].billTypeGroupId ==
0) {
1340                     userbills_number++;
1341                 }
1342                 break;
1343             }
1344         }
1345     }
1346 }
1347     return userbills_number;
1348 }
1349
1350 /**
1351     @function
1352
1353     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1354 */
1355     function parseOpenInvoices(r) {
1356         var h1=0;
1357         var h2=0;

```

```
1358
1359     if (r.invoice != null) {
1360         for (var i=0; i<r.invoice.length; i++) {
1361             var date = (r.invoice[i].data === null) ? "-" :
epoch2UTC(r.invoice[i].date).split("T")[0];
1362
1363             for (var j=0; j<session.registeredbill.length; j++)
{
1364                 if (r.invoice[i].invoiceConfigurationId ==
session.registeredbill[j].billConfigId) {
1365                     var billTypeName =
session.registeredbill[j].alias + ":" +
1366 session.registeredbill[j].billTypeName;
1367                     var invConfReference =
session.registeredbill[j].reference;
1368
1369                     if
(session.registeredbill[j].billTypeGroupId == 3) {
1370                         session.openRequestArray[h1] = new
OpenBill();
1371                         session.openRequestArray[h1].billId =
r.invoice[i].id;
1372 session.openRequestArray[h1].billTypeName = billTypeName;
1373                         session.openRequestArray[h1].dueDate =
date;
1374                         session.openRequestArray[h1].amount =
r.invoice[i].amount / 100;
1375                         session.openRequestArray[h1].reference =
invConfReference;
1376 session.openRequestArray[h1].billreference =
r.invoice[i].reference;
1377                         h1++;
1378                     } else if
(session.registeredbill[j].billTypeGroupId == 0) {
1379                         session.openBillArray[h2] = new
OpenBill();
```

```
1380             session.openBillArray[h2].billId =
r.invoice[i].id;
1381             session.openBillArray[h2].billTypeName =
billTypeName;
1382             session.openBillArray[h2].dueDate =
date;
1383             session.openBillArray[h2].amount =
r.invoice[i].amount / 100;
1384             session.openBillArray[h2].reference =
invConfReference;
1385             session.openBillArray[h2].billreference
= r.invoice[i].reference;
1386                 h2++;
1387             }
1388             break;
1389         }
1390     }
1391 }
1392 }
1393 }
1394
1395 /**
1396     @function
1397
1398     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1399     */
1400     function parsePickupCode(r) {
1401         for (var i=0; i<r.UnstructuredData.length; i++)
1402             if (r.UnstructuredData[i].Key == "PickupCode") {
1403                 return r.UnstructuredData[i].Value;
1404             }
1405         }
1406         return null;
1407     }
```

Money Mobiliser Core API

```
1408
1409  /**
1410     @function
1411
1412     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1413  */
1414  function parseBalanceAlert(r) {
1415      session.svaalert = [];
1416      for (var i=0; i<r.BalanceAlerts.length; i++) {
1417          session.svaalert[i] = new BalanceAlert();
1418          session.svaalert[i].id = r.BalanceAlerts[i].id;
1419          session.svaalert[i].paymentInstrumentId =
r.BalanceAlerts[i].paymentInstrumentId;
1420          session.svaalert[i].threshold =
r.BalanceAlerts[i].threshold / 100;
1421          session.svaalert[i].active =
r.BalanceAlerts[i].active;
1422          session.svaalert[i].onlyTransition =
r.BalanceAlerts[i].onlyTransition;
1423          session.svaalert[i].templateName =
r.BalanceAlerts[i].templateName;
1424      }
1425  }
1426
1427  /**
1428     @function
1429
1430     @param response This is the XML document/JSON object that
is returned by the AJAX call made by the MobiliserClient.
1431  */
1432  function loginParseXML(response) {
1433      if (response.sessionId !== null)
1434          session.sessionId = response.sessionId;
1435      if (response.sessionTimeoutSeconds !== null)
```



```
1436         sessionTimeoutWindow = response.sessionTimeoutSeconds
* 1000;
1437
1438         if (response.customer !== null) {
1439             session.customer = response.customer;
1440         }
1441     }
1442
1443     /**
1444     @function
1445
1446     @param response This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1447     */
1448     function fbssloginParseXML(response) {
1449         //console.log("fbsslogin:" +
JSON.stringify(response));
1450         if (response.sessionId !== null)
1451             session.sessionId = response.sessionId;
1452         if (response.sessionTimeoutSeconds !== null)
1453             sessionTimeoutWindow = response.sessionTimeoutSeconds
* 1000;
1454
1455         if (response.user !== null) {
1456             session.customer = response.user;
1457         }
1458     }
1459
1460     /**
1461     @function
1462
1463     @param response This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1464     */
```

Money Mobiliser Core API

```
1465     function getRsaPublicKeyParseXML(response) {
1466         if (response.sessionId !== null)
1467             session.sessionId = response.sessionId;
1468         if (response.sessionTimeoutSeconds !== null)
1469             sessionTimeoutWindow = response.sessionTimeoutSeconds *
1000;
1470
1471         if (response.user !== null) {
1472             //session.customer.customerId = response.user.id;
1473             //session.customer.orgUnitId =
response.user.orgUnitId;
1474             //session.customer.blacklist =
response.user.blackListReason;
1475             //session.customer.active = response.user.active;
1476             //session.customer.test = response.user.test;
1477             //session.customer.displayName =
response.user.displayName;
1478             //session.customer.riskCategoryId =
response.user.riskCategoryId;
1479             //session.customer.typeId =
response.user.customerTypeId;
1480             //session.customer.cancelId =
response.user.cancellationReasonId;
1481             //session.customer.modeId =
response.user.txnReceiptModeId;
1482         }
1483     }
1484
1485     /**
1486     @function
1487
1488     @param response This is the XML document/JSON object that
is returned by the AJAX call made by the MobiliserClient.
1489     */
1490     function walletParseXML(response) {
1491         var j = 1, k = 0;
```

```
1492     session.accounts = [];  
1493     session.totalaccounts = 0;  
1494  
1495     for (var i=0; i<response.walletEntries.length; i++) {  
1496         if (response.walletEntries[i].creditCard != null)  
1497         {  
1498             session.accounts[j] = new Accounts();  
1499             session.accounts[j].walletId =  
1500             response.walletEntries[i].id;  
1501             session.accounts[j].creditPriority =  
1502             response.walletEntries[i].creditPriority;  
1503             session.accounts[j].debitPriority =  
1504             response.walletEntries[i].debitPriority;  
1505             session.accounts[j].pIID =  
1506             response.walletEntries[i].paymentInstrumentId;  
1507             session.accounts[j].info =  
1508             response.walletEntries[i].alias;  
1509             session.accounts[j].type = "CreditCard";  
1510             session.accounts[j].pIClass = 'creditcard';  
1511             session.accounts[j].no =  
1512             response.walletEntries[i].creditCard.cardNumber;  
1513             session.accounts[j].acctHolderName =  
1514             response.walletEntries[i].creditCard.cardHolderName;  
1515             session.accounts[j].acctNumber = '';  
1516             session.accounts[j].displayNumber =  
1517             response.walletEntries[i].creditCard.displayNumber;  
1518             session.accounts[j].extra1 =  
1519             response.walletEntries[i].creditCard.cardType;  
1520             session.accounts[j].extra2 =  
1521             response.walletEntries[i].creditCard.securityNumber;  
1522             session.accounts[j].extra3 =  
1523             response.walletEntries[i].creditCard.yearExpiry;  
1524             session.accounts[j].extra4 =  
1525             response.walletEntries[i].creditCard.monthExpiry;  
1526             j++;  
1527         } else if (response.walletEntries[i].bankAccount !=  
1528         null && response.walletEntries[i].bankAccount.status == 0) {  
1529             session.accounts[j] = new Accounts();
```

Money Mobiliser Core API

```
1516         session.accounts[j].walletId =
response.walletEntries[i].id;

1517         session.accounts[j].creditPriority =
response.walletEntries[i].creditPriority;

1518         session.accounts[j].debitPriority =
response.walletEntries[i].debitPriority;

1519         session.accounts[j].pIID =
response.walletEntries[i].paymentInstrumentId;

1520         session.accounts[j].info =
response.walletEntries[i].alias;

1521         session.accounts[j].type = "BankAccount";

1522         session.accounts[j].pIClass = 'bank';

1523         session.accounts[j].no =
response.walletEntries[i].bankAccount.accountNumber;

1524         session.accounts[j].acctHolderName =
response.walletEntries[i].bankAccount.accountHolderName;

1525         session.accounts[j].acctNumber = '';

1526         session.accounts[j].displayNumber =
response.walletEntries[i].bankAccount.displayNumber;

1527         session.accounts[j].extra1 =
response.walletEntries[i].bankAccount.bankCode;

1528         session.accounts[j].extra2 =
response.walletEntries[i].bankAccount.branchCode;

1529         session.accounts[j].extra3 =
response.walletEntries[i].bankAccount.bankName;

1530         session.accounts[j].extra4 =
response.walletEntries[i].bankAccount.bankCountry;

1531         j++;

1532     } else if (response.walletEntries[i].sva !== null)
{

1533         session.accounts[0] = new Accounts();

1534         session.accounts[0].walletId =
response.walletEntries[i].id;

1535         session.accounts[0].creditPriority =
response.walletEntries[i].creditPriority;

1536         session.accounts[0].debitPriority =
response.walletEntries[i].debitPriority;

1537         session.accounts[0].pIID =
response.walletEntries[i].paymentInstrumentId;
```

```

1538         session.accounts[0].info =
response.walletEntries[i].sva.creditBalance -
1539
response.walletEntries[i].sva.debitBalance;
1540         session.accounts[0].type = "SVA";
1541         session.accounts[0].pIClass = 'sva';
1542         session.accounts[0].no = '';
1543     } else if (response.walletEntries[i].offlineSva !==
null) {
1544         session.offlinesvas[k] = new Accounts();
1545         session.offlinesvas[k].walletId =
response.walletEntries[i].id;
1546         session.offlinesvas[k].creditPriority =
response.walletEntries[i].creditPriority;
1547         session.offlinesvas[k].debitPriority =
response.walletEntries[i].debitPriority;
1548         session.offlinesvas[k].pIId =
response.walletEntries[i].paymentInstrumentId;
1549         session.offlinesvas[k].info =
response.walletEntries[i].offlineSva.balanceAmount;
1550         session.offlinesvas[k].type = "OFFLINESVA";
1551         session.offlinesvas[k].pIClass = 'offlinesva';
1552         session.offlinesvas[k].no =
response.walletEntries[i].offlineSva.svaCode;
1553         session.offlinesvas[k].displayNumber =
response.walletEntries[i].offlineSva.svaCode;
1554         k++;
1555     }
1556 }
1557     session.totalaccounts = j;
1558 }
1559
1560 /**
1561     @function
1562
1563     @param response This is the XML document/JSON object that
is returned by the AJAX call made by the MobiliserClient.

```

Money Mobiliser Core API

```
1564     */
1565     function txnsParseXML(response) {
1566         var max = response.transactions.length < 10 ?
response.transactions.length : 10;
1567
1568         for (var i=0; i<max; i++) {
1569             session.transaction[i] = new Transaction();
1570
1571             if (response.transactions[i].id !== null)
1572                 session.transaction[i].transactionId =
response.transactions[i].id;
1573
1574             if (response.transactions[i].creationDate !== null)
1575                 session.transaction[i].transactionDate =
epoch2UTC(response.transactions[i].creationDate);
1576
1577             if (response.transactions[i].amount !== null)
1578                 session.transaction[i].amount =
response.transactions[i].amount.value / 100;
1579
1580             if (response.transactions[i].useCase !== null)
1581                 session.transaction[i].usecase =
response.transactions[i].useCase;
1582
1583             switch (session.transaction[i].usecase) {
1584                 case 174:
1585                 case 193:
1586                 case 194:
1587                 case 191:
1588                 case 202:
1589                 for (var j=0; j<session.totalaccounts; j++)
1590                 {
1591                     if (response.transactions[i].payerPiId !==
null &&
```

```

1591         response.transactions[i].payerPiId ==
session.accounts[j].pIID)
1592             session.transaction[i].amount *= -1;
1593         }
1594         break;
1595         case 173:
1596         case 161:
1597         case 500:
1598         case 502:
1599             session.transaction[i].amount *= -1;
1600         break;
1601     }
1602     session.transaction[i].amount =
session.transaction[i].amount.toFixed(2);
1603 }
1604
1605     return max;
1606 }
1607
1608 /**
1609     @function
1610
1611     @param response This is the XML document/JSON object that
is returned by the AJAX call made by the MobiliserClient.
1612 */
1613     function txnDetailParse(response) {
1614         if (response.transaction !== null) {
1615             var tranDate = new
Date(response.transaction.creationDate);
1616             var txn = new Object();
1617             txn.Date = tranDate.getFullYear()
+'-'+pad2(tranDate.getMonth()+1)+'-'+pad2(tranDate.getDate());
1618             txn.Time = pad2(tranDate.getHours())
+':'+pad2(tranDate.getMinutes())+':'+pad2(tranDate.getSeconds());

```

Money Mobiliser Core API

```
1619         txn.Type =
getUseCaseName(response.transaction.useCase);
1620         txn.Error = response.transaction.errorCode;
1621         txn.Name = (response.transaction.payerDisplayName ==
session.customer.displayName) ?
1622         response.transaction.payeeDisplayName :
1623         response.transaction.payerDisplayName;
1624         txn.Reference = response.transaction.authCode;
1625         txn.Details = "";
1626         if (response.transaction.text != null)
1627             txn.Details = response.transaction.text;
1628         txn.Amount = response.transaction.amount.value/100;
1629         txn.Amount = txn.Amount.toFixed(2);
1630         txn.Fee = (response.transaction.payerDisplayName ==
session.customer.displayName) ?
1631             (response.transaction.payerAmount.value/100) -
txn.Amount :
1632             txn.Amount -
(response.transaction.payeeAmount.value/100);
1633         txn.Fee = txn.Fee.toFixed(2);
1634         return txn;
1635     }
1636 }
1637
1638 /**
1639  @function
1640
1641  @param response This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1642  */
1643  function couponparseXML(response) {
1644      session.coupons = [];
1645      if (response.coupon) {
```



```
1646         for (var i=0; i<response.coupon.length; i++) {
1647             session.coupons[i] = new Coupon();
1648             session.coupons[i].id = response.coupon[i].id;
1649             session.coupons[i].customerId =
response.coupon[i].customerId;
1650             if(session.coupons[i].couponType != null)
1651                 {
1652                     session.coupons[i].couponType.id =
response.coupon[i].couponType.id;
1653                     session.coupons[i].couponType.name =
response.coupon[i].couponType.name;
1654                     session.coupons[i].couponType.purchasePrice =
response.coupon[i].couponType.purchasePrice ?
(response.coupon[i].couponType.purchasePrice/100) : 0;
1655                     session.coupons[i].couponType.purchaseCurrency
= response.coupon[i].couponType.purchaseCurrency ?
response.coupon[i].couponType.purchaseCurrency : '';
1656                     if(response.coupon[i].couponType.description !=
null)
1657                         {
1658                             session.coupons[i].couponType.description.caption =
response.coupon[i].couponType.description.caption;
1659                             session.coupons[i].couponType.description.imageUrl =
response.coupon[i].couponType.description.imageUrl;
1660                             session.coupons[i].couponType.description.thumbnailUrl =
response.coupon[i].couponType.description.thumbnailUrl;
1661                         }
1662                     }
1663                     session.coupons[i].status =
response.coupon[i].status;
1664                     session.coupons[i].serialNumber =
response.coupon[i].serialNumber;
1665                     session.coupons[i].validTo =
getDateFormatOfSring(response.coupon[i].validTo);
1666                     session.coupons[i].code =
response.coupon[i].code;
```

```
1667         session.coupons[i].views =
response.coupon[i].views;

1668         session.coupons[i].uses =
response.coupon[i].uses;

1669     }

1670 }

1671 }

1672

1673 /**
1674  @function
1675
1676  @param response This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1677  */
1678  function getRootCategoriesParseXML(response) {
1679      session.categories = [];
1680      if(response.category != null) {
1681          for (var i=0; i<response.category.length; i++) {
1682              session.categories[i] = new Category();
1683              session.categories[i].id =
response.category[i].id;
1684              session.categories[i].internalName =
response.category[i].internalName;
1685              session.categories[i].caption =
response.category[i].caption;
1686              session.categories[i].priority =
response.category[i].priority;
1687              session.categories[i].noOfChildCategories =
response.category[i].child;
1688              session.categories[i].parent =
response.category[i].parent;
1689              session.categories[i].valid = false;
1690          }
1691          return true;
1692      } else
1693          return false;
```

```
1694     }
1695
1696     /**
1697     @function
1698
1699     @param response This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1700     */
1701     function getChildCategoriesParseXML(response) {
1702         session.categories = [];
1703         if(response.category != null) {
1704             for (var i=0; i<response.category.length; i++) {
1705                 session.categories[i] = new Category();
1706                 session.categories[i].id =
response.category[i].id;
1707                 session.categories[i].internalName =
response.category[i].internalName;
1708                 session.categories[i].caption =
response.category[i].caption;
1709                 session.categories[i].priority =
response.category[i].priority;
1710                 session.categories[i].noOfChildCategories =
response.category[i].child;
1711                 session.categories[i].parent =
response.category[i].parent;
1712                 session.categories[i].valid = false;
1713             }
1714             return true;
1715         } else
1716             return false;
1717     }
1718
1719     /**
1720     @function
1721
```

Money Mobiliser Core API

```
1722 @param response This is the XML document/JSON object that is
1723 */
1724 function getCouponTypesofCategoryParseXML(response) {
1725     session.couponTypes = [];
1726     if(response.couponType != null) {
1727         for (var i=0; i<response.couponType.length; i++) {
1728             session.couponTypes[i] = {};
1729             session.couponTypes[i].type =
1730 response.couponType[i];
1731             session.couponTypes[i].location = null;
1732             session.couponTypes[i].distance = null;
1733             /*
1734             session.couponTypes[i] = new CouponType();
1735             session.couponTypes[i].description = new
1736 CouponTypeDescription();
1737
1738             session.couponTypes[i].id =
1739 response.couponType[i].id;
1740             session.couponTypes[i].name =
1741 response.couponType[i].name;
1742             session.couponTypes[i].purchasePrice =
1743 response.couponType[i].purchasePrice ?
1744 (response.couponType[i].purchasePrice/100) : 0;
1745             session.couponTypes[i].purchaseCurrency =
1746 response.couponType[i].purchaseCurrency ?
1747 response.couponType[i].purchaseCurrency : '';
1748             session.couponTypes[i].maxViews =
1749 response.couponType[i].maxViews;
1750             session.couponTypes[i].maxUses =
1751 response.couponType[i].maxUses;
1752             session.couponTypes[i].isRestorableByUser =
1753 response.couponType[i].isRestorableByUser;
1754             if(response.couponType[i].description != null)
1755 {
1756             session.couponTypes[i].description.mimeType =
1757 response.couponType[i].description.mimeType;
```

```

1745         session.couponTypes[i].description.caption =
response.couponType[i].description.caption;
1746         session.couponTypes[i].description.content =
response.couponType[i].description.content;
1747         session.couponTypes[i].description.thumbnailUrl
= response.couponType[i].description.thumbnailUrl;
1748         session.couponTypes[i].description.imageUrl =
response.couponType[i].description.imageUrl;
1749     }
1750 */
1751     }
1752 }
1753 }
1754
1755
1756 /**
1757 @function
1758
1759 @param response This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
1760 */
1761 function couponTypesByTagsparseXML(response) {
1762     session.couponTypes = [];
1763     if(response.couponType != null) {
1764         for (var i=0; i<response.couponType.length; i++) {
1765             session.couponTypes[i] = {};
1766             session.couponTypes[i].type =
response.couponType[i];
1767             session.couponTypes[i].location = null;
1768             session.couponTypes[i].distance = null;
1769         /*
1770             session.couponTypes[i] = new CouponType();
1771             session.couponTypes[i].id =
response.couponType[i].id;

```

```
1772         session.couponTypes[i].name =
response.couponType[i].name;

1773         session.couponTypes[i].purchasePrice =
response.couponType[i].purchasePrice ?
(response.couponType[i].purchasePrice/100) : 0;

1774         session.couponTypes[i].purchaseCurrency =
response.couponType[i].purchaseCurrency ?
response.couponType[i].purchaseCurrency : '';

1775         session.couponTypes[i].maxViews =
response.couponType[i].maxViews;

1776         session.couponTypes[i].maxUses =
response.couponType[i].maxUses;

1777         session.couponTypes[i].isRestorableByUser =
response.couponType[i].isRestorableByUser;

1778         if(response.couponType[i].description != null)
{

1779             session.couponTypes[i].description.mimetype =
response.couponType[i].description.mimetype;

1780             session.couponTypes[i].description.caption =
response.couponType[i].description.caption;

1781             session.couponTypes[i].description.content =
response.couponType[i].description.content;

1782             session.couponTypes[i].description.thumbnailUrl =
response.couponType[i].description.thumbnailUrl;

1783             session.couponTypes[i].description.imageUrl =
response.couponType[i].description.imageUrl;

1784         }

1785     */

1786     }

1787 }

1788 }

1789

1790 function getCategoryTreeBackParseXML(response) {
1791     var categories = response.rootCategories;
1792     if (categories && categories.length) {
1793         for (var i=0; i<session.categories.length; i++) {
1794             // Used to retrieve an array from category tree,
which is having the id value, matching with category id value.
```

```

1795         var resp = getArrayOfObjects(categories, "id",
session.categories[i].id);
1796         if(resp.length) {
1797             session.categories[i].valid = true;
1798             // here resp.length is used to identify an
object availability and (resp[0]['child'] != null) is used to check
an object is not null.
1799             if(resp[0]['child'] != null)
1800                 session.categories[i].noOfChildCategories =
resp[0]['child'].length;
1801             else
1802                 session.categories[i].noOfChildCategories =
0;
1803
1804         }
1805     }
1806     return true;
1807 } else
1808     return false;
1809 }
1810
1811 /**
1812     @function
1813
1814     @param usecase This is usecase number in the AJAX call
response
1815     */
1816     function getUseCaseName(usecase) {
1817         var strname = "";
1818         usecase = usecase.toString();
1819         switch(usecase) {
1820             case '202': strname = "Demand4Payment"; break;
1821             case '194': strname = "Request Money"; break;
1822             case '193': strname = "P2P"; break;

```

Money Mobiliser Core API

```
1823         case '191': strname = "Pay Bill"; break;
1824         case '174': strname = "Send Voucher"; break;
1825         case '173': strname = "Airtime Topup"; break;
1826         case '161': strname = "SVA Unload"; break;
1827         case '160': strname = "SVA Load"; break;
1828         case '500': strname = "NFC Transaction"; break;
1829         case '501': strname = "OfflineSVA Load"; break;
1830         case '502': strname = "OfflineSVA Unload"; break;
1831         case '503': strname = "P2P NFC Transaction"; break;
1832         case '504': strname = "NFC Transaction"; break;
1833     }
1834     return strname;
1835 }
1836
1837 /**
1838     @function
1839
1840     @param r This is the XML document/JSON object that is
1841     returned by the AJAX call made by the MobiliserClient.
1842     */
1843     function getErrorMessage(r) {
1844         var errValue = r.Status.value;
1845         var errCode = r.Status.code.toString();
1846         if(errValue !== null) {
1847             if (errCode == '329' || errCode == '201')
1848                 return "Check your username or password and try
1849                 again!";
1850             else if (errCode == '302')
1851                 return "Your account is blocked! Please contact
1852                 service provider.";
1853             else if(errCode == '2517')
1854                 return "Payer has no valid wallet for this
1855                 transaction";
1856             else if(errCode == '2507')
```



```

1853         return "Payee has no valid wallet for this
transaction";
1854     else
1855         return r.Status.value;
1856     } else {
1857         var msg = '';
1858         switch (errCode) {
1859             case '1002': msg = "INVALID MSISDN: Indicates that (one
of) the specified MSISDN(s) is invalid. MSISDNs always have to
specified as +countryCodenetCodenumber"; break;
1860             case '1003': msg = "UNKNOWN TRANSACTION: Indicates
that the (or a) transaction specified in the request does not
exist."; break;
1861             case '1004': msg = "INVALID TRANSACTION: Indicates
that the (or a) transaction specified in the request cannot be used/
considered in the requested context."; break;
1862             case '1005': msg = "INVALID TRANSACTION STATUS:
Indicates that the (or a) transaction specified in the request cannot
be used/considered due to its status."; break;
1863             case '1006': msg = "INACTIVE CUSTOMER: Indicates that
the (or a) customer specified in the request cannot be used/
considered because he is inactive"; break;
1864             case '1008': msg = "BLACKLISTED CUSTOMER"; break;
1865             case '1050': msg = "The transaction is locked by
another process and cannot be handled right now"; break;
1866             case '1112': msg = "INACTIVE USER: Indicates that the
customer's status is inactive - see database
CUSTOMER_STATUS.BOL_IS_ACTIVE."; break;
1867             case '1114': msg = "CUSTOMER BLOCKED: Indicates that
the customer is currently blocked due to the number of times wrong
credentials have been specified."; break;
1868             case '1115': msg = "BLACKLISTED USER: Indicates that
the customer is blacklisted - see database
CUSTOMER.ID_BLACKLISTREASON."; break;
1869             case '1121': msg = "CREDENTIALS EXPIRED: Indicates
that the specified credentials have expired"; break;
1870             case '1250': msg = "TRANSACTION_EXPIRED: The
transaction was pending in initial state too long and invalidated
automatically."; break;
1871             case '1301': msg = "SESSION ACTIVE WARN: Indicates that
a login failed because the customer has still an active session -

```

```
this prevents logging in multiple times with just one customer.";
break;

1872         case '2001': msg = "CREDIT LIMIT EXCEEDED: Indicates
that the credit limit has been exceeded."; break;

1873         case '2002': msg = "DEBIT LIMIT EXCEEDED: Indicates
that the debit limit has been exceeded."; break;

1874         case '201': msg = "MINIMUM CREDENTIAL LENGTH:
Indicates that the credential (to create or update, presumably)
exceeds the minimum credential length."; break;

1875         case '2011': msg = "RESERVATION EXPIRED: Indicates
that the SVA reservation (to capture/cancel/...) expired."; break;

1876         case '202': msg = "MAXIMUM CREDENTIAL LENGTH:
Indicates that the credential (to create or update, presumably)
exceeds the maximum credential length."; break;

1877         case '203': msg = "CREDENTIAL REPEAT: Indicates that
the credential (to create or update, presumably) is the same as any
of the n previously used credentials."; break;

1878         case '204': msg = "CREDENTIAL WEAK ASCENDING:
Indicates that the credential (to create or update, presumably)
contains a block of ascending characters."; break;

1879         case '205': msg = "CREDENTIAL WEAK EQUALITY: Indicates
that the credential (to create or update, presumably) contains a
block of equal characters."; break;

1880         case '206': msg = "CREDENTIAL WEAK DESCENDING:
Indicates that the credential (to create or update, presumably)
contains a block of descending characters."; break;

1881         case '207': msg = "CREDENTIAL EQUALS IDENTIFICATION:
Indicates that the specified credential is the same as one of the
customers identifications."; break;

1882         case '208': msg = "CREDENTIAL REGEX MATCH: Indicates
that the credential does or does not match one of the security
policies' regular expressions and thus is invalid."; break;

1883         case '2201': msg = "UNKNOWN TARGET: Indicates that the
target customer is unknown."; break;

1884         case '2202': msg = "INVALID TARGET: Indicates that the
target customer is invalid, i.e. not allowed/able to participate in
the operation."; break;

1885         case '2214': msg = "FLOATING AMOUNT MISSING";
break;

1886         case '2215': msg = "FLOATING AMOUNT FALLS BELOW MIN
PRICE"; break;

1887         case '2216': msg = "FLOATING AMOUNT EXCEEDS MAX PRICE";
break;
```

```

1888         case '2222': msg = "IN BUSY: Indicates that the IN is
currently busy and cannot handle more requests."; break;
1889         case '2231': msg = "AIRTIME LIMIT EXCEEDED"; break;
1890         case '2232': msg = "CO-USER LIMIT EXCEEDED"; break;
1891         case '2259': msg = "VOUCHER SALE HANDLER FAILURE";
break;
1892         case '2299': msg = "TRANSACTION FAILED PERMANENTLY";
break;
1893         case '2405': msg = "NO COMM WITH MSG-GW: Send error,
indicates that the Notification Services are not going to send the
specified message (for whatever reason)"; break;
1894         case '2507': msg = "PAYEE PI UNKNOWN"; break;
1895         case '2511': msg = "UNKNOWN PAYER"; break;
1896         case '2512': msg = "INVALID PAYER"; break;
1897         case '2517': msg = "PAYER PI UNKNOWN"; break;
1898         case '2519': msg = "AUTHORIZATION EXPIRED"; break;
1899         case '2521': msg = "AUTHENTICATION REQUIRED: Indicates
that the participant authentication has been initialized, but the
authentication is asynchronous"; break;
1900         case '2522': msg = "AUTHENTIVATION WRONG: Indicates
that the supplied authentication was wrong"; break;
1901         case '2525': msg = "MANUAL DECISION REQUIRED:
Indicates that the transaction processing has been stopped and a
manual decission is needed"; break;
1902         case '2535': msg = "ERROR_PAYEE_OPEN_TRANSACTION";
break;
1903         case '2536': msg = "ERROR_PAYER_OPEN_TRANSACTION";
break;
1904         case '2537': msg = "ERROR_NO_OPEN_TRANSACTION";
break;
1905         case '2541': msg = "PAYER DENIED TRANSACTION";
break;
1906         case '2542': msg = "PAYER NOT REACHABLE"; break;
1907         case '2561': msg = "PAYEE ABSOLUTE LIMIT EXCEEDED:
Indicates that the payee's absolute credit limit has been exceeded.";
break;
1908         case '2562': msg = "PAYER ABSOLUTE LIMIT EXCEEDED:
Indicates that the payer's absolute debit limit has been exceeded.";
break;

```

Money Mobiliser Core API

```
1909         case '2563': msg = "PAYEE MONTHLY LIMIT EXCEEDED:
Indicates that the payee's monthly credit limit has been exceeded.";
break;

1910         case '2564': msg = "PAYER MONTHLY LIMIT EXCEEDED:
Indicates that the payer's monthly debit limit has been exceeded.";
break;

1911         case '2565': msg = "PAYEE DAILY LIMIT EXCEEDED:
Indicates that the payee's daily credit limit has been exceeded.";
break;

1912         case '2566': msg = "PAYER DAILY LIMIT EXCEEDED:
Indicates that the payer's daily debit limit has been exceeded.";
break;

1913         case '2571': msg = "PAYEE WALLET ABSOLUTE LIMIT
EXCEEDED: Indicates that the payee's wallet absolute credit limit has
been exceeded."; break;

1914         case '2572': msg = "PAYER WALLET ABSOLUTE LIMIT
EXCEEDED: Indicates that the payer's wallet absolute debit limit has
been exceeded."; break;

1915         case '2573': msg = "PAYEE WALLET MONTHLY LIMIT
EXCEEDED: Indicates that the payee's wallet monthly credit limit has
been exceeded."; break;

1916         case '2574': msg = "PAYER WALLET MONTHLY LIMIT
EXCEEDED: Indicates that the payer's wallet monthly debit limit has
been exceeded."; break;

1917         case '2575': msg = "PAYEE WALLET DAILY LIMIT EXCEEDED:
Indicates that the payee's wallet daily credit limit has been
exceeded."; break;

1918         case '2576': msg = "PAYER WALLET DAILY LIMIT EXCEEDED:
Indicates that the payer's wallet daily debit limit has been
exceeded."; break;

1919         case '2581': msg = "RESTRICTION PAYEE TRANSACTION
AMOUNT: Indicates that a transaction restriction restricts the payee
to participate in this transaction because the transaction amount is
higher than allowed"; break;

1920         case '2582': msg = "RESTRICTION PAYER TRANSACTION
AMOUNT: Indicates that a transaction restriction restricts the payer
to participate in this transaction because the transaction amount is
higher than allowed"; break;

1921         case '2583': msg = "RESTRICTION PAYEE TRANSACTION
COUNT: Indicates that a restriction restricts the payee to
participate in this transaction because the restricted number of
transaction per timeframe is exceeded"; break;

1922         case '2584': msg = "RESTRICTION PAYER TRANSACTION
COUNT: Indicates that a restriction restricts the payer to
participate in this transaction because the restricted number of
transaction per timeframe is exceeded"; break;
```

1923 case '2585': msg = "RESTRICTION PAYEE TRANSACTION SUM:
Indicates that an transaction restrictions restricts the payee to
participate in this transaction because the restricted transaction
sum per timeframe is exceeded"; break;

1924 case '2586': msg = "RESTRICTION PAYER TRANSACTION SUM:
Indicates that an transaction restrictions restricts the payee to
participate in this transaction because the restricted transaction
sum per timeframe is exceeded"; break;

1925 case '2587': msg = "RESTRICTION PAYEE TRANSACTION SUM
MIN"; break;

1926 case '2588': msg = "RESTRICTION PAYER TRANSACTION SUM
MIN"; break;

1927 case '2589': msg = "RESTRICTION INDIVIDUAL"; break;

1928 case '2604': msg = "PAYER PI IN INVALID STATE";
break;

1929 case '2605': msg = "PAYER PI NOT ALLOWED"; break;

1930 case '2607': msg = "PAYER PI DAILY LIMIT EXCEEDED:
Indicates that the payer's payment instrument daily debit limit has
been exceeded."; break;

1931 case '2608': msg = "PAYER PI MONTHLY LIMIT EXCEEDED:
Indicates that the payer's payment instrument monthly debit limit has
been exceeded."; break;

1932 case '2611': msg = "Balance of Payer's account is not
enough for this transaction amount!"; break;

1933 case '2622': msg = "PAYER PI CAPTURE CANCEL FULL
AMOUNT: Indicates that the payers payment instrument only allows
cancelling the complete capture amount."; break;

1934 case '2631': msg = "Payer's account is BLOCKED";
break;

1935 case '2650': msg = "PICKUP CODE NOT FOUND: Indicates
that the transaction has no pickup code attached."; break;

1936 case '2651': msg = "PICKUP MAX TRIES EXCEEDED:
Indicates that the maximum number of tries has been exceeded.";
break;

1937 case '2652': msg = "PICKUP CODE INVALID: Indicates that
the pickup code is invalid."; break;

1938 case '2653': msg = "PICKUP ATTRIBUTE NOT UNIQUE:
Indicates that the pickup code attribute attached to the transaction
is not unique."; break;

1939 case '2654': msg = "PICKUP INTERNAL VOUCHER NOT FOUND:
Indicates that the internal voucher associated with the transaction
could not be found."; break;

Money Mobiliser Core API

```
1940         case '2655': msg = "PICKUP INTERNAL VOUCHER TXN NOT
FOUND: Indicates that the internal voucher transaction associated
with the given transaction could not be found."; break;

1941         case '2661': msg = "WALLET ENTRY INVALID ALIAS:
Indicates that a wallet entry exists for the same customer with a
different payment instrument."; break;

1942         case '2662': msg = "WALLET ENTRY INVALID PRIORITY:
Indicates that a wallet entry exists for the same customer having the
same debit or credit priority"; break;

1943         case '2704': msg = "PAYEE PI IN INVALID STATE";
break;

1944         case '2705': msg = "PAYEE PI NOT ALLOWED"; break;

1945         case '2707': msg = "PAYEE PI DAILY LIMIT EXCEEDED:
Indicates that the payee's payment instrument daily credit limit has
been exceeded."; break;

1946         case '2708': msg = "PAYEE PI MONTHLY LIMIT EXCEEDED:
Indicates that the payee's payment instrument monthly credit limit
has been exceeded."; break;

1947         case '2709': msg = "PAYEE PI ABSOLUTE LIMIT EXCEEDED:
Indicates that the payee's payment instrument absolute credit limit
has been exceeded."; break;

1948         case '2711': msg = "PAYEE PI LIMIT HIT"; break;

1949         case '2731': msg = "PAYEE PI BLOCKED"; break;

1950         case '2811': msg = "DISCONNECTED BY CUSTOMER";
break;

1951         case '2907': msg = "CUSTOMER MOBILE BUSY"; break;

1952         case '2908': msg = "NO ANSWER FROM CUSTOMER MOBILE";
break;

1953         case '2916': msg = "COULD NOT REACH CUSTOMERS MOBILE";
break;

1954         case '2920': msg = "3 TIMES WRONG PIN"; break;

1955         case '2921': msg = "3 TIMES NO RESPONSE"; break;

1956         case '2943': msg = "WRONG OR INVALID PIN"; break;

1957         case '2945': msg = "AUTHENTICATION DECLINED";
break;

1958         default: msg = "Unknown Error!"; break;

1959     }

1960     return msg;

1961 }
```

```

1962     }
1963
1964     function epoch2UTC(epoch) {
1965         var utcDate = new Date(epoch);
1966         return utcDate.getFullYear()+'-'+pad2(utcDate.getMonth()+1)+'-'+pad2(utcDate.getDate())+
1967             'T'+pad2(utcDate.getHours())
1968         +':'+'+pad2(utcDate.getMinutes())+':'+'+pad2(utcDate.getSeconds());
1969     }
1970
1971     /**
1972     * @description callback handler for receiving existing
1973     alerts
1974     */
1975
1976     var getExistingAlertsBack = function(r) {
1977         hideProgress();
1978
1979         var rCode = getStatusCode(r);
1980         if (rCode == '0') {
1981             //Parse
1982             session.transit_value = parseExistingAlerts(r);
1983             //Subsequent Server Calls
1984             fetchActiveAlertNotificationMessage();
1985         } else {
1986             _alertBox(Str_alert_failgetexistingalerts +
1987                 getErrorMessage(r));
1988         }
1989         appstate = trans.TRAN_IDLE;
1990     };
1991
1992     /**
1993     * @description Parsing the Alerts for Alert List
1994     */
1995
1996     function parseExistingAlerts(r){

```

Money Mobiliser Core API

```
1992     var customerAlerts = new Array();
1993     if(r.customerAlert != null) {
1994         for(var i = 0;i < r.customerAlert.length;i++) {
1995             var ca = new CustomerAlert();
1996             ca.id = r.customerAlert[i].id;
1997             ca.customerId = r.customerAlert[i].customerId;
1998             ca.alertTypeId =
1999             r.customerAlert[i].alertTypeId;
2000             ca.active = r.customerAlert[i].active;
2001             ca.created = r.customerAlert[i].created;
2002             ca.alertDataList = [];
2003             if(r.customerAlert[i].alertDataList != null
2004                 && r.customerAlert[i].alertDataList !=
2005                 'undefined'
2006                 &&
2007                 r.customerAlert[i].alertDataList.alertData != null
2008                 &&
2009                 r.customerAlert[i].alertDataList.alertData != 'undefined') {
2010                 for(var j = 0;j <
2011                 r.customerAlert[i].alertDataList.alertData.length;j++) {
2012                     var ad = new AlertData();
2013                     ad.id =
2014                     r.customerAlert[i].alertDataList.alertData[j].id;
2015                     ad.customerAlertId =
2016                     r.customerAlert[i].alertDataList.alertData[j].customerAlertId;
2017                     ad.key =
2018                     r.customerAlert[i].alertDataList.alertData[j].key;
2019                     ad.value =
2020                     r.customerAlert[i].alertDataList.alertData[j].value;
2021                     ca.alertDataList.push(ad);
2022                 }
2023             }
2024         }
2025     }
2026     ca.alertDataList = [];
2027     ca.contactPointList = [];
```



```

2018         if(r.customerAlert[i].contactPointList != null &&
r.customerAlert[i].contactPointList != 'undefined'
2019             &&
r.customerAlert[i].contactPointList.contactPoint != null
2020             &&
r.customerAlert[i].contactPointList.contactPoint != 'undefined') {
2021         for(var k = 0;k <
r.customerAlert[i].contactPointList.contactPoint.length;k++) {
2022         if(r.customerAlert[i].contactPointList.contactPoint[k].otherIdentif
ication) {
2023             var oi = new OtherIdentification();
2024             oi.id =
r.customerAlert[i].contactPointList.contactPoint[k].otherIdentifica
tion.id;
2025             oi.customerId =
r.customerAlert[i].contactPointList.contactPoint[k].otherIdentifica
tion.customerId;
2026             oi.type =
r.customerAlert[i].contactPointList.contactPoint[k].otherIdentifica
tion.type;
2027             oi.identification =
r.customerAlert[i].contactPointList.contactPoint[k].otherIdentifica
tion.identification;
2028             oi.nickname =
r.customerAlert[i].contactPointList.contactPoint[k].otherIdentifica
tion.nickname;
2029             oi.provider =
r.customerAlert[i].contactPointList.contactPoint[k].otherIdentifica
tion.provider;
2030             oi.status =
r.customerAlert[i].contactPointList.contactPoint[k].otherIdentifica
tion.status;
2031             oi.active =
r.customerAlert[i].contactPointList.contactPoint[k].otherIdentifica
tion.active;
2032             ca.contactPointList.push(oi);
2033         } else if
(r.customerAlert[i].contactPointList.contactPoint[k].identification
) {
2034             var iden = new Identification();

```

Money Mobiliser Core API

```
2035             iden.id =
r.customerAlert[i].contactPointList.contactPoint[k].identification.
id;

2036             iden.customerId =
r.customerAlert[i].contactPointList.contactPoint[k].identification.
customerId;

2037             iden.type =
r.customerAlert[i].contactPointList.contactPoint[k].identification.
type;

2038             iden.identification =
r.customerAlert[i].contactPointList.contactPoint[k].identification.
identification;

2039             ca.contactPointList.push(iden);
2040         }
2041     }
2042 }
2043     customerAlerts.push(ca);
2044 }
2045 }
2046 //return customerAlerts.sortBy('created');
2047     return customerAlerts.sort(dynamicSort('created'));
2048 }
2049
2050 /**
2051  * @description Call back handler for Deleting customer
alert
2052  */
2053     var deleteCustomerAlertBack = function(r) {
2054         hideProgress();
2055
2056         var rCode = getStatusCode(r);
2057         if (rCode == '0') {
2058             _alertBox(Str_msg_deletealert);
2059             $('#managealert').trigger('pageshow');
2060         } else {
```

```

2061         _alertBox(Str_msg_faildeletealert +
getErrorMessage(r));
2062     }
2063     appstate = trans.TRAN_IDLE;
2064 };
2065
2066 /**
2067  * @description This function layout the UI for other
identifications
2068  */
2069     function listOtherIdentifications(oIdentifications,
container) {
2070         $(container).empty();
2071         var identification = session.identification;
2072         var checked =
checkOtherIdentificationInSelectedAlert(identification.id);
2073         $(container).append("<input type=\"checkbox\" name=
\"aa_devices_\"+identification.id+\"\" id=
\"aa_devices_\"+identification.id+\"\" class=\"custom check_add_alert
\"\" +
2074             " value=\"\"+identification.id+\"\" "+
(checked?"checked":"")+" data-theme=\"g\" data-
identification='true'>" +
2075             "<label for=\"aa_devices_\"+identification.id
+\"\">\"+identification.identification+\"</label>");
2076
2077         if(oIdentifications) { //Check null or undefined
2078             for(i = 0;i < oIdentifications.length;i++){
2079                 var oi = oIdentifications[i];
2080                 checked =
checkOtherIdentificationInSelectedAlert(oi.id);
2081                 //console.log('Checked: ' + checked);
2082                 $(container).append("<input type=\"checkbox\"
name=\"aa_devices_\"+oi.id+\"\" id=\"aa_devices_\"+oi.id+\"\" class=
\"custom check_add_alert\"\" +
2083             " value=\"\"+oi.id+\"\" "+
(checked?"checked":"")+" data-theme=\"g\">" +

```

Money Mobiliser Core API

```
2084         "<label for=\"aa_devices_\"+oi.id
+\"\">"+oi.nickname+": "+oi.identification+"</label>");
2085     }
2086 }
2087     $('#aad_otheridentifications_wrap').trigger('create');
2088     $(container).trigger('create');
2089     $(container + '
input:checkbox').checkboxradio('refresh');
2090 }
2091
2092 /**
2093  * @description finds out if this OI is there in the alert
2094  * @return boolean depicting true if exists else false
2095  * @param oIID for an contact point
2096  */
2097     function checkOtherIdentificationInSelectedAlert(oIID) {
2098         var isExists = false;
2099         if(session.customerAlert != null) {
2100             if(session.customerAlert.contactPointList != null &&
session.customerAlert.contactPointList.length > 0) {
2101                 for(var i = 0;i <
session.customerAlert.contactPointList.length;i++) {
2102                     if(session.customerAlert.contactPointList[i].otherIdentification &&
session.customerAlert.contactPointList[i].otherIdentification.id ==
oIID) {
2103                         isExists = true;
2104                         break;
2105                     } else if
(session.customerAlert.contactPointList[i].identification &&
session.customerAlert.contactPointList[i].identification.id == oIID)
{
2106                         isExists = true;
2107                         break;
2108                     }
2109                 }
2109             }
2109         }
```

```
2110     }
2111 }
2112     return isExists;
2113 }
2114
2115 /**
2116  * @description Callback handler for Adding Alert
2117  */
2118     function createNewAlertBack(r) {
2119         hideProgress();
2120
2121         var rCode = getStatusCode(r);
2122         if (rCode == '0') {
2123             $('#aas_title').html(Str_txt_addalert);
2124             //Get the newly created alertId
2125             session.transit_value = r.customerAlertId;
2126             $.mobile.changePage("addAlertSuccess.html",
2127 page_opt);
2127         } else {
2128             _alertBox(Str_msg_failcreatealert +
2129 getErrorMessage(r));
2129         }
2130         appstate = trans.TRAN_IDLE;
2131     }
2132
2133 /**
2134  * @description Callback handler for Get Alert Detail
2135  service
2136  * @param r response data
2137  */
2138     function getAlertDetailsForEditBack(r) {
2139         hideProgress();
2139     }
```

Money Mobiliser Core API

```
2140         var rCode = getStatusCode(r);
2141         if(rCode == '0') {
2142             //Parse customer alert
2143             var ca = parseCustomerAlert(r);
2144
2145             session.customerAlert = ca;
2146             $('#addAlertDetails').attr("action", "edit");
2147             $('#txt_addalertydetail').text(Str_editalert);
2148
2149             var alertTypeId = ca.alertTypeId;
2150
2151             if(alertTypeId != null || alertTypeId != undefined)
2152             {
2153                 $('#btn_ma_addnewcontinue').attr("alertTypeId",
2154                 alertTypeId);
2155
2156                 var fieldArr =
2157                 alerttypefieldsmapping[alertTypeId];
2158                 $('#alertcontent').empty();
2159                 var alertDataList = ca.alertDataList;
2160                 for (i = 0; i < fieldArr.length; i++) {
2161                     var field = fieldArr[i];
2162                     var ad = findAlertDataInDataList(alertDataList,
2163                     field);
2164                     var val = '';
2165                     if(ad != null) {
2166                         val = ad.value;
2167                     } else if (field == "frequency") {
2168                         var cnt = ca.notifMaxCnt;
2169                         var rec = ca.notifMaxRecur;
2170                         if(cnt > 0) {
2171                             val = rec + "_" + cnt;
2172                         } else {
2173                             val = cnt;
2174                         }
2175                     }
2176                 }
2177             }
2178         }
2179     }
```

```

2170         }
2171     } else if (field == "msgtype") {
2172         val =
getNotificationTypeIdForAlertNotificationMessageId(ca.alertNotifica
tionMsgId);
2173     }
2174         //console.log("Value of field: " + field + " is
" + val);
2175         var contentlist = getFieldLiContent(field,
val);
2176         $('#alertcontent').append(contentlist);
2177     }
2178 }
2179     $('#addalerttitle').html(Str_edit + " " +
2180         getAlertPageTitle(alertTypeId) + " " +
Str_alert);
2181     $('#addAlertDetails .ui-
droplist').trigger('create');
2182
2183     $('#aa_msgtype_fs_wrap').trigger('create');
2184     $('#aa_msgtype_fs').trigger('create');
2185     $('#aa_msgtype_fs' + '
input:checkbox').checkboxradio('refresh');
2186
2187     $('#addAlertDetails .aa_frequency_field').collapsible();
2188     $('#aa_frequency_num').textinput();
2189
2190     $.mobile.changePage("addAlertDetails.html",
page_opt);
2191 } else {
2192     _alertBox(Str_msg_failgetalertdetail +
getErrorMessage(r));
2193 }
2194     appstate = trans.TRAN_IDLE;
2195 }

```

Money Mobiliser Core API

```
2196
2197  /**
2198   * @description Iterate and find the key in the list and
  returns the object.
2199   * @param alertDataList list of alert data objects
2200   * @param field fieldName to be searched
2201   * @return AlertData or null
2202   */
2203   function findAlertDataInDataList(alertDataList, field) {
2204       var ad = null;
2205       if(alertDataList != null && alertDataList.length > 0) {
2206           for(var i = 0;i < alertDataList.length;i++) {
2207               //console.log(alertDataList[i].key + " == " +
  field);
2208               if(alertDataList[i].key == field) {
2209                   ad = alertDataList[i];
2210                   //console.log('AD Found ' +
  smartphoneService.toJSON(ad));
2211                   break;
2212               }
2213           }
2214       }
2215       return ad;
2216   }
2217
2218  /**
2219   * @description parse customer alert data from the response
2220   * @param r
2221   * @returns CustomerAlert object
2222   */
2223   function parseCustomerAlert(r) {
2224       var ca = new CustomerAlert();
2225       ca.id = r.customerAlert.id;
```



```

2226         ca.customerId = r.customerAlert.customerId;
2227         ca.alertTypeId = r.customerAlert.alertTypeId;
2228         ca.active = r.customerAlert.active;
2229         ca.alertNotificationMsgId =
r.customerAlert.alertNotificationMsgId;
2230         ca.notifMaxCnt = r.customerAlert.notifMaxCnt;
2231         ca.notifMaxRecur = r.customerAlert.notifMaxRecur;
2232         ca.created = r.customerAlert.created;
2233
2234         if(r.customerAlert.alertDataList != null &&
r.customerAlert.alertDataList.alertData != null
2235             &&
r.customerAlert.alertDataList.alertData.length > 0) { //Check for
null values
2236             for(var i = 0;i <
r.customerAlert.alertDataList.alertData.length;i++) {
2237                 var tmpAd =
r.customerAlert.alertDataList.alertData[i];
2238                 var ad = new AlertData();
2239                 ad.id = tmpAd.id;
2240                 ad.customerAlertId = tmpAd.customerAlertId;
2241                 ad.key = tmpAd.key;
2242                 ad.value = tmpAd.value;
2243                 ca.alertDataList.push(ad);
2244             }
2245         }
2246
2247         var contactPointList =
r.customerAlert.contactPointList;
2248         if(r.customerAlert.contactPointList != null &&
r.customerAlert.contactPointList.contactPoint != null
2249             &&
r.customerAlert.contactPointList.contactPoint.length > 0) {
2250             for(var i = 0;i <
r.customerAlert.contactPointList.contactPoint.length;i++) {
2251                 var cp = new ContactPoint();

```

Money Mobiliser Core API

```
2252         cp.id =
r.customerAlert.contactPointList.contactPoint[i].id;
2253         cp.customerAlertId =
r.customerAlert.contactPointList.contactPoint[i].customerAlertId;
2254
2255     if(r.customerAlert.contactPointList.contactPoint[i].otherIdentifica
tion) {
2256         var otherIdentification =
r.customerAlert.contactPointList.contactPoint[i].otherIdentificatio
n;
2257         var oi = new OtherIdentification();
2258         oi.id = otherIdentification.id;
2259         oi.customerId =
otherIdentification.customerId;
2260         oi.type = otherIdentification.type;
2261         oi.identification =
otherIdentification.identification;
2262         oi.nickname = otherIdentification.nickname;
2263
2264         cp.otherIdentification = oi;
2265         delete cp.identification;
2266     } else {
2267         var identification =
r.customerAlert.contactPointList.contactPoint[i].identification;
2268         var oi = new Identification();
2269         oi.id = identification.id;
2270         oi.customerId = identification.customerId;
2271         oi.type = identification.type;
2272         oi.identification =
identification.identification;
2273
2274         cp.identification = oi;
2275         delete cp.otherIdentification;
2276     }
2277
```

```
2278         ca.contactPointList.push(cp);
2279     }
2280 }
2281
2282 //Parsing Blackout should I?
2283 ca.blackoutList = r.customerAlert.blackoutList;
2284 return ca;
2285 }
2286
2287 /**
2288  * @description Callback handler for Updating Alert
2289  */
2290 function updateExistingAlertBack(r) {
2291     hideProgress();
2292
2293     var rCode = getStatusCode(r);
2294     if (rCode == '0') {
2295         $('#aas_title').html(Str_txt_editalert);
2296         $.mobile.changePage("addAlertSuccess.html",
2297             page_opt);
2298         $('#alertcontentSuccess').html(Str_txt_editalert);
2299         session.customerAlert = null; //Clear the alert data
2300         from the session to avoid misuse.
2301     } else {
2302         _alertBox(Str_msg_failupdatealert +
2303             getErrorMessage(r));
2304         $('#aas_title').html(Str_txt_titleeditalertfail);
2305     }
2306     appstate = trans.TRAN_IDLE;
2307 }
2308
2309 /**
2310  * @description callback handler of getting the alert
2311  notification message id back from server
```

Money Mobiliser Core API

```
2308     * @param r
2309     */
2310     function getAlertNotificationMsgIdBack(r) {
2311         hideProgress();
2312
2313         var rCode = getStatusCode(r);
2314         if (rCode == '0') {
2315             var alertNotificationMsgTypeId =
                parseAlertNotificationMessageTypeId(r);
2316             //console.log("Got AlertNotificationMessageId from
                server. it is " + alertNotificationMsgTypeId);
2317
2318             var alertTypeId = $
                ("#btn_ma_addnewcontinue").attr('alertTypeId');
2319             var existingAlertId = $
                ("#btn_ma_addnewcontinue").attr('alertId');
2320
2321             //Check whether we need to add a new alert or we need
                to update an existing one.
2322             if($('#addAlertDetails').attr("action") != "edit")
                {
2323                 submitNewAddAlert(alertTypeId,
                alertNotificationMsgTypeId);
2324             } else {
2325                 updateExistingAlert(alertTypeId,
                alertNotificationMsgTypeId, existingAlertId);
2326             }
2327         } else {
2328             _alertBox(Str_msg_failgetchequestatus +
                getErrorMessage(r));
2329         }
2330         appstate = trans.TRAN_IDLE;
2331     }
2332
2333     /**
```

```

2334     * @description parsing logic for
alertNotificationMessageTypeId
2335     * @param r response data to be parsed.
2336     * @returns parsed alert notification message type id
2337     */
2338     function parseAlertNotificationMessageTypeId(r) {
2339         var alertNotificationMessageId;
2340         if(r.alertNotificationMessage &&
r.alertNotificationMessage.length > 0)
2341             alertNotificationMessageId =
r.alertNotificationMessage[0].id;
2342         return alertNotificationMessageId;
2343     }
2344
2345     /**
2346     * @description callback handler for the active alert
notification message mapping
2347     * web service call
2348     * @param r response data
2349     */
2350     function getActiveAlertNotificationMessagesBack(r) {
2351         hideProgress();
2352
2353         var rCode = getStatusCode(r);
2354         if (rCode == '0') {
2355             var alertNotificationMessages =
parseAlertNotificationMessages(r);
2356             //         if(alertNotificationMessages &&
alertNotificationMessages.length)
2357             //             console.log(alertNotificationMessages.length + "
Alert notifications found.")
2358
2359             $.mobile.changePage("managealert.html", page_opt);
2360         } else {

```

Money Mobiliser Core API

```
2361         _alertBox(Str_msg_failgetalertnoti +
getErrorMessage(r));
2362         //TODO: Shall we push user back??
2363     }
2364     appstate = trans.TRAN_IDLE;
2365 }
2366
2367 /**
2368  * @description Parser for alert notification messages
response
2369  * @return the array of AlertNotificationMessage objects
2370  */
2371 function parseAlertNotificationMessages(r) {
2372     session.alertNotificationMessages = []; //Remove old
entries from the array.
2373     if(r.alertNotificationMessage != null) {
2374         for(var i = 0;i < r.alertNotificationMessage.length;i
++) {
2375             var anm = new AlertNotificationMessage();
2376             //Parsing logic goes here
2377             anm.id = r.alertNotificationMessage[i].id;
2378             anm.alertTypeId =
r.alertNotificationMessage[i].alertTypeId;
2379             anm.notificationMsgTypeId =
r.alertNotificationMessage[i].notificationMsgTypeId;
2380             session.alertNotificationMessages.push(anm);
2381         }
2382     }
2383     return session.alertNotificationMessages;
2384 }
2385
2386 var transactiontypefilter = {"Peer 2 Peer":"Send Money",
"Standard C2C pull": "Request Money", "Standard Cash-In":"Cash In"
2387     ,"Standard Cash-Out": "Cash Out","Standard Credit
Self":"Add Funds to SVA","Standard Debit Self":"Withdraw Funds from
SVA"
```

```

2388         , "Standard load internal voucher": "Load Internal
Voucher", "Send voucher to unknown": "Send Voucher",
2389         "CommissionSettlement": "Commission Settlement", "Transfer
Money": "Merchant to Merchant", "chequeBook": "Request Cheque Book"
2390         , "AccountInfo": "Account Info", "chequeStatus": "Cheque
Status", "chequeStop": "Stop Cheque Payment"};
2391
2392     /**
2393     * @description Callback handler for getting the transaction
types back
2394     * @param r
2395     */
2396     function getTransactionTypesBack(r) {
2397         hideProgress();
2398
2399         var rCode = getStatusCode(r);
2400         if (rCode == '0') {
2401             transactiontypes =
parseLookupEntities(r, transactiontypefilter);
2402             //console.log("Number of transactions found are " +
transactiontypes.length);
2403             if(transactiontypes && transactiontypes.length > 0)
{
2404                 loadTransactions(transactiontypes.sort(dynamicSort('name')));
2405             } else {
2406                 _alertBox(Str_msg_failnotrantype);
2407                 goBack();
2408             }
2409         } else {
2410             _alertBox(Str_msg_failgettrantype +
getErrorMessge(r));
2411             goBack();
2412         }
2413         appstate = trans.TRAN_IDLE;
2414     }

```

Money Mobiliser Core API

```
2415
2416  /**
2417   * @description parser for currencies response
2418   * @param r response data
2419   * @param filter an optional array which maps a new string for
name
2420   * @returns {Array} of parsed currecy objects with id and name
properties
2421   */
2422   function parseLookupEntities(r, filter) {
2423       var entities = [];
2424       if(r.lookupEntities != null) {
2425           for(var i = 0;i < r.lookupEntities.length;i++) {
2426               var entity = {};
2427               entity.id = r.lookupEntities[i].id;
2428               //Check if filter is provided
2429               if(filter) {
2430                   //Check if filter contains the key
2431                   entity.name = filter[r.lookupEntities[i].name]?
filter[r.lookupEntities[i].name]:r.lookupEntities[i].name;
2432               } else {
2433                   //No filter store the name as it is returned by
service
2434                   entity.name = r.lookupEntities[i].name;
2435               }
2436               entities.push(entity);
2437           }
2438       }
2439       return entities;
2440   }
2441
2442  /**
2443   * @description Call back handler for receiving other
identification
```



```
2444     */
2445     function getOtherIdentificationsBack(r) {
2446         hideProgress();
2447
2448         var rCode = getStatusCode(r);
2449         if (rCode == '0') {
2450             var oIdentifications =
parseOtherIdentifications(r);
2451             listOtherIdentifications(oIdentifications,
'#aad_otheridentifications');
2452             //If this is a transaction types alert than we need one
more call to load type of transactions
2453             var alertTypeId = $
('#btn_ma_addnewcontinue').attr("alertTypeId");
2454             if(alertTypeId == "9" && CACHE_TRANSACTIONS &&
transactiontypes.length > 0) {
2455                 loadTransactions(transactiontypes.sort(dynamicSort('name')));
2456             } else {
2457                 mc.getLookups(getTransactionTypesBack,
"usecase");
2458                 appstate = trans.TRAN_TRANSACTIONTYPES;
2459                 showProgress();
2460             }
2461         } else {
2462             _alertBox(Str_msg_failgetotheridentmb +
getErrorMessage(r));
2463         }
2464         appstate = trans.TRAN_IDLE;
2465     }
2466
2467     /**
2468     * @description Parser for other identification web service
2469     */
2470     function parseOtherIdentifications(r) {
```

Money Mobiliser Core API

```
2471     session.otherIdentifications = [];    //Remove old entries
from the array.
2472     if(r.otherIdentification != null) {
2473         for(var i = 0;i < r.otherIdentification.length;i++)
{
2474             var type = r.otherIdentification[i].type;
2475             //console.log("Type found: " + type + " checking in
Array " + ALLOWED_DEVICE_TYPES);
2476             if($.inArray(type, ALLOWED_DEVICE_TYPES) > -1)
{ //Collect only Email and Mobile
2477                 var oi = new OtherIdentification();
2478                 //Parsing logic goes here
2479                 oi.id = r.otherIdentification[i].id;
2480                 oi.nickname =
r.otherIdentification[i].nickname;
2481                 oi.identification =
r.otherIdentification[i].identification;
2482                 oi.type = r.otherIdentification[i].type;
2483                 oi.customerId =
r.otherIdentification[i].customerId;
2484                 session.otherIdentifications.push(oi);
2485             } else {
2486                 //console.log("Ignoring CP due to unmatched
types ");
2487             }
2488         }
2489     }
2490     return session.otherIdentifications;
2491 }
```

Open Bank API

The OpenBank API enables users to access third-party banking systems, and provides the ability to manage multiple bank accounts.

Account class

This class stores information about specific accounts

Syntax

```
new Account( accountIdentification, institutionCode, type, accountStatus, linkedAccount,
branchCode, countryCode, currencyCode, accountHolder, alias, balance,
outstandingAmount, minimumAmount, statementAmount, paymentInstrumentId,
accountFamily, code, categoryDescription )
```

Parameters

| Name | Type | Description |
|------------------------------|------|-------------|
| <i>accountIdentification</i> | | |
| <i>institutionCode</i> | | |
| <i>type</i> | | |
| <i>accountStatus</i> | | |
| <i>linkedAccount</i> | | |
| <i>branchCode</i> | | |
| <i>countryCode</i> | | |
| <i>currencyCode</i> | | |
| <i>accountHolder</i> | | |
| <i>alias</i> | | |
| <i>balance</i> | | |
| <i>outstandingAmount</i> | | |
| <i>minimumAmount</i> | | |
| <i>statementAmount</i> | | |

| | | |
|----------------------------|--|---|
| <i>paymentInstrumentId</i> | | |
| <i>accountFamily</i> | | Legal values CA, CASA, OTHERS, CREATECARD, DB |
| <i>code</i> | | |
| <i>categoryDescription</i> | | |

Source

SY_OB_Data_Objects.js line 107 on page 247.

Amount class

Syntax

`new Amount(value, currency)`

Parameters

| Name | Type | Description |
|-----------------|------|-------------|
| <i>value</i> | | |
| <i>currency</i> | | |

Source

SY_OB_Data_Objects.js line 165 on page 249.

Authentication class

This stores authentication object which is needed for further authentication

Syntax

`new Authentication()`

Source

SY_OB_Data_Objects.js line 53 on page 245.

Customer class

Contains basic information about the customer.

Syntax

```
new Customer( customerId, msisdn, blacklist, test, typeId, cancelId, modeId )
```

Parameters

| Name | Type | Description |
|-------------------|--------|--|
| <i>customerId</i> | | |
| <i>msisdn</i> | string | The msisdn of the customer |
| <i>blacklist</i> | | An id for blacklistreason to indicate customer account status. |
| <i>test</i> | | test state of customer account. |
| <i>typeId</i> | | The id of customerType |
| <i>cancelId</i> | | The id of cancellationReason of customer account |
| <i>modeId</i> | | The id of mode of txn receipt mode of customer account |

Source

SY_OB_Data_Objects.js line 70 on page 246.

Denomination class

This class stores information about denominations

Syntax

```
new Denomination( Id, value )
```

Parameters

| Name | Type | Description |
|--------------|------|-------------|
| <i>Id</i> | | |
| <i>value</i> | | |

Source

SY_OB_Data_Objects.js line 230 on page 251.

Favourites class

This class stores information about favourites

Syntax

```
new Favourites( favId, favType, alias, details )
```

Parameters

| Name | Type | Description |
|----------------|------|-------------|
| <i>favId</i> | | |
| <i>favType</i> | | |
| <i>alias</i> | | |
| <i>details</i> | | |

Source

SY_OB_Data_Objects.js line 201 on page 250.

LoginSession class

LoginSession This is the session class which stores all of the user's data throughout a login session.

Syntax

```
new LoginSession( customerId, password, sessionId, customerId, displayName, msisdn, transaction, contactNumber, invoiceNo, registeredbill, openbill, openBillArray, accounts, totalaccounts, customer, svaalert )
```

Parameters

| Name | Type | Description |
|-------------------|------|--|
| <i>customerId</i> | | The customer id |
| <i>password</i> | | The password of the customer |
| <i>sessionId</i> | | The session id as returned by the server |
| <i>customerId</i> | | The id of the customer |

| | | |
|-----------------------|--|--|
| <i>displayName</i> | | The display name of the customer |
| <i>msisdn</i> | | The phone number of the customer |
| <i>transaction</i> | | Is an array of transactions |
| <i>contactNumber</i> | | Is an array of contacts from the address book of the user's smartphone |
| <i>invoiceNo</i> | | The current invoice number the user is selecting to pay for |
| <i>registeredbill</i> | | A list of all registered bills for the user |
| <i>openbill</i> | | |
| <i>openBillArray</i> | | A list of all open and due bills for the user |
| <i>accounts</i> | | A list of accounts registered by the user |
| <i>totalaccounts</i> | | total number of accounts registered by the user |
| <i>customer</i> | | A list of accounts registered by the user |
| <i>svaalert</i> | | A list of accounts registered by the user |

Source

SY_OB_Data_Objects.js line 147 on page 248.

MobiliserClient class

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.

Syntax

```
new MobiliserClient()
```

Source

SY_OB_Mobiliser.js line 18 on page 252.

denominations function

Mobile Reload denominations

Syntax

`denominations(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source

SY_OB_Mobiliser.js line 397 on page 265.

favouriteInfo function

Open Bank API to get the favorite info

Syntax

`favouriteInfo(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | response handler for the web service call. |

Source

SY_OB_Mobiliser.js line 135 on page 256.

favouriteList function

Open Bank API to get the favourite list

Syntax

`favouriteList(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | response handler for the web service call. |

Source

SY_OB_Mobiliser.js line 117 on page 255.

favouriteTransfer function

Open Bank API to get the favorite Transfer

Syntax

`favouriteTransfer(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | response handler for the web service call. |

Source

SY_OB_Mobiliser.js line 152 on page 256.

fundTransfer function

this service call transfer funds from one internal account to another customer own account.

Syntax

`fundTransfer(responseBack, fromAccount, toAccount, amount, msg)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | callback handler |
| <i>fromAccount</i> | | { Account } object that contains source account details |
| <i>toAccount</i> | | { Account } object that contains target account details |

Open Bank API

| | | |
|---------------|--|--|
| <i>amount</i> | | {Amount} object that contains value and currency |
| <i>msg</i> | | {String} optional message |

Source

SY_OB_Mobiliser.js line 240 on page 260.

getAccountInfo function

Open Bank API to get the detailed account info from the bank

Syntax

`getAccountInfo(responseBack, account)`

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | response handler for the web service call. |
| <i>account</i> | | account object for which the details are sought |

Source

SY_OB_Mobiliser.js line 98 on page 255.

getAccountList function

Open Bank API to get the account list from the bank

Syntax

`getAccountList(responseBack)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | response handler for the web service call. |

Source

SY_OB_Mobiliser.js line 79 on page 254.

getBankDetailsList function

Get the list of banks

Syntax

```
getBankDetailsList( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|------------------|
| <i>responseBack</i> | | Response Handler |

Source

SY_OB_Mobiliser.js line 45 on page 253.

getChequeStatus function

Fetching Cheque Status from the server.

Syntax

```
getChequeStatus()
```

Source

SY_OB_Mobiliser.js line 275 on page 261.

getExchangeRate function

Get the exchange rate between two currencies from the server

Syntax

```
getExchangeRate( org, from, to )
```

Parameters

| Name | Type | Description |
|-------------|------|------------------------|
| <i>org</i> | | id of the organisation |
| <i>from</i> | | the from currency |
| <i>to</i> | | the to currency |

Source

SY_OB_Mobiliser.js line 220 on page 259.

getLookups function

Function to fetch list of supported look up items like currencies networkproviders etc

Syntax

```
getLookups( responseBack, entity )
```

Parameters

| Name | Type | Description |
|---------------------|------|-------------------------------|
| <i>responseBack</i> | | the callback handler |
| <i>entity</i> | | to be looked up on the server |

Source

SY_OB_Mobiliser.js line 379 on page 264.

logout function

Agent logout function

Syntax

```
logout( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to be called when a response is received. |

Source

SY_OB_Mobiliser.js line 360 on page 264.

mBankingTopUp function

Mbanking top up request

Syntax

```
mBankingTopUp( responseBack )
```

Parameters

| Name | Type | Description |
|---------------------|------|---|
| <i>responseBack</i> | | Indicate which function to be called when a response is received. |

Source

SY_OB_Mobiliser.js line 413 on page 266.

obLogin function

make the call to Open Bank API for Login

Syntax

`obLogin(responseBack, msisdn, mpin)`

Parameters

| Name | Type | Description |
|---------------------|------|--|
| <i>responseBack</i> | | Indicates which function to call on return |
| <i>msisdn</i> | | Customer's msisdn |
| <i>mpin</i> | | Customer's mpin |

Source

SY_OB_Mobiliser.js line 62 on page 253.

placeChequeBookRequest function

ChequeBook placement request webservice call

Syntax

`placeChequeBookRequest(responseBack, chequeLeafs, account)`

Parameters

| Name | Type | Description |
|---------------------|------|----------------------------------|
| <i>responseBack</i> | | callback handler for the service |

| | | |
|--------------------|--|--|
| <i>chequeLeafs</i> | | number of cheques requested in the book |
| <i>account</i> | | object containing account details for which the cheque is requested. |

Source

SY_OB_Mobiliser.js line 334 on page 263.

placeChequeStopPayment function

Placing Cheque Stop request to the server.

Syntax

`placeChequeStopPayment(responseBack, chequeNumber, chequeNumberStr, chequeNumberEnd, accountPIId)`

Parameters

| Name | Type | Description |
|------------------------|------|--|
| <i>responseBack</i> | | callback handler for response received |
| <i>chequeNumber</i> | | |
| <i>chequeNumberStr</i> | | Starting number of check range |
| <i>chequeNumberEnd</i> | | Ending number of check range |
| <i>accountPIId</i> | | account id for the specified cheque |

Source

SY_OB_Mobiliser.js line 306 on page 262.

transactionList function

Make call to fetch transaction list from the server.

Syntax

`transactionList(responseBack)`

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|---------------------|--|--|
| <i>responseBack</i> | | and Account for which transaction to be fetched. |
|---------------------|--|--|

Source

SY_OB_Mobiliser.js line 197 on page 258.

OrgUnit class

OrgUnits which contains the available org units for which customer can log into the app.

Syntax

`new OrgUnit(id, name, currency)`

Parameters

| Name | Type | Description |
|-----------------|------|-------------|
| <i>id</i> | | |
| <i>name</i> | | |
| <i>currency</i> | | |

Source

SY_OB_Data_Objects.js line 43 on page 245.

Setting class

Setting is a connection configurations class to connect to the back-end server.

Syntax

`new Setting(protocol, ipaddress, port, wsname)`

Parameters

| Name | Type | Description |
|------------------|------|---|
| <i>protocol</i> | | String representing the protocol type. Either <code>"http://"</code> or <code>"https://"</code> . |
| <i>ipaddress</i> | | The IP address of the money mobiliser server |

| | | |
|---------------|--|---|
| <i>port</i> | | The port number of the money mobiliser server |
| <i>wsname</i> | | The name of the WS_Core web service; |

Source

SY_OB_Data_Objects.js line 20 on page 244.

Timer class

Timer

Syntax

`new Timer()`

Source

SY_OB_Data_Objects.js line 219 on page 251.

Transaction class

Transaction class

Syntax

`new Transaction(id, debitCredit)`

Parameters

| Name | Type | Description |
|--------------------|------|-----------------|
| <i>id</i> | | |
| <i>debitCredit</i> | | debit or credit |

Source

SY_OB_Data_Objects.js line 180 on page 249.

Global

alertXML function

This function can be used for debugging purposes to display the contents of an XML message.

Depending on the browser capabilities, the data will be serialized.

Syntax

```
alertXML( xmlDoc )
```

Parameters

| Name | Type | Description |
|---------------|------|---|
| <i>xmlDoc</i> | | This is the XML data that will be shown in the alert box. |

Source

SY_OB_Transactions.js line 704 on page 292.

denominationsBack function

Callback handler for Mbanking denominations

Syntax

```
denominationsBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|----------------------------------|
| <i>r</i> | | responseJSON containing products |

Source

SY_OB_Transactions.js line 778 on page 294.

favouriteInfoBack function

callback handler for favouriteInfo openbank api call

Syntax

```
favouriteInfoBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | responseXML |

Source

SY_OB_Transactions.js line 220 on page 274.

favouriteListBack function

callback handler for favouriteList openbank api call

Syntax

`favouriteListBack(r)`

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | responseXML |

Source

SY_OB_Transactions.js line 198 on page 273.

favouriteTransferBack function

callback handler for favouriteTransfer openbank api call

Syntax

`favouriteTransferBack(r)`

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | responseXML |

Source

SY_OB_Transactions.js line 264 on page 276.

fundTransferBack function

callback handler for transferFund openbank api call

Syntax

```
fundTransferBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | responseXML |

Source

SY_OB_Transactions.js line 125 on page 271.

getAccountInfoBack function

callback handler for getTransactionList openbank api call

Syntax

```
getAccountInfoBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | responseXML |

Source

SY_OB_Transactions.js line 108 on page 270.

getAccountListBack function

callback handler for getAccountList openbank api call

Syntax

```
getAccountListBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | responseXML |

Source

SY_OB_Transactions.js line 80 on page 269.

getBankDetailsListBack function

Callback handler for Bank Details list service call

Syntax

```
getBankDetailsListBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|--------------|
| <i>r</i> | | responseJSON |

Source

SY_OB_Transactions.js line 18 on page 267.

getChequeStatusResponse function

response handler for cheque status web service calls

Syntax

```
getChequeStatusResponse( r )
```

Parameters

| Name | Type | Description |
|----------|------|------------------|
| <i>r</i> | | responseXML/JSON |

Source

SY_OB_Transactions.js line 488 on page 284.

getChequeStopResponse function

response handler for cheque stop payment request

Syntax

```
getChequeStopResponse( r )
```

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

*Source**SY_OB_Transactions.js line 514 on page 284.***getCurrenciesBack function**

ResponseHandler for getCurrencies web service call

Syntax`getCurrenciesBack(r)`*Parameters*

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

*Source**SY_OB_Transactions.js line 157 on page 272.***getExchangeRateBack function**

callback handler for getting the exchange rate back

Syntax`getExchangeRateBack(r)`*Parameters*

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

*Source**SY_OB_Transactions.js line 180 on page 273.***getTransactionListBack function**

callback handler for getTransactionList openbank api call

Syntax`getTransactionListBack(r)`*Parameters*

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|----------|--|-------------|
| <i>r</i> | | responseXML |
|----------|--|-------------|

Source

SY_OB_Transactions.js line 141 on page 271.

MBankingTopUpBack function

Mbanking TopUp

Syntax

MBankingTopUpBack()

Source

SY_OB_Transactions.js line 762 on page 294.

parseAccountInfo function

Parser for Favourite List

Syntax

parseAccountInfo(*r*) { Array }

Parameters

| Name | Type | Description |
|----------|------|-------------------------------|
| <i>r</i> | | responseJSON from the service |

Returns

of Favourites objects

Type:

Array

Source

SY_OB_Transactions.js line 322 on page 278.

parseAccountList function

Parser for Bank Account List

Syntax

parseAccountList(*r*) { Array }

Parameters

| Name | Type | Description |
|----------|------|-------------------------------|
| <i>r</i> | | responseJSON from the service |

Returns

of Account objects

Type:

Array

Source

SY_OB_Transactions.js line 281 on page 276.

parseBankDetails function

Parser for results of getBankDetailsList

Syntax

```
parseBankDetails(r)
```

Parameters

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

Returns

array of orgunits

Source

SY_OB_Transactions.js line 35 on page 268.

parseChequeStatus function

Parser for cheque status call response

Syntax

```
parseChequeStatus(r) { String }
```

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|----------|--|--|
| <i>r</i> | | |
|----------|--|--|

Returns

Type:

String

Source

SY_OB_Transactions.js line 506 on page 284.

parseCurrencies function

parser for currencies response

Syntax

`parseCurrencies(r) { Array }`

Parameters

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

Returns

of parsed currency objects with id and name properties

Type:

Array

Source

SY_OB_Transactions.js line 453 on page 282.

parseDenomination function

parser for denomination response

Syntax

`parseDenomination(r) { Array }`

Parameters

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

Returns

of parsed denomination objects with id and name properties

Type:

Array

Source

SY_OB_Transactions.js line 800 on page 295.

parseExchangeRate function

parses the exchange rate fields from the server.

Syntax

```
parseExchangeRate(r) { Object }
```

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

Returns

containing exchange rate fields

Type:

Object

Source

SY_OB_Transactions.js line 471 on page 283.

parseFavouriteList function

Parser for Favourite List

Syntax

```
parseFavouriteList(r) { Array }
```

Parameters

| Name | Type | Description |
|----------|------|-------------------------------|
| <i>r</i> | | responseJSON from the service |

Returns

of Favourites objects

Type:

Array

Source

SY_OB_Transactions.js line 346 on page 279.

parseLookupEntities function

parser for currencies response

Syntax

```
parseLookupEntities(r, filter) { Array }
```

Parameters

| Name | Type | Description |
|---------------|------|--|
| <i>r</i> | | response data |
| <i>filter</i> | | an optional array which maps a new string for name |

Returns

of parsed currency objects with id and name properties

Type:

Array

Source

SY_OB_Transactions.js line 739 on page 293.

parseTransactionList function

Parser for Transaction List

Syntax

```
parseTransactionList(r) { Array }
```

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|----------|--|-------------------------------|
| <i>r</i> | | responseJSON from the service |
|----------|--|-------------------------------|

Returns
of Transaction objects

Type:

Array

Source

SY_OB_Transactions.js line 418 on page 281.

placeChequeBookRequestBack function

Callback handler for Chequebook placements

Syntax

placeChequeBookRequestBack(*r*)

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

Source

SY_OB_Transactions.js line 530 on page 285.

Source code

SY_OB_Data_Objects.js

A set of data objects utilized by the Mobiliser Smartphone application.

```

1      /**
2      * @overview
3      * A set of data objects utilized by the Mobiliser Smartphone
  application.
4      *
5      * @name SY_OB_Data_Objects.js
6      * @author SAP AG
7      * @version 1.0

```

Open Bank API

```
8      *
9      */
10
11
12     /**
13     @class Setting is a connection configurations class to
connect to the back-end server.
14     @constructor
15     @param protocol String representing the protocol type.
Either "http://" or "https://".
16     @param ipaddress The IP address of the money mobiliser
server
17     @param port The port number of the money mobiliser server
18     @param wsname The name of the WS_Core web service;
19     */
20     function Setting() {
21         this.protocol = 'http://';
22         this.ipaddress =
'216.207.70.199';//'192.168.2.4';//'10.99.29.110';//'192.168.2.4';/
/'10.91.28.89';//'10.99.28.89';//'202.45.6.122';//'10.0.2.2';//'216
.207.70.198';
23         this.port = '8080';//'80';
24         this.wsname = 'mobiliser/rest/spmmbanking';
25         this.spmws = 'mobiliser/rest/smartphone';
26         this.origin = "MAPP";
27         this.appinfo = {};
28         this.appinfo.device = '';
29         this.appinfo.deviceId = '';
30         this.appinfo.otherDeviceId = '';
31         this.appinfo.application = "MAPP"; // TODO: change app
name and add version number (svn revision ? build number ?)
32         // Currently, appid used for
AuditData expects "MAPP", otherwise user cannot login.
33         this.appinfo.applicationVersion = "5.1.3";
34     }
```

```
35
36     /**
37     * @class OrgUnits which contains the available org units for
38     * which customer can
39     *       log into the app.
40     * @param id
41     * @param name
42     * @param currency
43     */
44     function OrgUnit() {
45         this.id = '';
46         this.name = '';
47         this.currency = '';
48     }
49     /**
50     * @class This stores authentication object which is needed
51     * for further
52     * authentication
53     */
54     function Authentication() {
55         this.token = '';
56     }
57     /**
58     * @class Contains basic information about the customer.
59     */
60     @constructor
61
62     @param customerId
63     @param {string} msisdn The msisdn of the customer
64     @param blacklist An id for blacklistreason to indicate
65     customer account status.
```

Open Bank API

```
65     @param test test state of customer account.
66     @param typeId The id of customerType
67     @param cancelId The id of cancellationReason of customer
account
68     @param modeId The id of mode of txn receipt mode of customer
account
69     */
70     function Customer() {
71         this.customerId = '';
72         this.orgUnitId = '';
73         this.blacklist = '';
74         this.active = '';
75         this.test = '';
76         this.displayName = '';
77         this.language = '';
78         this.timezone = '';
79         this.typeId = '';
80         this.cancelId = '';
81         this.modeId = '';
82     }
83
84
85
86     /**
87     * @class This class stores information about specific
accounts
88     * @param accountIdentification
89     * @param institutionCode
90     * @param type
91     * @param accountStatus
92     * @param linkedAccount
93     * @param branchCode
94     * @param countryCode
```

```
95      * @param currencyCode
96      * @param accountHolder
97      * @param alias
98      * @param balance
99      * @param outstandingAmount
100     * @param minimumAmount
101     * @param statementAmount
102     * @param paymentInstrumentId
103     * @param accountFamily Legal values CA, CASA, OTHERS,
104     * @param code
105     * @param categoryDescription
106     */
107     function Account() {
108         this.accountIdentification = '';
109         this.institutionCode = '';
110         this.type = '';
111         this.accountStatus = '';
112         this.paymentInstrumentId = '';
113         this.linkedAccount = '';
114         this.branchCode = '';
115         this.countryCode = '';
116         this.currencyCode = '';
117         this.accountHolder = '';
118         this.alias = '';
119         this.balance = '';
120         this.availableFunds = '';
121         this.outstandingAmount = '';
122         this.minimumAmount = '';
123         this.statementAmount = '';
124     }
125
```

Open Bank API

```
126     /**
127         @class LoginSession This is the session class which stores
128         all of the user's data throughtout a login session.
129
129         @constructor
130         @param customerId The customer id
131         @param password The password of the customer
132         @param sessionId The session id as returned by the server
133         @param customerId The id of the customer
134         @param displayName The display name of the customer
135         @param msisdn The phone number of the customer
136         @param transaction Is an array of tranasctions
137         @param contactNumber Is an array of contacts from the
138         address book of the user's smartphone
139         @param invoiceNo The current invoice number the user is
140         selecting to pay for
141         @param registeredbill A list of all registered bills for the
142         user
143         @param openbill
144         @param openBillArray A list of all open and due bills for
145         the user
146         @param accounts A list of accounts registered by the user
147         @param totalaccounts total number of accounts registered by
148         the user
149         @param customer A list of accounts registered by the user
150         @param svaalert A list of accounts registered by the user
151
152     */
153     function LoginSession() {
154         /*Open Bank API data*/
155         this.customerId = '';
156         this.userId = '';
157         this.userMsisdn = '';
158         this.bankId = '';
159         this.customerSupportNo = '';
```



```
154     this.idNumber = '';
155     this.idType = '';
156     this.authentication = new Authentication();
157     this.accountList = [];
158     this.customer = new Customer();
159     this.totalfavourites = '';
160     this.favourites = [];
161     this.txns = [];
162     this.denominations = [];
163 }
164
165 /**
166  * @class Amount
167  * @param value
168  * @param currency
169  */
170 function Amount() {
171     this.value = '';
172     this.currency = '';
173 }
174
175 /**
176  * @class Transaction class
177  * @param id
178  * @param debitCredit debit or credit
179  */
180 function Transaction() {
181     this.id = '';
182     this.debitCredit = '';
183     this.amount = new Amount();
184     this.date = '';
185     this.postingDate = '';
```

Open Bank API

```
186         this.account = new Account();
187         this.balance = new Amount();
188         this.chequeNumber = '';
189         this.text1 = '';
190         this.text2 = '';
191         this.text3 = '';
192         this.text4 = '';
193     }
194     /**
195      * @class This class stores information about favourites
196      * @param favId
197      * @param favType
198      * @param alias
199      * @param details
200      */
201     function Favourites(){
202         this.favId = '';
203         this.favType = '';
204         this.alias = '';
205         this.accountHolder = '';
206         this.accountIdentification = '';
207         this.paymentType = '';
208         this.paymentInstrumentId = '';
209         this.to_account_type = '';
210         this.billerName = '';
211         this.billerCode = '';
212     }
213
214     /**
215      * @class Timer
216
217      * @constructor
```

```

218     */
219     function Timer() {
220         this.handle = '';
221         this.on_flag = 0;
222     }
223
224
225     /**
226     * @class This class stores information about denominations
227     * @param Id
228     * @param value
229     */
230     function Denomination(){
231         this.id = '';
232         this.amount = new Amount();
233         this.description = '';
234         this.code = '';
235         this.subcode = '';
236     }

```

SY_OB_Mobiliser.js

A thin JavaScript web service client that accesses the Mobiliser platform.

```

1     /**
2     * @fileOverview A thin JavaScript web service client that
3     * accesses the Mobiliser platform.
4     * It provides an abstraction layer to communicate with the
5     * system and returns
6     * XML documents as a result.
7     *
8     * @name SY_OB_Mobiliser.js
9     * @author SAP AG
10    * @version 1.0
11    *

```

Open Bank API

```
10      */
11
12      /**
13       * @class A thin JavaScript web service client that accesses
14       * the Mobiliser platform. It provides an abstraction layer to
15       * communicate with the system and returns XML documents as a result.
16
17       * @constructor
18
19       */
20
21      function MobiliserClient() {
22          this.url = setting.protocol + setting.ipaddress;
23
24          if(setting.port)
25              this.url += ":" + setting.port;
26
27          if(typeof setting.spmws !== 'undefined')
28              this.spmurl = this.url + "/" + setting.spmws;
29
30          if(typeof setting.wsname !== 'undefined')
31              this.url += "/" + setting.wsname;
32
33          smartphoneService.url = this.url;
34          this.network_access = true;
35      }
36
37      /**
38       * *****/
39       * ** OPEN BANK API CALLS **/
40       * *****/
41
42      /* web service calls */
43
44      /**
```

```

41      * @description Get the list of banks
42      * @param responseBack
43      *           Response Handler
44      */
45      MobiliserClient.prototype.getBankDetailsList =
function(responseBack) {
46          var pl = new Object();
47          pl.origin = setting.origin;
48          pl.traceNo = UUIDv4();
49          pl.AuditData = setting.appinfo;
50          smartphoneService.post(pl, "getBankList",
responseBack);
51      };
52
53      /**
54      * @description make the call to Open Bank API for Login
55      * @param responseBack
56      *           Indicates which function to call on
return
57      * @param msisdn
58      *           Customer's msisdn
59      * @param mpin
60      *           Customer's mpin
61      */
62      MobiliserClient.prototype.obLogin = function(responseBack,
msisdn, mpin, bankId) {
63          var pl = {};
64          pl.origin = setting.origin;
65          pl.traceNo = UUIDv4();
66          pl.AuditData = setting.appinfo;
67          pl.bankId = bankId;
68          pl.userMsisdn = msisdn;
69          pl.mPin = mpin;

```

Open Bank API

```
70     smartphoneService.post(pl, "mBankingLogin",
responseBack);
71     };
72
73
74     /**
75     * @description Open Bank API to get the account list from the
bank
76     * @param responseBack
77     *         response handler for the web service call.
78     */
79     MobiliserClient.prototype.getAccountList =
function(responseBack, accountId) {
80         var pa = {};
81         jQuery.extend(pa, pl);
82         pa.origin = setting.origin;
83         pa.traceNo = UUIDv4();
84         pa.AuditData = setting.appinfo;
85
86         if(accountTypeId)
87             pa.accountListType = accountTypeId;
88         smartphoneService.post(pa, "getAccountList",
responseBack);
89     }
90
91     /**
92     * @description Open Bank API to get the detailed account info
from the bank
93     * @param responseBack
94     *         response handler for the web service call.
95     * @param account
96     *         account object for which the details are
sought
97     */
```

```

98     MobiliserClient.prototype.getAccountInfo =
function(responseBack, account) {
99         var pm = {};
100        jQuery.extend(pm, pl);
101        pm.origin = setting.origin;
102        pm.traceNo = UUIDv4();
103        pm.AuditData = setting.appinfo;
104
105        pm.primaryAccount = {};
106        pm.primaryAccount.accountIdentification =
account.accountIdentification;
107        pm.primaryAccount.type = account.type;
108        pm.primaryAccount.paymentInstrumentId =
account.paymentInstrumentId;
109        smartphoneService.post(pm, "getAccountBalance",
responseBack);
110    };
111
112    /**
113     * @description Open Bank API to get the favourite list
114     * @param responseBack
115     *         response handler for the web service call.
116     */
117    MobiliserClient.prototype.favouriteList =
function(responseBack, favTypeId, favTypeIdForBankCall) {
118        var pa = {};
119        jQuery.extend(pa, pl);
120
121        pa.origin = setting.origin;
122        pa.traceNo = UUIDv4();
123        pa.AuditData = setting.appinfo;
124
125        pa.favouriteType = favTypeId;
126        pa.favouriteTypeForBankCall = favTypeIdForBankCall;

```

Open Bank API

```
127     smartphoneService.post(pa, "favouriteList",
responseBack);
128     }
129
130     /**
131     * @description Open Bank API to get the favorite info
132     * @param responseBack
133     *         response handler for the web service call.
134     */
135     MobiliserClient.prototype.favouriteInfo =
function(responseBack,id,type) {
136         var pa = {};
137         jQuery.extend(pa,pl);
138
139         pa.origin = setting.origin;
140         pa.traceNo = UUIDv4();
141         pa.AuditData = setting.appinfo;
142         pa.favouriteIdentification = id;
143         pa.favouriteType = type;
144         smartphoneService.post(pa, "favouriteInfo",
responseBack);
145     }
146
147     /**
148     * @description Open Bank API to get the favorite Transfer
149     * @param responseBack
150     *         response handler for the web service call.
151     */
152     MobiliserClient.prototype.favouriteTransfer =
function(responseBack,id_paymenttype,from_accnt,to_transfer,amount,
pIID) {
153         var pa ={};
154         jQuery.extend(pa, pl);
155
```



```
156
157     pa.origin = setting.origin;
158     pa.traceNo = UUIDv4();
159     pa.AuditData = setting.appinfo;
160
161     pa.operationType = id_paymenttype.paymentType;
162
163     var pfa = {};
164     pfa.accountIdentification =
from_accnt.accountIdentification;
165     pfa.institutionCode = from_accnt.institutionCode;
166     pfa.type = from_accnt.type
167     pfa.paymentInstrumentId =
from_accnt.paymentInstrumentId;
168     pfa.code = from_accnt.code;
169
170     pa.fromAccount = pfa;
171     pa.paymentType = id_paymenttype.paymentType;
172     pa.favouriteIdentification = id_paymenttype.favId;
173
174     if(to_transfer.accountIdentification){
175         var pta = {};
176         pta.accountIdentification =
to_transfer.accountIdentification;
177         pta.type = to_transfer.to_account_type;
178         pta.paymentInstrumentId =
to_transfer.paymentInstrumentId;
179         pa.toAccount = pta;
180     }
181     else if(to_transfer.billerName){
182         pa.billerName = to_transfer.billerName;
183         pa.billerPaymentInstrumentId =
to_transfer.paymentInstrumentId;
184         pa.billerCode = to_transfer.billerCode;
```

Open Bank API

```
185     }
186
187     pa.amount = amount;
188
189     console.log(smartphoneService.toJSON(pa));
190     smartphoneService.post(pa, "favouriteTransfer",
responseBack);
191 };
192
193 /**
194  * @description Make call to fetch transaction list from the
server.
195  * @param responseBack and Account for which transaction to be
fetched.
196  */
197     MobiliserClient.prototype.transactionList =
function(responseBack, account) {
198         var pa = jQuery.extend({}, pl);
199         pa.origin = setting.origin;
200         pa.traceNo = UUIDv4();
201         pa.AuditData = setting.appinfo;
202
203         pa.primaryAccount = {};
204         pa.primaryAccount.accountIdentification =
account.accountIdentification;
205         pa.primaryAccount.type = account.type;
206         pa.primaryAccount.paymentInstrumentId =
account.paymentInstrumentId;
207
208         smartphoneService.post(pa, "getAccountTxnHistory",
responseBack);
209     };
210
211     /**
```

```

212      * @description Get the exchange rate between two currencies
from the server
213      * @param org
214      *          id of the organisation
215      * @param from
216      *          the from currency
217      * @param to
218      *          the to currency
219      */
220      MobiliserClient.prototype.getExchangeRate =
function(responseBack, from, to) {
221          var pa = {};
222          pa.origin = "android";
223          pa.traceNo = UUIDv4();
224          pa.AuditData = setting.appinfo;
225          pa.bankId = pl.bankId;
226          pa.fromCurrency = from;
227          pa.toCurrency = to;
228          smartphoneService.post(pa, "getMBExchangeRate",
responseBack);
229      };
230
231      /**
232      * @description this service call transfer funds from one
internal account
233      *          to another customer own account.
234      * @param responseBack callback handler
235      * @param fromAccount {Account} object that contains source
account details
236      * @param toAccount {Account} object that contains target
account details
237      * @param amount {Amount} object that contains value and
currency
238      * @param msg {String} optional message
239      */

```

Open Bank API

```
240 MobiliserClient.prototype.fundTransfer =
function(responseBack, fromAccount, toAccount, amount, msg) {
241     var plm = {};
242     jQuery.extend(plm, pl);
243
244     plm.origin = setting.origin;
245     plm.traceNo = UUIDv4();
246     plm.AuditData = setting.appinfo;
247
248     //plm.operationType = "FUND_TRANSFER_INTRA_BANK";
249
250     plm.fromAccount = {};
251     plm.fromAccount.accountIdentification =
fromAccount.accountIdentification;
252     plm.fromAccount.institutionCode =
fromAccount.institutionCode;
253     plm.fromAccount.type = fromAccount.type;
254     plm.fromAccount.paymentInstrumentId =
fromAccount.paymentInstrumentId;
255     plm.fromAccount.code = fromAccount.code;
256
257     plm.toAccount = {};
258     plm.toAccount.accountIdentification =
toAccount.accountIdentification;
259     plm.toAccount.institutionCode =
toAccount.institutionCode;
260     plm.toAccount.type = toAccount.type;
261     plm.toAccount.paymentInstrumentId =
toAccount.paymentInstrumentId;
262     plm.toAccount.code = toAccount.code;
263
264     plm.amount = amount;
265     plm.text = msg;
266
```

```
267         console.log("Fund Transfer request is " +
smartphoneService.toJSON(plm));
268
269         smartphoneService.post(plm, "transferFund",
responseBack);
270     };
271
272     /**
273     * @description Fetching Cheque Status from the server.
274     */
275     MobiliserClient.prototype.getChequeStatus =
function(responseBack, chequeNumber, account) {
276         var pa = {};
277         jQuery.extend(pa, pl);
278
279         pa.primaryAccount = {};
280         pa.primaryAccount.accountIdentification =
account.accountIdentification;
281         pa.primaryAccount.type = account.type;
282         pa.primaryAccount.paymentInstrumentId =
account.paymentInstrumentId;
283
284         pa.origin = setting.origin;
285         pa.traceNo = UUIDv4();
286         pa.AuditData = setting.appinfo;
287
288         // pl.bankId = session.customer.orgUnitId;
289         pa.chequeNumber = chequeNumber;
290
291         smartphoneService.post(pa, "chequeStatus",
responseBack);
292     };
293
294     /**
```

Open Bank API

```
295      * @description Placing Cheque Stop request to the server.
296      * @param responseBack
297      *           callback handler for response received
298      * @param chequeNumber
299      * @param chequeNumberStr
300      *           Starting number of check range
301      * @param chequeNumberEnd
302      *           Ending number of check range
303      * @param accountPIId
304      *           account id for the specified cheque
305      */
306      MobiliserClient.prototype.placeChequeStopPayment =
function(responseBack, chequeNumber, account) {
307          var pa = {};
308          jQuery.extend(pa, pl);
309
310          pa.primaryAccount = {};
311          pa.primaryAccount.accountIdentification =
account.accountIdentification;
312          pa.primaryAccount.type = account.type;
313          pa.primaryAccount.paymentInstrumentId =
account.paymentInstrumentId;
314
315          pa.origin = setting.origin;
316          pa.traceNo = UUIDv4();
317          pa.AuditData = setting.appinfo;
318
319          pa.chequeNumber = chequeNumber;
320
321          smartphoneService.post(pa, "stopCheque", responseBack);
322      };
323
324      /**
```

```
325 * @description ChequeBook placement request webservice
call
326 * @param responseBack
327 * callback handler for the service
328 * @param chequeLeafs
329 * number of cheques requested in the book
330 * @param account
331 * object containing account details for which the
cheque is
332 * requested.
333 */
334 MobiliserClient.prototype.placeChequeBookRequest =
function(responseBack, chequeLeafs, account, pIID) {
335     var cr = {};
336     jQuery.extend(cr, pl);
337
338     var pa = {};
339     pa.accountIdentification =
account.accountIdentification;
340     pa.type = account.type;
341     pa.paymentInstrumentId =
account.linkedAccount.paymentInstrumentId;
342
343     cr.origin = setting.origin;
344     cr.traceNo = UUIDv4();
345     cr.AuditData = setting.appinfo;
346
347     // pl.bankId =
session.customer.orgUnitId;pl.primaryAccount = pa;
348     cr.primaryAccount = pa;
349     cr.numberOfChequeBooks = chequeLeafs;
350
351     smartphoneService.post(cr, "requestChequeBook",
responseBack);
352     };
```

```
353
354
355
356     /**
357         @description Agent logout function
358         @param responseBack Indicates which function to be called
when a response is received.
359     */
360     MobiliserClient.prototype.logout = function(responseBack) {
361         var pl = {};
362         pl.origin = setting.origin;
363         pl.traceNo = UUIDv4();
364         pl.AuditData = setting.appinfo;
365         pl.sessionId = session.sessionId;
366         if(!mbankingService.url) {
367             jQuery.extend(mbankingService, smartphoneService);
368             mbankingService.url = mc.spmurl;
369         }
370         mbankingService.post(pl, "logout", responseBack);
371     };
372
373     /**
374         * @description Function to fetch list of supported look up
items like currencies
375         * networkproviders etc
376         * @param responseBack the callback handler
377         * @param entity to be looked up on the server
378     */
379     MobiliserClient.prototype.getLookups =
function(responseBack, entity) {
380         var pl = new Object();
381         pl.origin = setting.origin;
382         pl.traceNo = UUIDv4();
```



```

383     pl.AuditData = setting.appinfo;
384     pl.entityName = entity;
385     if(!mbankingService.url) {
386         jQuery.extend(mbankingService, smartphoneService);
387         mbankingService.url = mc.spmurl;
388     }
389     mbankingService.post(pl, "getLookups", responseBack);
390 };
391
392 /**
393  * @description Mobile Reload denominations
394  * @param responseBack Indicate which function to be called
395  * when a response is received.
396  */
397     MobiliserClient.prototype.denominations =
398     function(responseBack){
399         var pa = {};
400         jQuery.extend(pa, pl);
401         pa.origin = setting.origin;
402         pa.traceNo = UUIDv4();
403         pa.AuditData = setting.appinfo;
404         pa.operatorId = session.customerId;
405         console.log(smartphoneService.toJSON(pa));
406         smartphoneService.post(pa, "getDenominationList",
407         responseBack);
408     };
409 /**
410  * @description Mbanking top up request
411  * @param responseBack Indicate which function to be called
412  * when a response is received.
413  */

```

Open Bank API

```
413     MobiliserClient.prototype.mBankingTopUp =  
function(responseBack, account, product){  
414         var pa = {};  
415         jQuery.extend(pa, pl);  
416         pa.origin = setting.origin;  
417         pa.traceNo = UUIDv4();  
418         pa.AuditData = setting.appinfo;  
419  
420         pa.fromAccount = {};  
421         pa.fromAccount.accountIdentification =  
account.accountIdentification;  
422         pa.fromAccount.institutionCode =  
account.institutionCode;  
423         pa.fromAccount.type = account.type;  
424         pa.fromAccount.paymentInstrumentId =  
account.paymentInstrumentId;  
425  
426         pa.mobilePhoneNumber = session.userMsisdn;  
427  
428         pa.operator = session.customerId;  
429  
430         pa.product = {};  
431         pa.product.productId = product.id;  
432         pa.product.productCode = product.code;  
433         pa.product.productType = product.subcode;  
434  
435         pa.amount = product.amount;  
436  
437         smartphoneService.post(pa, "mobileReload",  
responseBack);  
438     };  
439
```

SY_OB_Transactions.js

This is the actual response handling and processing of the web service calls that are made by the MobiliserClient class.

```

1      /**
2      * @overview
3      * This is the actual response handling and processing of the
web service calls
4      * that are made by the MobiliserClient class. Any
manipulation of the response,
5      * extraction of needed data and the making of Data Objects
(DO) out of the XML
6      * document would be inserted here.
7      *
8      * @name SY_OB_Transactions.js
9      * @author SAP AG
10     * @version 1.0
11     *
12     */
13
14     /**
15     * @description Callback handler for Bank Details list service
call
16     * @param r responseJSON
17     */
18     var getBankDetailsListBack = function(r) {
19         hideProgress();
20         var rCode = getStatusCode(r);
21         if (rCode == '0') {
22             orgunits = parseBankDetails(r);
23             updateOrgUnitList('#bankdroplist', orgunits);
24         } else {
25             _alertBox('Failed to get orgunits. ' +
getErrorMessage(r));
26         }

```

Open Bank API

```
27     appstate = trans.TRAN_IDLE;
28 };
29
30 /**
31  * @description Parser for results of getBankDetailsList
32  * @param r response data
33  * @returns array of orgunits
34  */
35     function parseBankDetails(r) {
36         orgunits = []; //Remove old entries from the array.
37         if(r.bankDetails != null) {
38             for(var i = 0;i < r.bankDetails.length;i++) {
39                 var ou = new OrgUnit();
40                 //Parsing logic goes here
41                 ou.id = r.bankDetails[i].bankId;
42                 ou.name = r.bankDetails[i].bankName;
43                 ou.currency =
r.bankDetails[i].currencyDisplaySymbol;
44                 orgunits.push(ou);
45             }
46         }
47         return orgunits;
48     }
49
50 /**
51     @function
52
53     @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
54     */
55     var obLoginBack = function(r) {
56         var rCode = getStatusCode(r);
57         if (appstate == trans.TRAN_LOGIN) {
```

```

58         if(rCode == '0') {
59             loginParseXML(r);
60             session.userMsisdn = $('#username').val();
61             pl.userMsisdn = $('#username').val();
62             loginUpdateUI();
63
64             appstate = trans.TRAN_ACCOUNTLIST;
65             mc.getAccountList(getAccountListBack, "SA");
66         } else {
67             hideProgress();
68             appstate = trans.TRAN_IDLE;
69             _alertBox("Login failed! " +
getErrorMessage(r));
70         }
71         $('#username').val('');
72         $('#password').val('');
73     }
74 };
75
76 /**
77  * @description callback handler for getAccountList openbank
api call
78  * @param r responseXML
79  */
80     function getAccountListBack(r) {
81         hideProgress();
82         var rCode = getStatusCode(r);
83         if (rCode == '0') {
84             if(appstate == trans.TRAN_ACCOUNTLIST) {
85                 session.accountList = parseAccountList(r);
86                 if(session.accountList.length < 1) {
87                     _alertBox("Failed to fetch any relevant account
from the Bank, cheque management cannot be used.");

```

Open Bank API

```
88         }
89     } else if (appstate == trans.TRAN_ACCOUNTLISTBYTYPE)
90     {
91         var accountList = parseAccountList(r);
92         if(accountList.length < 1) {
93             _alertBox("Failed to fetch any relevant account
94 from the Bank, cheque management cannot be used.");
95         } else {
96             listExistingAccounts(accountList);
97             $.mobile.changePage("accountlist.html",
98 page_opt);
99         }
100     }
101     } else {
102         _alertBox('Failed to get account list for customer. ' +
103 getErrorMessage(r));
104     }
105     appstate = trans.TRAN_IDLE;
106 }
107
108 /**
109  * @description callback handler for getTransactionList
110 openbank api call
111  * @param r responseXML
112  */
113 function getAccountInfoBack(r) {
114     hideProgress();
115     var rCode = getStatusCode(r);
116     if (rCode == '0') {
117         //Todo parse and display transactions
118         var account = parseAccountInfo(r);
119         listBankAccount(account);
120     } else {
```

```
116     _alertBox('Failed to get account list for customer. ' +
117         getErrorMessage(r));
118     }
119     appstate = trans.TRAN_IDLE;
120 }
121 /**
122  * @description callback handler for transferFund openbank
123  * @param r responseXML
124  */
125 function fundTransferBack(r) {
126     hideProgress();
127     var rCode = getStatusCode(r);
128     if (rCode == '0') {
129         updateTransactionDoneUI(r.fromAccountBalance,
130             r.toAccountBalance, r.transactionId);
131     } else {
132         _alertBox('Failed to complete the transaction. ' +
133             getErrorMessage(r));
134     }
135     appstate = trans.TRAN_IDLE;
136 }
137 /**
138  * @description callback handler for getTransactionList
139  * @param r responseXML
140  */
141 function getTransactionListBack(r) {
142     hideProgress();
143     var rCode = getStatusCode(r);
144     if (rCode == '0') {
```

Open Bank API

```
145         session.txns = parseTransactionList(r);
146         txnsUpdateUI(session.txns);
147     } else {
148         _alertBox('Failed to get account list for customer. ' +
149         getErrorMessage(r));
150     }
151     appstate = trans.TRAN_IDLE;
152 }
153 /**
154  * @description ResponseHandler for getCurrencies web service
155  * call
156  * @param r
157  */
158 function getCurrenciesBack(r) {
159     hideProgress();
160     var rCode = getStatusCode(r);
161     if (rCode == '0') {
162         currencies = parseCurrencies(r);
163         if(currencies && currencies.length > 0) {
164             listCurrencies('#fx_fromcurrency', currencies);
165             listCurrencies('#fx_tocurrency', currencies);
166         } else {
167             //TODO Show error message and go back??
168         }
169     } else {
170         _alertBox('Failed to get Currencies.\n' +
171         getErrorMessage(r));
172         //TODO: Shall we push user back??
173     }
174     appstate = trans.TRAN_IDLE;
175 }
```



```

175
176     /**
177     * @description callback handler for getting the exchange rate
back
178     * @param r response data
179     */
180     function getExchangeRateBack(r) {
181         hideProgress();
182         var rCode = getStatusCode(r);
183         if (rCode == '0') {
184             var exchangeRate = parseExchangeRate(r);
185             updateExchangeRateUI(exchangeRate);
186             $.mobile.changePage("fxresult.html", page_opt);
187         } else {
188             _alertBox('Failed to get Exchange rate.\n' +
getErrorMessage(r));
189             //TODO: Shall we push user back??
190         }
191         appstate = trans.TRAN_IDLE;
192     }
193
194     /**
195     * @description callback handler for favouriteList openbank
api call
196     * @param r responseXML
197     */
198     function favouriteListBack(r) {
199         hideProgress();
200         var rCode = getStatusCode(r);
201         if (rCode == '0') {
202             //code to be done
203             session.favourites = parseFavouriteList(r);
204             if(session.favourites.length < 1) {

```

Open Bank API

```
205         _alertBox("No favourite list fetched for this
user");
206     } else {
207         updateFavouiteList(session.favourites);
208         $.mobile.changePage("favouritelist.html",
page_opt);
209     }
210 } else {
211     _alertBox('Failed to fetch favourite list for
customer. ' + getErrorMessage(r));
212 }
213     appstate = trans.TRAN_FAVOURITELIST;
214 }
215 /**
216  * @description callback handler for favouriteInfo openbank
api call
217  * @param r responseXML
218  */
219
220     function favouriteInfoBack(r) {
221         hideProgress();
222         var rCode = getStatusCode(r);
223         if (rCode == '0') {
224             $("#fav_type").text(r.favourite.favouriteType);
225             $("#fav_alias").text(r.favourite.alias);
226             if (r.favourite.details.toAccount) {
227                 $
("#fav_accnt_holder").text(r.favourite.details.accountHolder);
228                 $
("#fav_accnt_id").text(r.favourite.details.toAccount.accountIdentif
ication);
229                 $(".biller_details").hide();
230             } else if (r.favourite.details.billerInfo) {
231                 $
("#fav_biller_name").text(r.favourite.details.billerInfo.billerName
);

```

```

232         $
("#fav_biller_code").text(r.favourite.details.billerInfo.billerCode
);
233         $(".account_details").hide();
234     }
235     $("#btn_fav_tranfer_money").unbind('click');
236     $("#btn_fav_tranfer_money").bind('click',function()
{
237         //need to change
238         var obj= new Object();
239         obj.alias=r.favourite.alias;
240         obj.favId =
r.favourite.favouriteIdentification;
241         obj.paymentType =
r.favourite.details.paymentType;
242         if(r.favourite.details.toAccount){
243             obj.accountIdentification =
r.favourite.details.toAccount.accountIdentification;
244             obj.paymentInstrumentId =
r.favourite.details.toAccount.paymentInstrumentId;
245             obj.to_account_type =
r.favourite.details.toAccount.type;
246         } else if (r.favourite.details.billerInfo) {
247             obj.billerName =
r.favourite.details.billerInfo.billerName;
248             obj.paymentInstrumentId =
r.favourite.details.billerInfo.billerPaymentInstrumentId;
249             obj.billerCode =
r.favourite.details.billerInfo.billerCode;
250         }
251         favStartTransferMoney(obj,"false");
252     });
253     $.mobile.changePage("favouriteinfo.html",
page_opt);
254     } else {
255         _alertBox('Failed to fetch favourite Information. ' +
getErrorMessage(r));

```

Open Bank API

```
256     }
257     appstate = trans.TRAN_FAVOURITEINFO;
258 }
259 /**
260  * @description callback handler for favouriteTransfer
261  * @param r responseXML
262  */
263
264 function favouriteTransferBack(r) {
265     hideProgress();
266     var rCode = getStatusCode(r);
267     if (rCode == '0') {
268         $.mobile.changePage("favouritetransferdone.html",
269         page_opt);
270     } else {
271         _alertBox('Error:\n' + getErrorMessage(r));
272     }
273     appstate = trans.TRAN_FAVOURITETRANSFER;
274 }
275
276 /**
277  * @description Parser for Bank Account List
278  * @param r responseJSON from the service
279  * @returns {Array} of Account objects
280  */
281 function parseAccountList(r) {
282     var al = [];
283     if(r.accounts) {
284         for(var i = 0;i < r.accounts.length;i++) {
285             var account = new Account();
```

```
286         account.accountIdentification =
r.accounts[i].accountIdentification;
287         account.institutionCode =
r.accounts[i].institutionCode;
288         account.type = r.accounts[i].type;
289         account.accountStatus =
r.accounts[i].accountStatus;
290         if(r.accounts[i].linkedAccount)
291             account.linkedAccount =
r.accounts[i].linkedAccount;
292         account.paymentInstrumentId =
r.accounts[i].paymentInstrumentId;
293         if(r.accounts[i].branchCode)
294             account.branchCode =
r.accounts[i].branchCode;
295         if(r.accounts[i].countryCode)
296             account.countryCode =
r.accounts[i].countryCode;
297         account.currencyCode =
r.accounts[i].currencyCode;
298         account.accountHolder =
r.accounts[i].accountHolder;
299         if(r.accounts[i].alias)
300             account.alias = r.accounts[i].alias;
301         if(r.accounts[i].balance)
302             account.balance = r.accounts[i].balance;
303         if(r.accounts[i].availableFunds)
304             account.availableFunds =
r.accounts[i].availableFunds;
305         if(r.accounts[i].outstandingAmount)
306             account.outstandingAmount =
r.accounts[i].outstandingAmount;
307         if(r.accounts[i].minimumAmount)
308             account.minimumAmount =
r.accounts[i].minimumAmount;
309         if(r.accounts[i].statementAmount)
```

Open Bank API

```
310         account.statementAmount =
r.accounts[i].statementAmount;
311         al.push(account);
312     }
313 }
314     return al;
315 }
316 /**
317  * @description Parser for Favourite List
318  * @param r responseJSON from the service
319  * @returns {Array} of Favourites objects
320  */
321
322     function parseAccountInfo(r) {
323         var account = new Account();
324         if(r.accountHolder)
325             account.accountHolder = r.accountHolder;
326         if(r.linkedAccount)
327             account.linkedAccount = r.linkedAccount;
328         if(r.balance)
329             account.balance = r.balance;
330         if(r.availableFunds)
331             account.availableFunds = r.availableFunds;
332         if(r.outstandingAmount)
333             account.outstandingAmount = r.outstandingAmount;
334         if(r.minimumAmount)
335             account.minimumAmount = r.minimumAmount;
336         if(r.statementAmount)
337             account.statementAmount = r.statementAmount;
338         return account;
339     }
340 /**
```

```
341     * @description Parser for Favourite List
342     * @param r responseJSON from the service
343     * @returns {Array} of Favourites objects
344     */
345
346     function parseFavouriteList(r) {
347         var al = [];
348         if (r.favourites) {
349             for (var i = 0; i < r.favourites.length; i++) {
350                 var favourite = new Favourites();
351                 favourite.favId =
r.favourites[i].favouriteIdentification;
352                 favourite.favType =
r.favourites[i].favouriteType;
353                 favourite.alias = r.favourites[i].alias;
354                 favourite.paymentType =
r.favourites[i].details.paymentType;
355                 if (r.favourites[i].details.toAccount) {
356                     favourite.accountIdentification =
r.favourites[i].details.toAccount.accountIdentification;
357                     favourite.paymentInstrumentId =
r.favourites[i].details.toAccount.paymentInstrumentId;
358                     favourite.to_account_type =
r.favourites[i].details.toAccount.type;
359                 } else if (r.favourites[i].details.billerInfo)
{
360                     favourite.billerName =
r.favourites[i].details.billerInfo.billerName;
361                     favourite.paymentInstrumentId =
r.favourites[i].details.billerInfo.billerPaymentInstrumentId;
362                     favourite.billerCode =
r.favourites[i].details.billerInfo.billerCode;
363                 }
364                 al.push(favourite);
365             }
366         }
```

Open Bank API

```
367         return al;
368
369     }
370
371     /**
372     @function
373
374     @param response This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
375     */
376     if(loginParseXML)
377         delete loginParseXML;
378
379     function loginParseXML(response) {
380         if (response.customerId !== null) {
381             session.customerId = response.customer.id;
382             session.userId = response.userId;
383             pl.userId = response.userId;
384             session.customerSupportNo =
response.customerSupportNo;
385             session.idNumber = response.idNumber;
386             pl.idNumber = response.idNumber;
387             session.idType = response.idType;
388             pl.idType = response.idType;
389             session.authentication = new Authentication();
390             //session.authentication.pin =
response.authentication.pin;
391             session.authentication.token =
response.authentication.token;
392             pl.authentication = session.authentication;
393             if(response.customer) {
394                 var c = new Customer();
395                 c.customerId = response.customer.id;
396                 c.orgUnitId = response.customer.orgUnitId;
```



```
397         session.bankId = response.customer.orgUnitId;
398         c.blacklist = response.customer.blackListReason;
399         c.active = response.customer.active;
400         c.test = response.customer.test;
401         c.displayName = response.customer.displayName;
402         c.language = response.customer.language;
403         c.timezone = response.customer.timeZone;
404         c.typeId = response.customer.customerTypeId;
405         c.cancelId =
response.customer.cancellationReasonId;
406         c.modeId = response.customer.txnReceiptModeId;
407
408         session.customer = c;
409     }
410 }
411 }
412
413 /**
414  * @description Parser for Transaction List
415  * @param r responseJSON from the service
416  * @returns {Array} of Transaction objects
417  */
418 function parseTransactionList(r) {
419     var txns = [];
420     if(r.transaction) {
421         for(var i = 0;i < r.transaction.length;i++) {
422             var txn = new Transaction();
423             txn.id = r.transaction[i].id;
424             txn.debitCredit = r.transaction[i].debitCredit;
425             txn.amount.value =
r.transaction[i].amount.value;
426             txn.amount.currency =
r.transaction[i].amount.currency;
```

Open Bank API

```
427         txn.date = r.transaction[i].date;
428         txn.postingDate = r.transaction[i].postingDate;
429         txn.account.accountIdentification =
r.transaction[i].account.accountIdentification;
430         txn.account.institutionCode =
r.transaction[i].account.institutionCode;
431         txn.account.type =
r.transaction[i].account.type;
432         txn.account.paymentInstrumentId =
r.transaction[i].account.paymentInstrumentId;
433         txn.account.code =
r.transaction[i].account.code;
434         txn.balance.value =
r.transaction[i].balance.value;
435         txn.balance.currency =
r.transaction[i].balance.currency;
436         txn.chèqueNumber =
r.transaction[i].chequeNumber;
437         txn.text1 = r.transaction[i].text1;
438         txn.text2 = r.transaction[i].text2;
439         txn.text3 = r.transaction[i].text3;
440         txn.text4 = r.transaction[i].text4;
441
442         txns.push(txn);
443     }
444 }
445     return txns;
446 }
447
448 /**
449  * @description parser for currencies response
450  * @param r response data
451  * @returns {Array} of parsed currency objects with id and name
properties
452  */
453     function parseCurrencies(r) {
```

```
454     var currencies = [];  
455     if(r.lookupEntities != null) {  
456         for(var i = 0;i < r.lookupEntities.length;i++) {  
457             var currency = {};  
458             currency.id = r.lookupEntities[i].id;  
459             currency.name = r.lookupEntities[i].name;  
460             currencies.push(currency);  
461         }  
462     }  
463     return currencies;  
464 }  
465  
466 /**  
467  * @description parses the exchange rate fields from the  
468  * server.  
469  * @param r  
470  * @returns {Object} containing exchange rate fields  
471  */  
472 function parseExchangeRate(r) {  
473     var er = {};  
474     if(r.exchangeRate != null) {  
475         er["From Amount"] = r.exchangeRate.fromAmount;  
476         er["To Amount"] = r.exchangeRate.toAmount;  
477         er["Rate"] = r.exchangeRate.rate;  
478         er["From Currency"] = r.exchangeRate.fromCurrency;  
479         er["To Currency"] = r.exchangeRate.toCurrency;  
480     }  
481     return er;  
482 }  
483  
484 /**
```

Open Bank API

```
485     * @description response handler for cheque status web service
calls
486     * @param r responseXML/JSON
487     */
488     function getChequeStatusResponse(r) {
489         hideProgress();
490         var rCode = getStatusCode(r);
491         if (rCode == '0') {
492             var status = parseChequeStatus(r);
493             $('#csr_status').html(status);
494             $.mobile.changePage("chequestatusresult.html",
page_opt);
495         } else {
496             _alertBox('Failed to get cheque status.\n' +
getErrorMessage(r));
497         }
498         appstate = trans.TRAN_IDLE;
499     }
500
501     /**
502     * @description Parser for cheque status call response
503     * @param r
504     * @returns {String}
505     */
506     function parseChequeStatus(r) {
507         return r.chequeStatus?r.chequeStatus:"UNKNOWN";
508     }
509
510     /**
511     * @description response handler for cheque stop payment
request
512     * @param r
513     */
514     function getChequeStopResponse(r) {
```

```
515         hideProgress();
516         var rCode = getStatusCode(r);
517         if (rCode == '0') {
518             _alertBox('Stop cheque request have been placed
519 successfully.');
```

```
518             _alertBox('Stop cheque request have been placed
519 successfully.');
```

```
519             $.mobile.changePage("chequemgt.html", page_opt);
520         } else {
521             _alertBox('Error:\n' + getErrorMessage(r));
522         }
523         appstate = trans.TRAN_IDLE;
524     }
525
526     /**
527     * @description Callback handler for Chequebook placements
528     * @param r
529     */
530     function placeChequeBookRequestBack(r) {
531         hideProgress();
532         var rCode = getStatusCode(r);
533         if (rCode == '0') {
534             _alertBox('Cheque book request have been placed
535 successfully.');
```

```
534             _alertBox('Cheque book request have been placed
535 successfully.');
```

```
535             goMBBack();
536         } else {
537             _alertBox('Failed to place cheque book request.\n' +
538 getErrorMessage(r));
539         }
540         appstate = trans.TRAN_IDLE;
541     }
542
543     /**
544     @function
```

Open Bank API

```
545     @param r This is the XML document/JSON object that is
546     */
547     var logoutBack = function(r) {
548         hideProgress();
549
550         var rCode = getStatusCode(r);
551         if (appstate == trans.TRAN_LOGOUT) {
552             if(rCode == '0') {
553                 $.mobile.changePage("login.html", page_opt);
554                 session = new LoginSession();
555             } else if (rCode == '353') {
556                 _alertBox('Logout failed! ' +
557                 getErrorMessage(r));
558                 $.mobile.changePage("login.html", page_opt);
559                 session = new LoginSession();
560             } else {
561                 _alertBox('Logout failed! ' +
562                 getErrorMessage(r));
563             }
564             appstate = trans.TRAN_IDLE;
565         };
566     }
567     /
568     *****
569     *****
570     * Functions
571     */
572     /**
573     @function
```

```
573     @param r This is the XML document/JSON object that is
574     */
575     function getStatusCode(r) {
576         return r.Status.code.toString();
577     }
578
579     /**
580     @function
581
582     @param r This is the XML document/JSON object that is
583     returned by the AJAX call made by the MobiliserClient.
584     @param entryName This is string of the entry node name in
585     XML document/JSON object.
586     */
587     function parseReturnEntry(r, entryName) {
588         return r[entryName];
589     }
590
591     /**
592     @function
593
594     @param r This is the XML document/JSON object that is
595     returned by the AJAX call made by the MobiliserClient.
596     @param entryName This is string of the entry node name in
597     XML document/JSON object.
598     @param attrName This is string of attribute name of the
599     entry node in XML document/JSON object.
600     */
601     function parseReturnEntryAttr(r, entryName, attrName) {
602         return r[entryName][attrName].toString();
603     }
604 }
```

Open Bank API

```
601     @function
602
603     @param r This is the XML document/JSON object that is
        returned by the AJAX call made by the MobiliserClient.
604     */
605     function getErrorMessage(r) {
606         var errValue = r.Status.value;
607         var errCode = r.Status.code.toString();
608         if(errValue !== null) {
609             if (errCode == '329' || errCode == '201')
610                 return "Check your username or password and try
        again!";
611             else if (errCode == '302')
612                 return "Your account is blocked! Please contact
        service provider.";
613             else if(errCode == '2517')
614                 return "Payer has no valid wallet for this
        transaction";
615             else if(errCode == '2507')
616                 return "Payee has no valid wallet for this
        transaction";
617             else
618                 return r.Status.value;
619         } else {
620             var msg = '';
621             switch (errCode) {
622                 case '1002': msg = "INVALID MSISDN: Indicates that (one
        of) the specified MSISDN(s) is invalid. MSISDNs always have to
        specified as +countryCodenetCodenummer"; break;
623                 case '1003': msg = "UNKNOWN TRANSACTION: Indicates that
        the (or a) transaction specified in the request does not exist.";
        break;
624                 case '1004': msg = "INVALID TRANSACTION: Indicates that
        the (or a) transaction specified in the request cannot be used/
        considered in the requested context."; break;
```



```

625         case '1005': msg = "INVALID TRANSACTION STATUS:
Indicates that the (or a) transaction specified in the request cannot
be used/considered due to its status."; break;

626         case '1006': msg = "INACTIVE CUSTOMER: Indicates that
the (or a) customer specified in the request cannot be used/
considered because he is inactive"; break;

627         case '1008': msg = "BLACKLISTED CUSTOMER"; break;

628         case '1050': msg = "The transaction is locked by
another process and cannot be handled right now"; break;

629         case '1112': msg = "INACTIVE USER: Indicates that the
customer's status is inactive - see database
CUSTOMER_STATUS.BOL_IS_ACTIVE."; break;

630         case '1114': msg = "CUSTOMER BLOCKED: Indicates that
the customer is currently blocked due to the number of times wrong
credentials have been specified."; break;

631         case '1115': msg = "BLACKLISTED USER: Indicates that
the customer is blacklisted - see database
CUSTOMER.ID_BLACKLISTREASON."; break;

632         case '1301': msg = "SESSION ACTIVE WARN: Indicates that
a login failed because the customer has still an active session -
this prevents logging in multiple times with just one customer.";
break;

633         case '2231': msg = "AIRTIME LIMIT EXCEEDED"; break;

634         case '2232': msg = "CO-USER LIMIT EXCEEDED"; break;

635         case '2259': msg = "VOUCHER SALE HANDLER FAILURE";
break;

636         case '2299': msg = "TRANSACTION FAILED PERMANENTLY";
break;

637

638         //mBanking error

639

640         case '3100': msg = Str_do_not_honor; break;

641         case '3102': msg = Str_suspected_fraud; break;

642         case '3110': msg = Str_already_registerd; break;

643         case '3115': msg = Str_request_function; break;

644         case '3116': msg = Str_no_sufficient_fund; break;

645         case '3117': msg = Str_incorrect_pin; break;

646         case '3119': msg = Str_transaction_not_permitted;
break;

```

Open Bank API

```
647         case '3120': msg = Str_not_permitted_to_terminal;
break;
648         case '3125': msg = Str_retry_exceeded; break;
649         case '3126': msg = Str_invalid_pin_block; break;
650         case '3127': msg = Str_pin_length; break;
651         case '3129': msg = Str_suspected_card; break;
652         case '3180': msg = Str_unknown_userid; break;
653         case '3181': msg = Str_mbanking_session; break;
654         case '3200': msg = Str_request_in_progress; break;
655         case '3210': msg = Str_invalid_amount; break;
656         case '3211': msg = Str_withdrawal_amount; break;
657         case '3212': msg = Str_withdrawal_frequency; break;
658         case '3220': msg = Str_invalid_account; break;
659         case '3221': msg = Str_no_credit_account; break;
660         case '3222': msg = Str_no_investment_account;
break;
661         case '3223': msg = Str_no_current_account; break;
662         case '3224': msg = Str_no_saving_account; break;
663         case '3250': msg = Str_completed_partially; break;
664         case '3300': msg = Str_account_suspended; break;
665         case '3301': msg = Str_suspected_malfunction;
break;
666         case '3302': msg = Str_technical_problem; break;
667         case '3310': msg = Str_card_frequency; break;
668         case '3320': msg = Str_bill_payment_frequency;
break;
669         case '3330': msg = Str_own_account_frequency;
break;
670         case '3331': msg = Str_other_account_frequency;
break;
671         case '3332': msg = Str_interbank_frequency; break;
672         case '3340': msg = Str_mobile_reload_frequency;
break;
673         case '3341': msg = Str_mobile_transfer_intrabank;
break;
```

```

674         case '3342': msg = Str_mobile_transfer_interbank;
break;
675         case '3350': msg = Str_stop_cheque_frequency;
break;
676         case '3351': msg = Str_requaset_check_frequency;
break;
677         case '3901': msg = Str_invalid_response; break;
678         case '3902': msg = Str_invalid_transaction; break;
679         case '3904': msg = Str_format_error; break;
680         case '3907': msg = Str_card_issuer_switch_inoperative;
break;
681         case '3911': msg = Str_invalid_request; break;
682         case '3913': msg = Str_duplicate_transmission;
break;
683         case '3915': msg = Str_response_recieved_late;
break;
684
685         default: msg = "Unkown Error!"; break;
686     }
687     return msg;
688 }
689 }
690
691 function epoch2UTC(epoch) {
692     var utcDate = new Date(epoch);
693     return utcDate.getFullYear()+'-'+pad2(utcDate.getMonth()+1)+'-'
+pad2(utcDate.getDate())+
694         'T'+pad2(utcDate.getHours())
+':'+pad2(utcDate.getMinutes())+':'+pad2(utcDate.getSeconds());
695 }
696
697 /**
698     @function
699     @description This function can be used for debugging
purposes to display the contents of an XML message.

```

Open Bank API

```
700     Depending on the browser capabilities, the data will be
serialized.
701
702     @param xmlDoc This is the XML data that will be shown in the
alert box.
703     */
704     function alertXML(xmlDoc) {
705         if (xmlDoc.xml) // IE
706             _alertBox(xmlHttpRequest.responseXML.xml);
707         else // MOZ
708             _alertBox((new
XMLSerializer()).serializeToString(xmlDoc));
709     }
710
711     /**
712     @param message Message to display in the alert box
713     */
714     function _alertBox(message) {
715         if (typeof navigator.notification != 'undefined' && typeof
navigator.notification.alert == 'function')
716             navigator.notification.alert(message, null, "Mobiliser
Smartphone", "OK");
717         else
718             alert(message); // default alert function to be
used
719     }
720
721     /**
722     @param message Message to display in the confirm box
723     */
724     function _confirmBox(message, confirmCallback) {
725         if (typeof navigator.notification != 'undefined' && typeof
navigator.notification.confirm == 'function')
726             navigator.notification.confirm(message,
confirmCallback, "Mobiliser Smartphone", "OK, Cancel");
```

```
727         else {
728             confirmCallback(confirm(message));
729         }
730     }
731
732
733     /**
734     * @description parser for currencies response
735     * @param r response data
736     * @param filter an optional array which maps a new string for
737     name
738     * @returns {Array} of parsed currecy objects with id and name
739     properties
740     */
741     function parseLookupEntities(r, filter) {
742         var entities = [];
743         if(r.lookupEntities != null) {
744             for(var i = 0;i < r.lookupEntities.length;i++) {
745                 var entity = {};
746                 entity.id = r.lookupEntities[i].id;
747                 //Check if filter is provided
748                 if(filter) {
749                     //Check if filter contains the key
750                     entity.name = filter[r.lookupEntities[i].name]?
751 filter[r.lookupEntities[i].name]:r.lookupEntities[i].name;
752                 } else {
753                     //No filter store the name as it is returned by
754 service
755                     entity.name = r.lookupEntities[i].name;
756                 }
757                 entities.push(entity);
758             }
759         }
760         return entities;
761     }
```

Open Bank API

```
757     }
758
759     /**
760     * @description Mbanking TopUp
761     */
762     function MBankingTopUpBack(r) {
763         hideProgress();
764         var rCode = getStatusCode(r);
765         if (rCode == '0') {
766             $.mobile.changePage("prepaiddone.html", page_opt);
767             $('#tp_fromaccountbal').text(r.fromAccountBalance.currency + " " +
768             r.fromAccountBalance.value/100);
769             $('#refno3').text(r.transactionId);
770         } else {
771             _alertBox("Failed to top up! " +
772             getErrorMessage(r));
773         }
774     }
775     /**
776     * @description Callback handler for Mbanking
777     denominations
778     * @param r responseJSON containing products
779     */
780     function denominationsBack(r){
781         hideProgress();
782         var rCode = getStatusCode(r);
783         if (rCode == '0') {
784             session.denominations = parseDenomination(r);
785             if(session.denominations.length < 1) {
786                 _alertBox("No products found for you.");
787             } else {
```

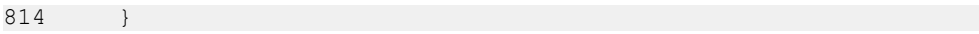
```

786         populateDenominations('#topup_denomination',
session.denominations);
787     }
788     } else {
789         _alertBox('getting denominations failed! ' +
getErrorMessage(r));
790     }
791 }
792
793
794
795 /**
796  * @description parser for denomination response
797  * @param r response data
798  * @returns {Array} of parsed denomination objects with id and
name properties
799  */
800 function parseDenomination(r) {
801     var denominations = [];
802     if(r.denomination) {
803         for(var i = 0;i <r.denomination.length;i++) {
804             var denomination = new Denomination();
805             denomination.id =
r.denomination[i].denominationId;
806             denomination.amount = r.denomination[i].amount;
807             denomination.description =
r.denomination[i].description;
808             denomination.code = r.denomination[i].code;
809             denomination.subcode =
r.denomination[i].subCode;
810             denominations.push(denomination);
811         }
812     }
813     return denominations;

```

Open Bank API

```
814 }
```



mBanking API

The mBanking API provides mobile banking functionalit functionality, which includes providing the ability for customers to manage accounts, query exchange rates, and query and accept terms and conditions.

Global

acceptTermsAndConditionBack function

Callback handler for the T&C acceptance

Syntax

```
acceptTermsAndConditionBack(r)
```

Parameters

| Name | Type | Description |
|----------|------|---|
| <i>r</i> | | This is the XML document/JSON object that is returned by the AJAX call made by the MobiliserClient. |

Source

SY_MB_Transactions.js line 192 on page 314.

getCurrenciesBack function

ResponseHandler for getCurrencies web service call

Syntax

```
getCurrenciesBack(r)
```

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

Source

SY_MB_Transactions.js line 52 on page 309.

getCustomerServicePackageRequestBack function

Callback handler for Getting Customer Service packages

Syntax

`getCustomerServicePackageRequestBack(r)`

Parameters

| Name | Type | Description |
|----------|------|---|
| <i>r</i> | | responseXML containing service package or error description |

Source

SY_MB_Transactions.js line 124 on page 312.

getExchangeRateBack function

callback handler for getting the exchange rate back

Syntax

`getExchangeRateBack(r)`

Parameters

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

Source

SY_MB_Transactions.js line 92 on page 310.

getNetworkProvidersBack function

Callback handler for GetNetworkProvider service call

Syntax

`getNetworkProvidersBack()`

*Source**SY_MB_Transactions.js line 17 on page 308.***mbLoginBack function**

Callback handler for the MB specific login check for T&C acceptance Restricted Number and Service Package restrictions

*Syntax*mbLoginBack(*r*)*Parameters*

| Name | Type | Description |
|----------|------|--|
| <i>r</i> | | This is the XML document/ JSON object that is returned by the AJAX call made by the MobiliserClient. |

*Source**SY_MB_Transactions.js line 168 on page 313.***NetworkProvider function**

Network Provider class containing details of NetworkProviders

Syntax

NetworkProvider()

*Source**SY_MB_Data_Objects.js line 13 on page 302.***parseCurrencies function**

parser for currencies response

*Syntax*parseCurrencies(*r*) { Array }*Parameters*

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

Returns

of parsed currency objects with id and name properties

Type:

Array

Source

SY_MB_Transactions.js line 75 on page 310.

parseCustomerServicePackage function

Parser for CustomerServicePackage requests

Syntax

`parseCustomerServicePackage(r)`

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | responseXML |

Returns

ServicePackage object

Source

SY_MB_Transactions.js line 145 on page 312.

parseNetworkProviders function

Parses the response of NetworkProvider service call

Syntax

`parseNetworkProviders(r)`

Parameters

| Name | Type | Description |
|----------|------|-------------|
| <i>r</i> | | |

Source

SY_MB_Transactions.js line 34 on page 309.

sendTxnAsSMSBack function

callback handler for sending txn as SMS

Syntax

```
sendTxnAsSMSBack( r )
```

Parameters

| Name | Type | Description |
|----------|------|---------------|
| <i>r</i> | | response data |

Source

SY_MB_Transactions.js line 109 on page 311.

ServicePackage function

Service Package for a particular customer

Syntax

```
ServicePackage( service, list )
```

Parameters

| Name | Type | Description |
|----------------|------|------------------------|
| <i>service</i> | | package id |
| <i>list</i> | | of allowed alert types |

Source

SY_MB_Data_Objects.js line 23 on page 302.

Source code

SY_MB_Data_Objects.js

A set of data objects utilized by the Mobiliser Smartphone application for mBanking services.

```

1      /**
2      * @overview
3      * A set of data objects utilized by the Mobiliser Smartphone
application for mBanking services.
```

mBanking API

```
4      *
5      * @name SY_MB_Data_Objects.js
6      * @author SAP AG
7      * @version 1.0
8      */
9
10     /**
11     * Network Provider class containing details of
12     NetworkProviders
13     */
14     function NetworkProvider() {
15         this.id = ''; //id of the provider
16         this.provider = ''; //displayStr for provider
17     }
18     /**
19     * @description Service Package for a particular customer
20     * @param service package id
21     * @param list of allowed alert types
22     */
23     function ServicePackage() {
24         this.servicePackageId = '';
25         this.supportedAlertList = [];
26         this.supportedChannel = [];
27     }
28
29     /**
30     * Extend the existing Setting to point to mbanking service
31     */
32     Setting.prototype.mbwsname = 'mobiliser/rest/spmmbanking';
33
34     /**
```

```

35     * Extend the existing accounts object to have additional
member variable
36     */
37     Accounts.prototype.accountIdentification = '';
38     Accounts.prototype.status = 0;
39
40     /**
41     * Extend the existing LoginSession object to have additional
member variables
42     */
43     LoginSession.prototype.customerServicePackage = new
ServicePackage();
44     LoginSession.prototype.tc = [];
45     LoginSession.prototype.exchangeRate = '';

```

SY_MB_Mobiliser.js

A thin JavaScript web service client that accesses the Mobiliser platform.

```

1     /**
2     * @fileOverview A thin JavaScript web service client that
accesses the Mobiliser platform.
3     * It provides an abstraction layer to communicate with the
system and returns
4     * XML documents or JSON as a result.
5     *
6     * @name SY_MB_Mobiliser.js
7     * @author SAP AG
8     * @version 1.0
9     */
10
11     MobiliserClient.prototype.mcurl = function() {
12         var url = setting.protocol + setting.ipaddress;
13
14         if(setting.port)
15             url += ":" + setting.port;
16

```

mBanking API

```
17         if(typeof setting.mbwsname !== 'undefined')
18             url += "/" + setting.mbwsname;
19
20         return url;
21     };
22
23     /**
24     * @description Get the exchange rate between two currencies
25     * from the server
26     * @param org
27     *           id of the organisation
28     * @param from
29     *           the from currency
30     * @param to
31     *           the to currency
32     */
33     MobiliserClient.prototype.getExchangeRate =
34     function(responseBack, from, to) {
35         var pl = new Object();
36         pl.origin = "android";
37         pl.traceNo = UUIDv4();
38         pl.AuditData = setting.appinfo;
39         pl.bankId = '0000';
40         pl.fromCurrency = from;
41         pl.toCurrency = to;
42         var mbankingService = {};
43         jQuery.extend(mbankingService, smartphoneService);
44         mbankingService.url = mc.mcurl();
45         mbankingService.post(pl, "getMBExchangeRate",
46         responseBack);
47     };
48
49     // Needed to override this as there was no way to get the txn
50     id inside
```



```

47     // the txn detail page
48     /**
49     * @description Call to send transaction details as SMS
50     * @param responseBack
51     *           callback handler after the call is successful
52     * @param id
53     *           of the transaction
54     */
55     MobiliserClient.prototype.sendTxnAsSMS =
function(responseBack, id) {
56         var pl = new Object();
57         pl.origin = setting.origin;
58         pl.traceNo = UUIDv4();
59         pl.AuditData = setting.appinfo;
60         pl.userId = session.username;
61         pl.userMsisdn = session.msisdn;
62         pl.bankId = session.customer.orgUnitId;
63         pl.txnId = id;
64         //TODO: check if this is available to smartphone endpoint?
cannot find it.
65         smartphoneService.post(pl, "sendTxnDetailsAsSMS",
responseBack);
66     };
67
68     /**
69     * @description Web service call to get the list of network
providers
70     * @param responseBack
71     *           Indicates which function to be called when a
response is received.
72     */
73     MobiliserClient.prototype.getCustomerServicePackageRequest =
function(responseBack) {
74         var pl = new Object();

```

mBanking API

```
75         pl.origin = setting.origin;
76         pl.traceNo = UUIDv4();
77         pl.AuditData = setting.appinfo;
78         pl.customerId = session.customer.id;
79         //TODO: check if this is available to smartphone endpoint?
cannot find it.
80         smartphoneService.post(pl, "getCustomerServicePackage",
responseBack);
81     };
82
83     /**
84     * @description checking MB specific login logic
85     * @param responseBack
86     *         Indicates which function to be called when a
response is received.
87     * @param customerId
88     *         Indicates which customer to add with default
package
89     */
90     MobiliserClient.prototype.checkMBLogin =
function(responseBack) {
91         var pl = new Object();
92         pl.origin = setting.origin;
93         pl.traceNo = UUIDv4();
94         pl.bankId = session.customer.orgUnitId;
95         pl.userMsisdn = session.msisdn;
96         pl.AuditData = setting.appinfo;
97         var mbankingService = {};
98         jQuery.extend(mbankingService, smartphoneService);
99         mbankingService.url = mc.mburl();
100        mbankingService.post(pl, "mBankingingChecks",
responseBack);
101
102        // TODO: mBankingingChecks seems to be defined in web
service ... remove "ing".
```

```

103     };
104
105     /**
106     * @description accepts T&C on behalf of customer
107     * @param responseBack
108     *          Indicates which function to be called when a
response is received.
109     * @param customerId
110     *          Indicates which customer to add with default
package
111     */
112     MobiliserClient.prototype.acceptTermsAndCondition =
function(responseBack, customerId, optInTermsId) {
113         var pl = new Object();
114         pl.origin = setting.origin;
115         pl.traceNo = UUIDv4();
116         pl.AuditData = setting.appinfo;
117         pl.userMsisdn = session.msisdn;
118         pl.bankId = session.customer.orgUnitId;
119         pl.optInTermsId = optInTermsId;
120         var mbankingService = {};
121         jQuery.extend(mbankingService, smartphoneService);
122         mbankingService.url = mc.mcurl();
123         mbankingService.post(pl, "acceptTerms", responseBack);
124     };
125

```

SY_MB_Transactions.js

This is the actual response handling and processing of the web service calls that are made by the MobiliserClient class for mBanking.

```

1     /**
2     * @overview
3     * This is the actual response handling and processing of the
web service calls

```

mBanking API

```
4      * that are made by the MobiliserClient class for mBanking.
Any manipulation of

5      * the response, extraction of needed data and the making of
Data Objects (DO)

6      * out of the XML document would be inserted here.

7      *

8      * @name SY_MB_Transactions.js

9      * @author SAP AG

10     * @version 1.0

11     *

12     */

13

14     /**

15     * @description Callback handler for GetNetworkProvider
service call

16     */

17     function getNetworkProvidersBack(r) {

18         hideProgress();

19         var rCode = getStatusCode(r);

20         if (rCode == '0') {

21             networkProviders = parseNetworkProviders(r);

22             listNetworkProviders(networkProviders);

23         } else {

24             _alertBox(Str_msg_failupdatealert +
getErrorMessage(r));

25             $('#aas_title').html(Str_msg_editalertfailed);

26         }

27         appstate = trans.TRAN_IDLE;

28     }

29

30     /**

31     * @description Parses the response of NetworkProvider service
call

32     * @param r
```

```
33     */
34     function parseNetworkProviders(r) {
35         networkProviders = []; //Remove old entries from the
array.
36         if(r.lookupEntities != null) {
37             for(var i = 0;i < r.lookupEntities.length;i++) {
38                 var oi = new NetworkProvider();
39                 //Parsing logic goes here
40                 oi.id = r.lookupEntities[i].id;
41                 oi.provider = r.lookupEntities[i].name;
42                 networkProviders.push(oi);
43             }
44         }
45         return networkProviders;
46     }
47
48     /**
49     * @description ResponseHandler for getCurrencies web service
call
50     * @param r
51     */
52     function getCurrenciesBack(r) {
53         hideProgress();
54         var rCode = getStatusCode(r);
55         if (rCode == '0') {
56             currencies = parseCurrencies(r);
57             if(currencies && currencies.length > 0) {
58                 listCurrencies('#fx_fromcurrency', currencies);
59                 listCurrencies('#fx_tocurrency', currencies);
60             } else {
61                 //TODO Show error message and go back??
62             }
63         } else {
```

mBanking API

```
64         _alertBox(Str_msg_failgetcurrency +
getErrorMessage(r));
65         //TODO: Shall we push user back??
66     }
67     appstate = trans.TRAN_IDLE;
68 }
69
70 /**
71  * @description parser for currencies response
72  * @param r response data
73  * @returns {Array} of parsed currecy objects with id and name
properties
74  */
75     function parseCurrencies(r) {
76         var currencies = [];
77         if(r.lookupEntities != null) {
78             for(var i = 0;i < r.lookupEntities.length;i++) {
79                 var currency = {};
80                 currency.id = r.lookupEntities[i].id;
81                 currency.name = r.lookupEntities[i].name;
82                 currencies.push(currency);
83             }
84         }
85         return currencies;
86     }
87
88 /**
89  * @description callback handler for getting the exchange rate
back
90  * @param r response data
91  */
92     function getExchangeRateBack(r) {
93         hideProgress();
```

```

94         var rCode = getStatusCode(r);
95         if (rCode == '0') {
96             session.exchangeRate = r.exchangeRate;
97             $.mobile.changePage("fxresult.html", {transition:
"none", changeHash: false});
98         } else {
99             _alertBox(Str_msg_failgetfxrate +
getErrorMessage(r));
100             //TODO: Shall we push user back??
101         }
102         appstate = trans.TRAN_IDLE;
103     }
104
105     /**
106     * @description callback handler for sending txn as SMS
107     * @param r response data
108     */
109     function sendTxnAsSMSBack(r) {
110         hideProgress();
111         var rCode = getStatusCode(r);
112         if (rCode == '0') {
113             _alertBox(Str_msg_txnassmssent);
114         } else {
115             _alertBox(Str_msg_failsendtxnsms +
getErrorMessage(r));
116         }
117         appstate = trans.TRAN_IDLE;
118     }
119
120     /**
121     * @description Callback handler for Getting Customer Service
packages
122     * @param r responseXML containing service package or error
description

```

mBanking API

```
123     */
124     function getCustomerServicePackageRequestBack(r) {
125         hideProgress();
126         var rCode = getStatusCode(r);
127         if (rCode == '0') {
128             session.customerServicePackage =
parseCustomerServicePackage(r);
129             if(session.customerServicePackage &&
session.customerServicePackage.supportedChannel &&
$.isArray(setting.origin,
session.customerServicePackage.supportedChannel)) {
130                 applyFilterForAddAlerts(session.customerServicePackage.supportedAlertList);
131             } else {
132                 _alertBox(Str_msg_channelnotsupport);
133             }
134         } else {
135             _alertBox(Str_msg_failgetservicepackage +
getErrorMessage(r));
136         }
137         appstate = trans.TRAN_IDLE;
138     }
139
140     /**
141     * @description Parser for CustomerServicePackage requests
142     * @param r responseXML
143     * @returns ServicePackage object
144     */
145     function parseCustomerServicePackage(r) {
146         var sp = new ServicePackage();
147
148         if(r.servicePackage &&
r.servicePackage.servicePackageId)
149             sp.servicePackageId =
r.servicePackage.servicePackageId;
```



```

150         if(r.servicePackage &&
r.servicePackage.supportedAlertList &&
r.servicePackage.supportedAlertList.supportedAlert.length) {
151             for(var i = 0;i <
r.servicePackage.supportedAlertList.supportedAlert.length;i++) {
152                 sp.supportedAlertList.push(r.servicePackage.supportedAlertList.supp
ortedAlert[i].alertName);
153             }
154         }
155         if(r.servicePackage &&
r.servicePackage.supportedChannellist &&
r.servicePackage.supportedChannellist.supportedChannel.length) {
156             for(var i = 0;i <
r.servicePackage.supportedChannellist.supportedChannel.length;i++)
{
157                 sp.supportedChannel.push(r.servicePackage.supportedChannellist.supp
ortedChannel[i].channelId);
158             }
159         }
160         return sp;
161     }
162
163     /**
164     * @function
165     * @description Callback handler for the MB specific login
check for T&C acceptance Restricted Number and Service Package
restrictions
166     * @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
167     */
168     var mbLoginBack = function(r) {
169         var rCode = getStatusCode(r);
170         if (appstate == trans.TRAN_LOGIN) {
171             if(rCode == '0') {
172                 session.customerServicePackage =
parseCustomerServicePackage(r);

```

mBanking API

```
173         } else if (rCode == '3183' || rCode == '3186') {
174             _alertBox(Str_alert_loignfailed + "\n" +
getErrorMessage(r));
175         } else if(rCode == '3184') {
176             session.customerServicePackage =
parseCustomerServicePackage(r);
177             if(r.tc && r.tc.length > 0) {
178                 session.tc = r.tc;
179             }
180         } else {
181             _alertBox(Str_msg_fail_because + Str_msg_code +
rCode + Str_msg_message + getErrorMessage(r));
182         }
183     }
184     return rCode;
185 };
186
187 /**
188  * @function
189  * @description Callback handler for the T&C acceptance
190  * @param r This is the XML document/JSON object that is
returned by the AJAX call made by the MobiliserClient.
191  */
192     var acceptTermsAndConditionBack = function(r) {
193         var rCode = getStatusCode(r);
194         if(rCode == '0') {
195             return true;
196         } else {
197             _alertBox("Error in accepting Terms And Condition." +
getErrorMessage(r));
198             return false;
199         }
200     };
```