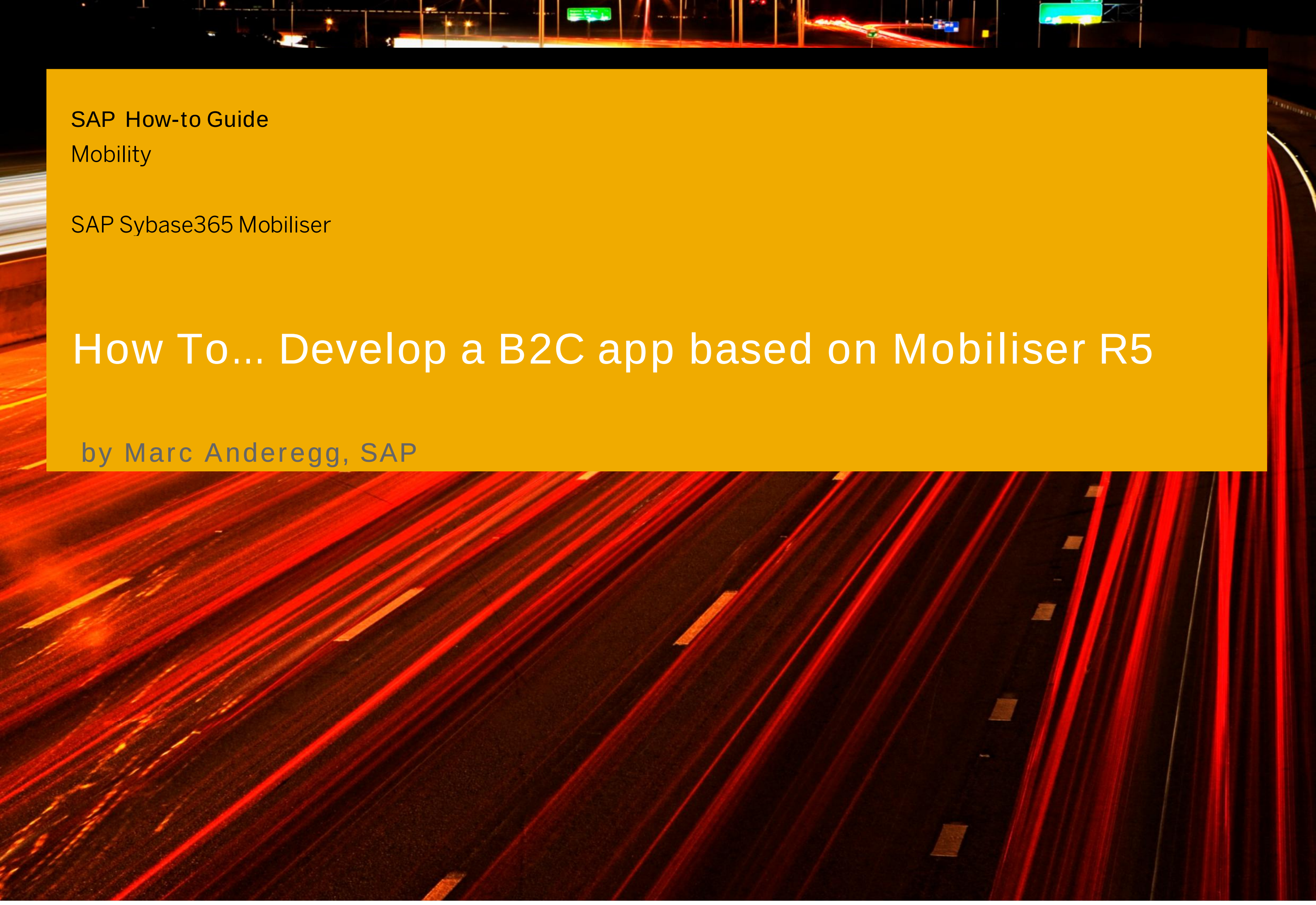
SAP How-to Guide
Mobility

SAP Sybase365 Mobiliser

How To... Develop a B2C app based on Mobiliser R5

by Marc Anderegg, SAP



Applicable Releases:

SAP Sybase365 Mobiliser release 5

Version 1.0

September 2012

The Best-Run Businesses Run SAP™

The Best-Run Businesses Run SAP™

Document History

Document Version	Description
1.00	First official release of this guide

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, F2 or ENTER.

Icons

Icon	Description
⚠	Caution
●	Note or Important
●	Example
⬆	Recommendation or Tip

Table of Contents

1. Introduction

This document gives a high-level view on how to develop in SAP Sybase365 Mobiliser new central services and a smartphone app calling these services. It shows how to make use of the Mobiliser platform/framework and will allow developing simple Business-to-Consumer (B2C) applications based on the currently available release 5 of Mobiliser.

This document is structured as follows: A short overview of the Mobiliser architecture shows the different components that belong to the system and highlights the spots where new functionality can be added to build new applications based on the Mobiliser platform and system. Furthermore, some fundamental concepts of the Mobiliser framework are described that form the basis on which a new B2C app can be developed for this platform. Based on these chapters about generic Mobiliser considerations we describe how to develop new services and mobile apps. First, the minimal development steps for new services are listed and explained on a high-level. Then the development of a smartphone app that would consume these new services is described. The detailed description of the single development steps together with screenshots based on an example app would be part of a separate HowToGuide.

This document is not taking into account the mCommerce part of Mobiliser (mBanking, mPayment, mTopUp, mRemittance) but rather focuses on how the Mobiliser platform can be used for new B2C apps.

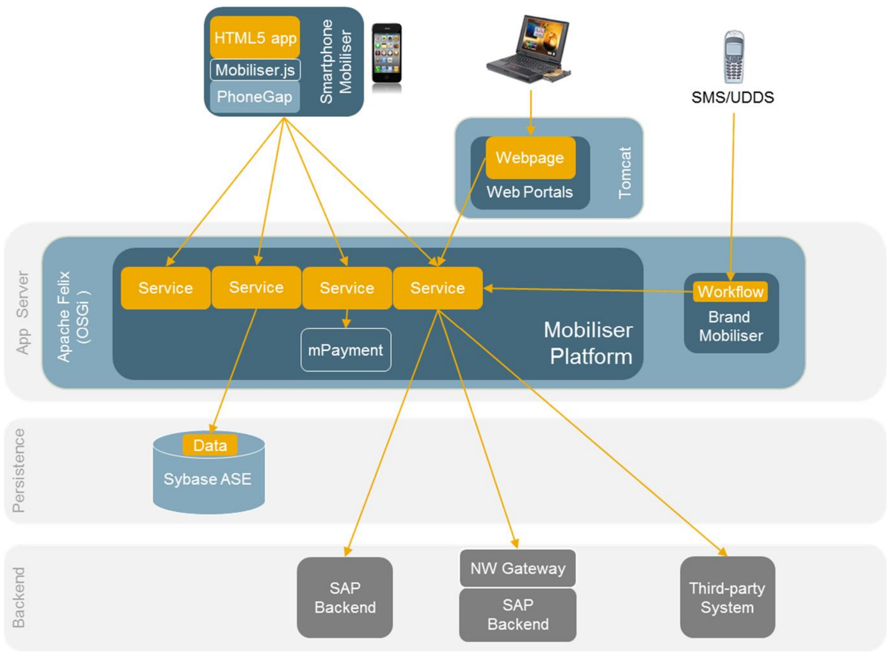
For more detailed information please refer to the official document *Mobiliser Framework Development Guide*, including Best Practice, Testing, Troubleshooting and Mobiliser Error Codes.

2. Overview of the Mobiliser architecture

Sybase365 Mobiliser is a system providing a platform designed by a service-oriented architecture and can be seen as set of sub-systems covering different aspects of a mobile system: generic platform and mCommerce services, smartphone application, SMS based application, portal access and backend integration.

The Mobiliser Platform is the heart of this set and is running on a Java application server based on OSGi, currently on the open source project Apache Felix of The Apache Software Foundation. The Mobiliser Platform is also known as Money Mobiliser and Sybase 365 mCommerce with its service sets mBanking, mPayment, mTopUp and mRemittance.

The following graphic illustrates a typical Mobiliser architecture overview and highlights in orange the components that can be developed for a new application, be it new services, a new smartphone app, a portal or classic phone SMS workflows.



The platform provides a framework that allows adding new services and hence building new applications for the B2C market. These services can deliver various services ranging from simple CRUD (Create, Read, Update, Delete) services to very complex transactional applications as mCommerce. They can store data locally via the persistence layer of Mobiliser, invoke existing functions like mPayment or integrate with other backend system. These services can be called from different parts, for example from Smartphone Mobiliser applications, Mobiliser Portals or Brand Mobiliser.

The Smartphone Mobiliser is based on HTML5 and PhoneGap. The JavaScript files that are provided with Smartphone Mobiliser allow the application to connect and call the services running on the Mobiliser platform. They contain the generic functions required for the connectivity, but also implement the mCommerce functionality for the client side.

The following chapters will explain the Mobiliser framework architecture and describes the mechanisms of service calls, and they show how to develop new services based on the framework and how to develop an app based on Smartphone Mobiliser.

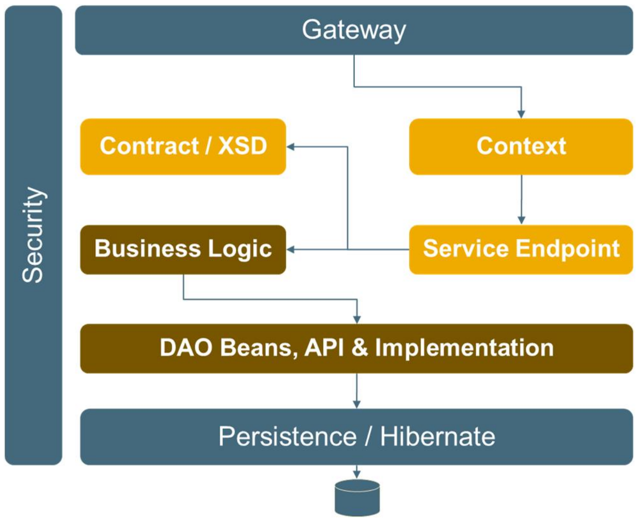
3. The Mobiliser framework architecture for B2C apps

3.1 Mobiliser Service-Oriented Architecture

The Mobiliser platform is based on a classical Service-Oriented Architecture (SOA) and allows the implementation of additional custom services making use of the Mobiliser framework. Any kind of service can be implemented ranging from basic CRUD (Create, Read, Update and Delete) services to very complex transaction processing, i.e. the mCommerce services. All services are exposed as SOAP and REST (XML/POST and JSON) Web Services and can be called by clients through any of these techniques.

The Mobiliser platform provides the framework required to orchestrate the services and the core functionalities of service gateway, security and persistence for the services. The service gateway provides functionality of different nature: Central Service Registry, Common Service Publishing, Common security and audit Layers, Standardized service definition and Multi-protocol support.

To add a new custom service, the minimum implementation requires the development of a service's contract with its context and the service endpoint which will contain the functionality of the service. For more complex processing, business logic can be implemented in separate classes or packages, or data can be persisted in the platform database by implementing DAO beans that make use of the persistence layer of the platform. Mobiliser is making use of the Hibernate technology for the persistence layer that is based on JPA, Java Persistence Architecture. DOA Beans are implemented by declaring the API and defining the implementation.



All services have a contract that defines the data structures and the interfaces of the service. The contract is defined, and by this mean documented, in a XSD file which would be the first step in the development process of a new service. The contract holds also the interface declaration by defining the Java Interfaces for the services that will be implemented in the service endpoints, mainly requests and responses.

The context is used to register the new service endpoints. The registration itself will be handled by the Mobiliser framework. The context of the service endpoints are defined in property files by describing the URL and the WSDL of the service in case of SOAP.

The service endpoint contains the implementation of the interface defined in the contract and the methods for all the functions of the contract. As mentioned earlier, this can be basic functions up to very complex implementations with data persistency and integration of mCommerce services, e.g. mPayment or other backend systems.

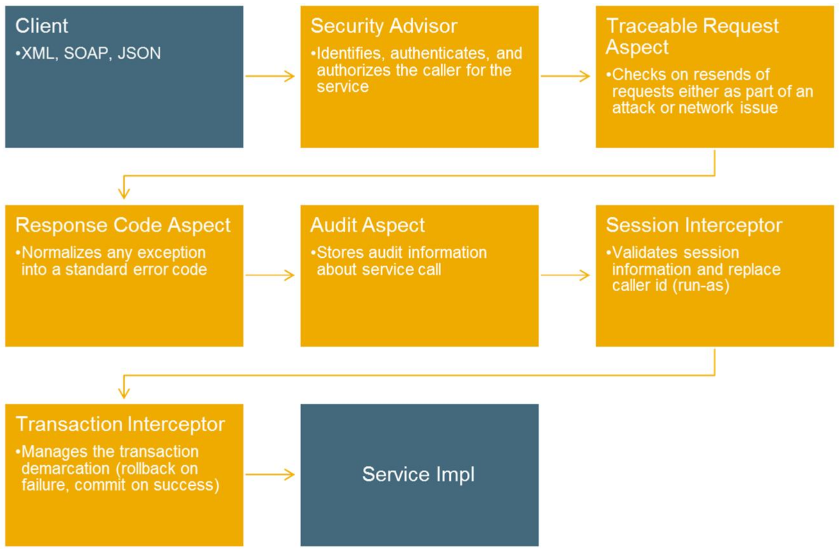
Business Logic is where the logic of the service is implemented. Even though it is not mandatory to structure and separate the logic from the service endpoint when developing a new service, it is for sure good practice. This is how mCommerce has been implemented.

3.2 Mobiliser framework services filters

Services that are exposed publicly can be called by a client like the Smartphone Mobiliser client, Mobiliser Portal, Brand Mobiliser, custom B2C client. Every service call will pass through a set of filters that are provided by the Mobiliser platform and that can be configured to match the requirements of the specific implementation.

The set of filters comprises:

- Security Advisor
- Traceable Request Aspect – to check requests against re-submission
- Response Code Aspect – to normalize faults into responses with proper error codes
- Audit Aspect – pluggable Audit Managers log the data by multiple means
- Session Interceptor – managing a second identity that is bound to a user session (beside standard caller credentials)
- Transaction Interceptor


Client
•XML, SOAP, JSON
Security Advisor
•Identifies, authenticates, and authorizes the caller for the service
Traceable Request Aspect
•Checks on resends of requests either as part of an attack or network issue
Response Code Aspect
•Normalizes any exception into a standard error code
Audit Aspect
•Stores audit information about service call
Session Interceptor
•Validates session information and replace caller id (run-as)
Transaction Interceptor
•Manages the transaction demarcation (rollback on failure, commit on success)
Service Impl

More details can be found in chapter "Component Architecture" of the official document Mobiliser Framework, Development Guide".

3.3 Role based access concept

Mobiliser allows the implementation of a role based access concept by assigning consumers to roles that are collections of privileges.

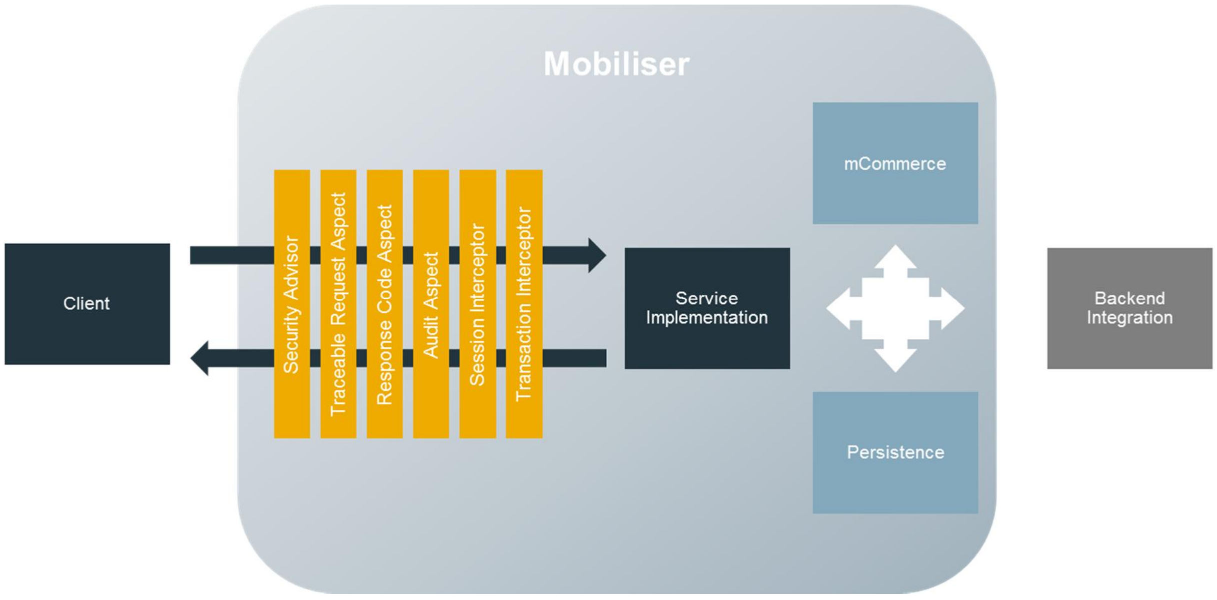
A list of access privileges can be attributed to each service endpoint's interface that will be called by a client. A default privilege (MOBILISER_ACCESS) is required to access the interface that can be changed by configuration in the context.properties file. Additional privileges can be defined to refine the access to the service.

3.4 Basic overview on how to develop new services

As described in the previous chapters, each service call runs through a set of service filters before it reaches the service endpoints. The developer of a new service does not need to care about theses filters as they are provided by the framework and will be invoked automatically by the runtime environment. He can take influence for example by defining access privileges to his services' interface, hence refining the role access to his new services.

The framework also exposes the service to client calls in different types of protocols, SOAP, XML/HTTP and JSON,

It is up to the developer to implement the service and its functionality. He can implement a simple request handling, use the persistence layer of the Mobiliser framework based on the Java Persistence Architecture (JPA) and Hibernate, invoke mCommerce services like mPayment or integrate with other backend systems. The Mobiliser mCommerce services are available for consumption by new developed services through the OSGi service registry.



The developer will have to setup a development environment for Java development that is based on Eclipse and that makes use of open source development tools like Maven, Spring, Hibernate. He will also have to import the Mobiliser development artifacts to his projects; these artifacts include the Mobiliser framework.

The development of a new service must adhere to the Mobiliser framework by implementing the service's contract, endpoint and context. The OSGi specific information and Spring configurations are defined in a couple of XML files. Maven is handling the build process and the instructions for the building are defined in Maven's build files called pom.xml.

The outcome of the build process will be a complete Mobiliser distribution with all the OSGi bundles that can be installed on a server which has a running database, e.g. Sybase ASE. The distribution will be contained in a ZIP file with a predefined structure that will include a complete Mobiliser runtime environment with the OSGi application server Apache Felix. The new services will be located in the directory 'money', SQL statements will be located in the directory 'sql'.

4. How to develop a new service in Mobiliser

4.1 Prerequisites for development environment

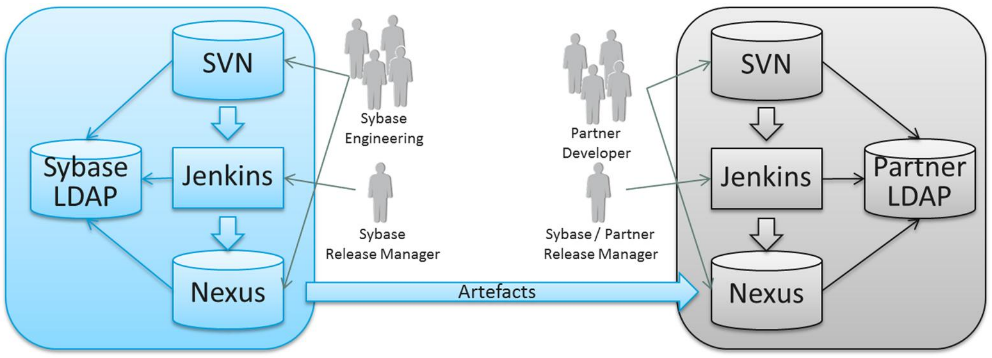
The development environment is based on Eclipse and the build processes use the open source build management tool Maven, part of the Apache project. Different technologies are used by Mobiliser and the corresponding tools need to be installed in the development environment:

Eclipse	http://www.eclipse.org/
Maven	http://maven.apache.org/
Spring	http://www.springsource.org/

There are no specific Mobiliser tools or plug-ins to support the development phase other than the standard open source tools of the technologies listed above or available publicly, e.g. tools to model and manage the database, editors for XML, HTML, JavaScript and CSS. The standard Java plug-ins for Eclipse will provide support for the plain Java development.

The Mobiliser artifacts are provided on a Nexus server and can be imported to the project repository of the development environment in which the new services shall be implemented. This would be the simplest way to get the artifacts.

A more professional and enterprise oriented way would be to setup a complete development environment with a versioning system, a continuous integration tool and access to a LDAP server. In the illustration SVN (Apache Subversion) is used as versioning system. Project Nexus is virtual community software written in PHP with a MySQL database. Jenkins is the continuous integration tool used to continuously build upon each commit and nightly builds. Please refer to the following link about how to setup the Nexus Repository Manager to build Jenkins: https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+Development+Environment+with+Nexus.



Once you have imported the standard artifacts in your Eclipse project, you will find a structure of modules that have names starting with the prefix com.sybase365.mobiliser.custom.project.

The relevant standard modules when writing new B2C services are in the following first block. This document focuses only on these modules:

- com.sybase365.mobiliser.custom.project.dist
- com.sybase365.mobiliser.custom.project.persistence.dao-api
- com.sybase365.mobiliser.custom.project.persistence.dao-hibernate

- com.sybase365.mobiliser.custom.project.persistence.dao-model
- com.sybase365.mobiliser.custom.project.services.context
- com.sybase365.mobiliser.custom.project.services.contract
- com.sybase365.mobiliser.custom.project.services.endpoint

The other modules will be needed to be customized when touching Brand Mobiliser, mCommerce or general jobs and events.

Standard modules relevant to Brand Mobiliser are:

- com.sybase365.mobiliser.custom.project.brand.client
- com.sybase365.mobiliser.custom.project.brand.states
- com.sybase365.mobiliser.custom.project.channels

Standard modules relevant to mCommerce are:

- com.sybase365.mobiliser.custom.project.businesslogic
- com.sybase365.mobiliser.custom.project.handlers.authentication
- com.sybase365.mobiliser.custom.project.handlers.billpayment
- com.sybase365.mobiliser.custom.project.handlers.exchangerate
- com.sybase365.mobiliser.custom.project.handlers.payment

Standard modules relevant to jobs and events are:

- com.sybase365.mobiliser.custom.project.ams
- com.sybase365.mobiliser.custom.project.jobs.cronjob-custom
- com.sybase365.mobiliser.custom.project.jobs.event-handler-custom
- com.sybase365.mobiliser.custom.project.jobs.event-model
- com.sybase365.mobiliser.custom.project.jobs.task-custom

Please refer to the Mobiliser document "Mobiliser Framework, Development Guide" for further details.

4.2 Development steps

The steps that need to be performed are on a high-level:

1. Implement the service contract
    a. Define complex type in beans.xsd
    b. Define request/response in requests.xsd
    c. Generate java classes
    d. Add new java interface
    e. Define user privileges to interface
2. Implement the service endpoint
    a. Implement java interface
    b. Add and implement service methods
    c. Handle exceptions
    d. Logging
3. Implement the service context
    a. Define service bean

The following sub-chapters describe these development steps.

4.2.1 Implement the service contract

In the service contract we will define the data structures in XSD files and based on these definitions we will let the system generate the corresponding Java classes with the required method interfaces. At this stage we will provide the user privileges for each interface that will allow implementing the role based aspect of Mobiliser on function level.

Module used is com.sybase365.mobiliser.custom.project.services.contract

4.2.1.1 Define complex type in beans.xsd

The provided standard file beans.xsd contains all data structures in XML and we can add the new data structures that are going to be processed by the new service. For convenience, the Eclipse environment can visualize the data model. Mobiliser is providing predefined types that shall be used for the new data structures. The length must be defined for each field. A good practice would be to use annotation and documentation tags with descriptions for each definition.

4.2.1.2 Define request / response in requests.xsd

For each data structure the request and response function must be defined as XML in the provided standard file requests.xsd.

4.2.1.3 Generate java classes

Based on the request/responses defined, the Java classes shall be generated by invoking the Maven XML build. The generated Java classes will be located in the contract API package.

4.2.1.4 Add new java interface

New Java interfaces have to be created that will allow the request method invocation and that will return the response data. The Eclipse wizard allows the Java interface generation.

4.2.1.5 Define user privileges to interface

To each Java interface created you shall add a list of privileges that will be required to call this service. The @RolesAllowed annotation defines this list.

Privileges are defined in the Mobiliser database and must be inserted to the database if not done previously. For this plain SQL statements can be added to the SQL files of the persistence module.

4.2.2 Implement the service endpoint

With the definition of the Service Endpoint you implement the interface that has been defined in the contract.

Module used is com.sybase365.mobiliser.custom.project.services.endpoint

Create a new class with the Eclipse wizard for the service implementation. Add the call service, populate the response. Add exception handling and logging.

4.2.3 Implement the service context

The new service needs to be defined as an OSGi service. To accomplish this, META_INF file needs to be edited and the service bean needs to be defined in XML files (bundle-context-endpoint.xml, bundle-context-osgi.xml, bundle-context-service-config.xml)

Module used is com.sybase365.mobiliser.custom.project.services.context

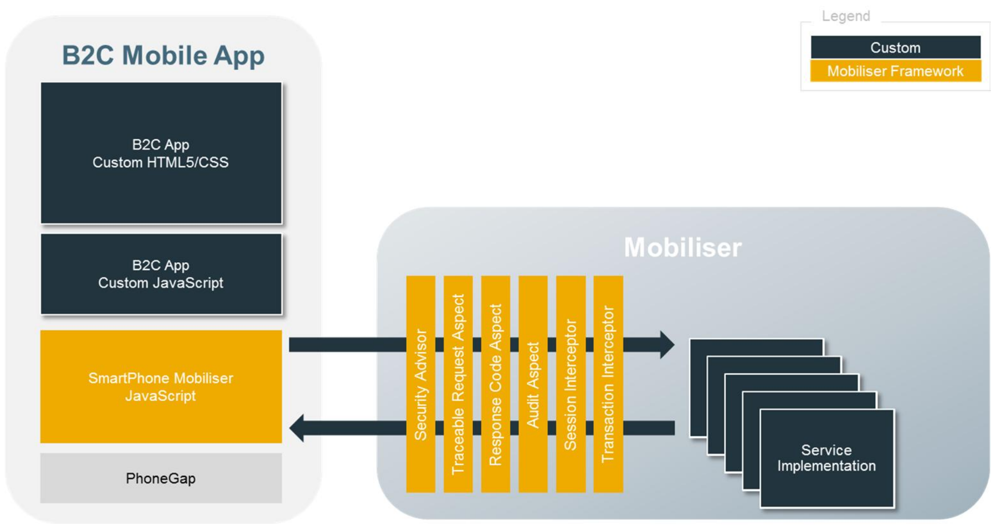
5. How to develop a mobile app

5.1 Overview of Smartphone Mobiliser architecture

It is important to understand that the services on the Mobiliser Platform are accessed by Web Service calls. Any client emitting service calls that use either SOAP or REST (XML and JSON) and that adhere to the Web Services' contract can connect to Mobiliser. Along the Mobiliser Platform an additional component called Smartphone Mobiliser is delivered that provides the integration part on the client side together with a template app for the mCommerce services. Smartphone Mobiliser can also serve as a template for any B2C client app and would need to be enhanced or modified to suite the custom requirements.

In general, Smartphone Mobiliser is a HTML5 (HTML5/CSS/JavaScript) app based on PhoneGap. The Mobiliser logic and connectivity is provided by some JavaScript files and the HTML5 file contains the presentation layer to access mCommerce functionality. The Mobiliser JavaScript files handle the connectivity to the Mobiliser Platform and each service call will have to pass through the service filter set described previously to access the service endpoint with the service's implementation.

A new B2C client app can leverage the technology provided by Smartphone Mobiliser; custom HTML5/CSS/JavaScript can be developed that call the B2C services on the central system.



As mentioned, any kind of mobile app can access the exposed Mobiliser services by adhering to the service call contract and protocols, one variant would be to use SAP UI5 or Sencha touch as a technology basis.

The following will focus on plain Smartphone Mobiliser.

5.2 How to develop a Smartphone Mobiliser app

Smartphone Mobiliser is providing client development artifacts for iOS (iPhone and iPad version), Android (phone and tablet version) and BlackBerry. As the app is HTML5 based the JavaScripts and HTML components/images are similar, mainly the structuring of the artifacts for the corresponding development environment defer between the different device types.

A classical development environment will be required depending on the device type. XCode for iOS, Eclipse for Android and BlackBerry. For Android you can use for example Eclipse Java EE IDE for Web Developers plus the Android Development Toolkit (ADT) and the Android SDK. Please refer to the Smartphone Mobiliser Setup Guide for pre-requirements and instructions on how to setup the development environment including how to import the Mobiliser artifacts.

Again, no specific Mobiliser tools are provided to support the development phase of a smartphone app.

The Mobiliser artifacts contain all required development objects like PhoneGap and jQuery Mobile. Once you have imported the client artifacts by selecting the import variant of an existing project, you should have the Mobiliser project structure. The name of the project can be changed (refactored) to any name. The structure will slightly vary depending on the device target (iOS, Android, BlackBerry).

For the development of the Smartphone Mobiliser app, three parts under the project directory 'www' are relevant:

- Smartphone Mobiliser standard JavaScript files - directory 'mobiliser'
- Smartphone Mobiliser HTML file - Index.html
- Smartphone Mobiliser application JavaScript and CSS files - directory 'app'

5.2.1 Smartphone Mobiliser JavaScript files

In the directory 'mobiliser' you fill find 4 JavaScript files that are worth to read and understand.

- serviceclient.js
- SY_Data_Objects.js
- SY_Mobiliser.js
- SY_Transactions.js

5.2.1.1 serviceclient.js

This file contains the JavaScript "Service Client" library for SOAP and JSON clients.

The SOAP client is more complex and is implemented by several JavaScript functions:

- SOAPClient
- SOAPClientAttributes - helper function
- SOAPClientParameters - helper function

The SOAP client has been implemented with dependencies to the mCommerce functionality vs. the JSON client that is more generic and simpler by implementing a single function:

- smartphoneService

Additionally, there are some utility functions:

- Base64
- ServiceErrorMessage
- ServiceTerminate

5.2.1.2 SY_Data_Objects.js

This JavaScript file contains a set of data objects utilized by the Mobiliser smartphone app. Most of these objects are mCommerce related, but some could be reused for the B2C app, e.g. Customer, TxnData, LoginSession, Timer and Setting.

Customer	contains basic information about the customer
TxnData	transaction data
LoginSession	stores all of the user's data throughout a login session
Accounts	stores information pertaining to certain accounts
Transaction	stores information pertaining to certain transaction
BillTypes	stores data about any invoice that the user has registered for
OpenBill	representing an open invoice which the user has to pay for a certain merchant
RegisteredBill	stores information about any invoice that the user has registered for
BalanceAlert	stores information about a balance alert
Timer	a timer function
Setting	connection configuration to connect to the backend server (protocol, IP address, port, Web Service name, SOAP Web Service path, JSON Web Service path)

5.2.1.3 SY_Mobiliser.js

This standard Mobiliser file contains a thin JavaScript Web Service client that accesses the Mobiliser platform and provides an abstraction layer to communicate with the system and returns XML documents as a result.

MobiliserClient	name of the object and uses the settings information out of SY_Data_Objects.js to connect
MobiliserClient.prototype.login	login function
MobiliserClient.prototype.logout	logout function
MobiliserClient.prototype.createSmsToken	create SMS function

All other functions are related to mCommerce.

5.2.1.4 SY_Transactions.js

This JavaScript file contains the actual response handling and processing of the Web Service calls that are made by MobiliserClient. Any manipulation of the response, extraction of needed data and the making of Data Objects out of the XML document/JSON object that is returned by the AJAX call made by MobiliserClient would be inserted here. The interesting functions for B2C apps would be:

- loginBack
- logoutBack
- createSmsTokenBack
- registerPasswordBack
- registerBack
- getStatusBack
- parseReturnedEntry
- parseReturnedEntryAttr
- parseIdentification
- loginParseXML
- getUseCaseName
- getErrorMessage
- epoch2UTC
- alertXML
- _alertBox
- _confirmationBox

The function alertXML can be used for debugging purposes to display the content of the XML message. Depending on the browser capabilities, the data will be serialized.

5.2.2 Smartphone Mobiliser HTML file

The file index.html contains all the HTML5 code. This is the place where you will implement your custom HTML5 code.

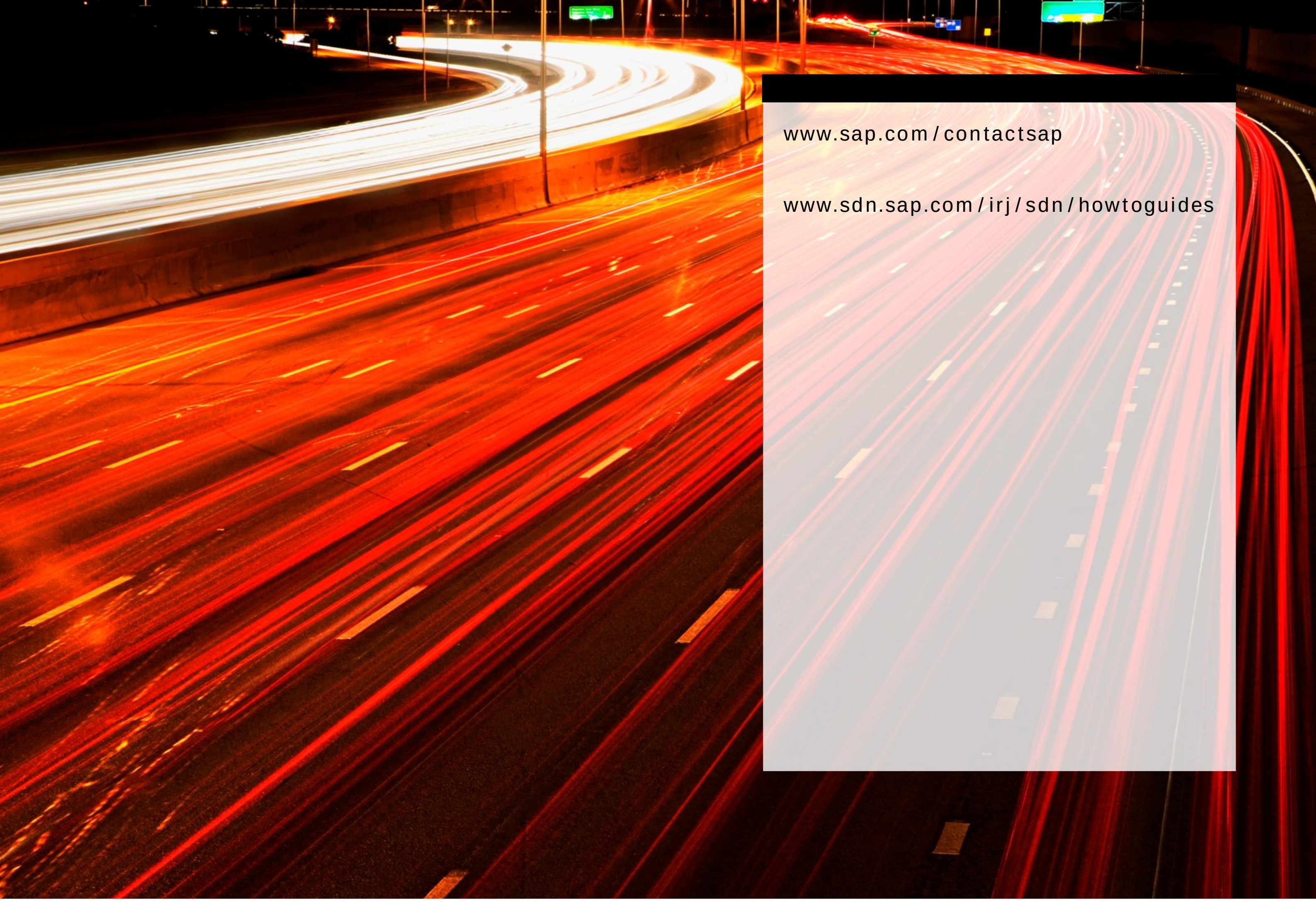
The provided index.html file contains the UI and function calls. The functions are implemented in separate JavaScript files. As a reminder, this app is implementing the client side of the mCommerce services, however the frame of this application could be reused or at least serve as an inspiring template for the B2C app.

5.2.3 Smartphone Mobiliser application JavaScript and CSS files

In the directory 'app' you will find the CSS code and the file app.js. This is the place where you will implement your CSS style definition and your custom JavaScript functions. The Smartphone Mobiliser has separated the logic from the presentation part and moved the JavaScripts out of the HTML file to the file app.js. This would be best practice and for larger apps the scripts would even be split to multiple js files. The app.js is for a large part implementing the logic for the mCommerce functionality. However, some function can be of general use and could be retained for the development of new apps.

List of the generic functions:

- Start application
- Functions regarding orientation handling for iOS devices
- Functions providing separate handling of demo mode. In this mode, the app is running standalone without access to the services on the central Mobiliser server.
- Language handling
- Timer, session timer functions, timeout handling
- PhoneGap integration
- Check connection and connection status
- Small utility functions



www.sap.com/contactsap

www.sdn.sap.com/irj/sdn/howtoguides