Nibiru Mobile 0.2 Reference

December 4, 2012

Part I

Introduction

1 Framework objective

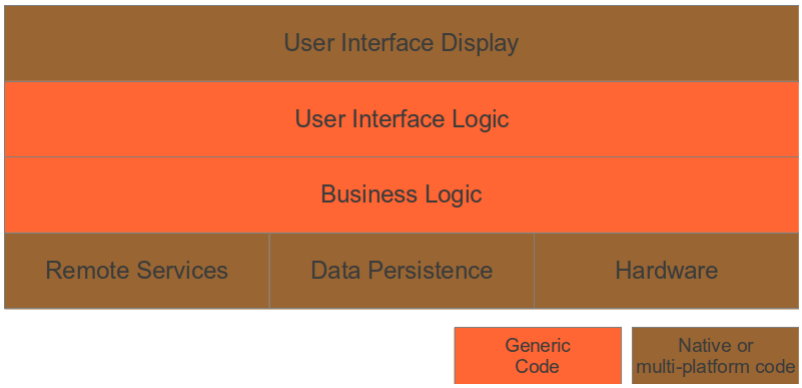
The framework objective is to facilitate building portable mobile applications. The following goals are established in order to meet such objective:

- Portable mobile development, allowing reusing most of the components.
- Support for both, native and HTML5 platforms.
- Unified API for commons components, with different implementations.
- Resuable components for common funcionality, built on top of unified API.
- Common structure and patterns for developing apps.

2 Architecture

This section explains architectural decisions.

2.1 High-level diagram



2.2 Portable development

Nibiru Mobile provides a high degree of portability among differents platforms. It is not focused on developing 100% portable apps. Instead, it aims to make easy cross-platform development reusing most of the components.

Since Java is the base language (see next section), a common API is provided for both, Android and GWT platforms. This way, you can develop both, HTML5 applications (using GWT) and native Android apps.

2.3 Java platform

Javawas chosen because it is currently the most widespread platform within the enterprise applications, in addition to being easily portable to different environments and having many frameworks and libraries.

Most portable development platforms, such as Apache Cordova (aka PhoneGap) or Appcelerator are based on JavaScript. We believe that this language is not suitable for enterprise development, since it lacks of many features such as packaging, strong typing, etc. Adding GWT over PhoneGap can solve these issues. However, Nibiru Mobile goes one step further, providing a common API not just for accessing hardware, but for many common components..

2.4 IoC pattern

In order to decouple each component from the container and other components, the dependencies of each component are injected (IoC pattern).

JSR330, a Java standard for dependency injection, is used for configuring components in a framework-agnostic way. At each platform. specific frameworks such as Guice, RoboGuice or GIN are used in order to implement this pattern.

2.5 MVP pattern

The model used for the presentation layer is the MVP pattern, under its passive view variant. This allows the presenters to be decoupled from each other by an event bus and also to be decoupled from view implementation. Google also makes a good description of this pattern.

Also, the concept of abstracting the view was taken a step further, creating abstractions for common components. Thus, the user can choose creating a generic view or creating a view using the particular advantages of a specific technology.

3 Getting started

3.1 Required software

1. Java (http://www.java.com/en/download/).
2. Eclipse (http://www.eclipse.org/).
3. Maven (http://maven.apache.org/).
4. A GIT client (http://git-scm.com/). We use EGit.
5. Android Plugin (http://developer.android.com/tools/sdk/eclipse-adt.html)
6. GWT Plugin (https://developers.google.com/eclipse/)
7. A Servlet container (such as Tomcat)

3.2 Installation

1. Clone the project as explained in http://poner.un.link
2. Run "mvn eclipse:eclipse" from root directory in order to build the Eclipse project from Maven files and downloading target platform JARS.
3. Import the projects into Eclipse. You must create a M2_REPO classpath variable pointing to the m2/repository directory in your home directory.

3.3 Sample project

1. As in the previous section, import the projects into Eclipse.
2. Android:

(a) Select project properties and mark all the libraries for export (Java Build Path section).

(b) Run the project usign the Android plugin.

3. GWT:

   (a) Compile the project using the GWT plugin.

   (b) Run the project inside a Web server or inside PhoneGap. Running on a Web server+browser, you can test it usgin GWT's development mode (however, you will not have access to mobile hardware).

Part II

Project Structure

4 Main subprojects

The structure for Nibiru Mobile project is arranged in an hierarchical way. In this structure, the main subprojects are the following:

- ar.com.oxen.nibiru.mobile.core
- ar.com.oxen.nibiru.mobile.android
- ar.com.oxen.nibiru.mobile.gwt
- ar.com.oxen.nibiru.mobile.mgwt
- ar.com.oxen.nibiru.mobile.smartgwt
- ar.com.oxen.nibiru.mobile.kendoui

They can be found on a directory called "main"

5 Sample project

A sample application can be found in the ar.com.oxen.nibiru.mobile.sample.app project. It can be found at "sample" directory, along with different platform implementations.

5.1 Typical project structure

Typically, a project will be divided into two or more modules:

- A module containing app-specific components, which are generic and reusable.
- One or more modules containing platform-specific components.

These modules are explained in the following sections. For a deeper undestanding, please look at the sample application.

5.1.1 Application module

The application module usually will depend on ar.com.oxen.nibiru.mobile.core module, in order to access the platform-agnostic API.

In a typical application, this module will contain:

- An ar.com.oxen.nibiru.mobile.core.api.app.EntryPoint implementation, for application startup logic.
- UI logic:
  - All the presenters.
  - A enumeration for custom places.
  - A presenter mapper, for mapping places with presentes.
- Text internationalization resources (properties and an interface).
- Business logic components (which may be divide into API and implementation).
- Data access API (DAOs and domain model).
- Remote services API.

Some packaging issues must be taken into account:

- In order to compile from GWT, you must include:
  - A Module.gwt.xml file, with proper configuraiton for source paths.
  - Java source files.

5.1.2 Platform modules

Platform specific modules will depend on application module and on Nibiru Mobile platform-specific module.

Such modules typically will inlcude:

- Classes for configuring dependency injection.
- View components, in order to implement presenter displays.
- Any platform-specific file or class required for proper application operation.

Android Android modules will depend on application module and on ar.com.oxen.nibiru.mobile.android module.

Android specific components can be set up as follows:

- In the AndroidManifest.xml file declare the ar.com.oxen.nibiru.mobile.android.app.BootstrapActivity as launcher. For each presenter, you must declare a ar.com.oxen.nibiru.mobile.android.ui.mvp.PresenterAc The action name must match {app package name}.place.{place name}.
- Dependency injection is configured using Guice, so you must just write a com.google.inject.AbstractModule subclass for each module. Since RoboGuice is the framework which startups Guice, you must declare a string array resource called "roboguice_modules", which indicates what modules must be used (check the roboguice.xml file in the sample).
- The ar.com.oxen.nibiru.mobile.android.ui.mvp.BaseAndroidView class can be used as the super class for all the view implementations.
- Persistence is managed using OrmLite, so you must create a com.j256.ormlite.android.apptools.OrmLiteSq subclass for database creation, upgrade, etc.
- Jackson JSON serialization is configured using an org.codehaus.jackson.map.ObjectMapper instance. If you need yo customize serialization, you can write a javax.inject.Provider<ObjectMapper> for this class.

GWT GWT modules have more options on the UI layer. Usually, you will choose an UI implementation for your application (however, since they are HTML-based, you can mix UI technologies.

The following modules are provided for different UI technologies support:

- ar.com.oxen.nibiru.mobile.gwt: For generic GWT components and cross technologies (such as GWT-PhoneGap, GWT-Mobile-Persistence, etc.).

- ar.com.oxen.nibiru.mobile.mgwt: MGWT user interface implementation.
- ar.com.oxen.nibiru.mobile.smartgwt: SmartGWT Mobile user interface implementation.
- ar.com.oxen.nibiru.mobile.kendoui: Kendo UI user interface implementation.

Regarding platform-specific components, you will typically need:

- A Module.gwt.xml inhetinting the application module and the platform specific modules.
- A com.google.gwt.core.client.EntryPoint in order to startup the GWT application. This will typically just call the Nibiru Mobile bootstrap.
- Dependency injection is done through GIN. Because of this, you must create a ar.com.oxen.nibiru.mobile.gwt.ioc.GwtInjector subinterface and a com.google.gwt.inject.client.AbstractGinModule sublcass for each module.
- An interface extending both, application message internationalization interface and com.google.gwt.i18n.client.Messages. This way, GWT knows that it must be treated as an i18n interface. Also, you must declare the used languages in the Module.gwt.xml file.

Part III

Modules

6 Core

The ar.com.oxen.nibiru.mobile.core project contains common classes: unified API and generic components. These classes are arranged into different packages, which are explained in the following sections.

6.1 Generic cross components

6.1.1 Application

The ar.com.oxen.nibiru.mobile.core.api.app package contains interfaces related to app setup.

The Bootstrap interface represents the steps which are neccessary in order to start the application on a given platform.

```
package ar.com.oxen.nibiru.mobile.core.api.app;

/**
 * Component for performing platform-specific startup.
 */
public interface Bootstrap {
        /**
         * Callback for performing startup.
         */
        void onBootstrap();
}
```

To put it in another way, there are specific bootstrap for Android, GWT, etc.

On the other side, the EntryPoint interface represents startup logic which is application-specific, but platform independent.

```
package ar.com.oxen.nibiru.mobile.core.api.app;

/**
 * Component for performing application-specific startup.
 */
public interface EntryPoint {
        /**
         * Callback for performing startup.
         */
        void onApplicationStart();
}
```

Typically, there will be a unique entry point for each application. For example, you could have an entry point like this:

```
... imports, etc ...

public class SampleEntryPoint implements EntryPoint {
        private PlaceManager placeManager;

        @Inject
        public SampleEntryPoint(PlaceManager placeManager) {
                super();
                this.placeManager = placeManager;
        }

        @Override
        public void onApplicationStart() {
                this.placeManager.createPlace(DefaultPlaces.LOGIN).go();
        }
}
```

6.1.2 Asynchronous callbacks

Since the frameworks aims to be compatible with HTML5 development, it must take into account asynchronous callback handling. In other platforms (such as Android or iOS), you would simply create threads as needed (for example, when you are going to execute a blocking operation). However, GWT code is translated into JavaScript, which is single-threaded. When a blocking operation is excetued (such as an Ajax call or a WebSQL operation) is executed, a callback must be used.

So, portable code must be written using callbacks. In order to accomplish this, the package ar.com.oxen.nibiru.mobile.core.api.async provides the Callback interface, which aims to unifiy different callbacks used on different libraries.

```
package ar.com.oxen.nibiru.mobile.core.api.async;

/**
 * A callback for any asynchronous call that can result in success or failure.
 *
 * @param <T>
 *            The type returned on success
 */
public interface Callback<T> {

        /**
         * Called when an asynchronous call fails to complete normally.
         *
         * @param reason
         *            failure encountered
         */
        void onFailure(Exception reason);

        /**
         * Called when an asynchronous call completes successfully.
         *
         * @param result
         *            the value returned
         */
        void onSuccess(T result);
}
```

The ar.com.oxen.nibiru.mobile.core.impl.async package provides some utility classes for asynchronous callback handling.

The BaseCallback is a generic base class which implements the exception callback method, simply showing the error to the user with an alert message.

```
package ar.com.oxen.nibiru.mobile.core.impl.async;
```

```
import ar.com.oxen.nibiru.mobile.core.api.async.Callback;
import ar.com.oxen.nibiru.mobile.core.api.ui.AlertManager;

/**
 * Abstract base class for implementing callbacks.
 *
 * @param <T>
 *            The type returned on success
 */
public abstract class BaseCallback<T> implements Callback<T> {
        private AlertManager alertManager;

        public BaseCallback(AlertManager alertManager) {
                super();
                this.alertManager = alertManager;
        }

        @Override
        public void onFailure(Exception error) {
                this.alertManager.showMessage("Error: " + error.getLocalizedMess
        }
}
```

On a layered application, callbacks usually will pass thorug different classes. However, each layer tipically would excetute some logic (otherwise, maybe the layer is poorly designed). Chaining callbacks is a tedious task, so we provide a class for this purpose.

```
package ar.com.oxen.nibiru.mobile.core.impl.async;

import ar.com.oxen.nibiru.mobile.core.api.async.Callback;

/**
 * Class for composing two callbacks.
 *
 * @param <T>
 *            The type returned by the callback
 * @param <C>
 *            The type returned by the chained callback
 */
public abstract class ChainCallback<T, C> implements Callback<T> {
        private Callback<C> chained;

        public ChainCallback(Callback<C> chained) {
                super();
                this.chained = chained;
```

```
        }

        protected Callback<C> getChained() {
                return chained;
        }

        @Override
        public void onFailure(Exception error) {
                this.chained.onFailure(error);
        }
}
```

It chains the exception callback method and provides a method for accessing the chained callback. For example, the following snippet shows how a login method could be implemented:

```
...
private void remoteLogin(final String username,
                final String password,
                Callback<Boolean> callback) {
        this.authenticationService.login(username,
                        password,
                        new ChainCallback<UserDto, Boolean>(callback) {
                @Override
                public void onSuccess(final UserDto userDto) {
                        if (userDto != null) {
                                // update profile, etc...
                                getChained().onSuccess(true);
                        } else {
                                getChained().onSuccess(false);
                        }
                }
        }
}
...
```

6.1.3 Configuration

The ar.com.oxen.nibiru.mobile.core.api.config package contains annotations used for application configuration. These include:

- AppName: The name of the application.
- AppVersion: The application version.
- BaseUrl: Base URL, used for remote service calling.

Such annotations are used when configuring IoC injector for a given application (see example code).

6.1.4 Registration handling

Some components (usually, listeners, handlers, etc.) require some kind of registration. When a component is registered inside another, a good approach is returning an object used for unregistering it in a future. In order to unify such process, the ar.com.oxen.nibiru.mobile.core.api.handler provides the HandlerRegistration interface:

```
package ar.com.oxen.nibiru.mobile.core.api.handler;

/**
 * Component for removing registrations.
 *
 */
public interface HandlerRegistration {
        /**
         * Removes the registered handler.
         */
        void removeHandler();
}
```

6.1.5 Inversion of control

The ar.com.oxen.nibiru.mobile.core.ioc package provides default Guice-based configuration modules. Specific (RoboGuice for Android and GIN for GWT) configuration modules can be found on platform-specific projects.

6.1.6 Common classes

There are interfaces which don't fit well into any of the above categories. They are stored inside ar.com.oxen.nibiru.mobile.core.api.common and ar.com.oxen.nibiru.mobile.core.impl.common packages - until we improve the design :).

The Configurable interface represent anything that can have parameters:

```
package ar.com.oxen.nibiru.mobile.core.api.common;

/**
 * Anything that can be configured.
 *
 * @param <C>
 *             The specific configurable type, for using in method chaining.
```

```
 */
public interface Configurable<C> {
        /**
         * Reads a parameter.
         *
         * @param key
         *              The parameter key
         * @return The parameter value
         */
        <T> T getParameter(String key);

        /**
         * Reads a parameter.
         *
         * @param key
         *              The parameter key
         * @return The parameter value
         */
        <T> T getParameter(Enum<?> key);

        /**
         * Add/sets a parameter.
         *
         * @param key
         *              The parameter key
         * @param The
         *              parameter value
         * @return The same configurable instance, for method chaining.
         */
        C addParameter(String key, Object value);

        /**
         * Add/sets a parameter.
         *
         * @param key
         *              The parameter key
         * @param The
         *              parameter value
         * @return The same configurable instance, for method chaining.
         */
        C addParameter(Enum<?> key, Object value);
}
```

ar.com.oxen.nibiru.mobile.core.impl.common.AbstractConfigurable provides a base implementation for configurables.

Meanwhile, Identifiable represents anything that has an identifier:

```
package ar.com.oxen.nibiru.mobile.core.api.common;

/**
 * Anything that has an identifier.
 *
 * @param <T>
 *            The identifier type.
 */
public interface Identifiable<T> {
        /**
         * @return The identifier
         */
        T getId();
}
```

6.2 Unified API components

6.2.1 Event handling

The ar.com.oxen.nibiru.mobile.core.api.event package provides an unified interface for accessing an EventBus:

```
package ar.com.oxen.nibiru.mobile.core.api.event;

import ar.com.oxen.nibiru.mobile.core.api.handler.HandlerRegistration;

/**
 * An event bus, for implementing a publish-subscribe model.
 *
 */
public interface EventBus {
        /**
         * Creates an event.
         *
         * @param id
         *            The event id
         * @return The event
         */
        Event createEvent(String id);

        /**
         * Creates an event.
         *
         * @param id
         *            The event id
```

```
         * @return The event
         */
        Event createEvent(Enum<?> id);

        /**
         * Adds a handler for listening on an specific event.
         *
         * @param eventId
         *            The event id to be listen
         * @param handler
         *            The handler
         * @return A registration to the event
         */
        HandlerRegistration addHandler(String eventId, EventHandler handler);

        /**
         * Adds a handler for listening on an specific event.
         *
         * @param eventId
         *            The event id to be listen
         * @param handler
         *            The handler
         * @return A registration to the event
         */
        HandlerRegistration addHandler(Enum<?> eventId, EventHandler handler);
}
```

Due to platform limitations, Events are identified by its ID (instead of using the class, as usual). They are represented by this this interface:

```
package ar.com.oxen.nibiru.mobile.core.api.event;

import ar.com.oxen.nibiru.mobile.core.api.common.Configurable;
import ar.com.oxen.nibiru.mobile.core.api.common.Identifiable;

/**
 * An event.
 */
public interface Event extends Identifiable<String>, Configurable<Event> {
        /**
         * Fires the event.
         */
        void fire();
}
```

Also, an Event can have parameters (Configurable interface). Together with EventBus, you can fire Events using DSL-like method chaining:

```
...
eventBus.createEvent("showAlert")
                .addParameter("message", message)
                .fire();
...
```

The EventHandler interface is also standardized:

```
package ar.com.oxen.nibiru.mobile.core.api.event;

/**
 * A handler for listening events.
 */
public interface EventHandler {
        /**
         * Callback method called when the event is fired.
         *
         * @param event
         *              The event
         */
        void onEvent(Event event);
}
```

6.2.2 HTTP requests

The ar.com.oxen.nibiru.mobile.core.api.http package provides interfaces for posting HTTP messages in an unified way.

The HttpManager allows sending this kind of messages:

```
package ar.com.oxen.nibiru.mobile.core.api.http;

import ar.com.oxen.nibiru.mobile.core.api.async.Callback;

/**
 * Manager for performing requests over HTTP.
 *
 */
public interface HttpManager {
        /**
         * Sends a POST reques.
         *
         * @param url
         *              The URL
         * @param callback
         *              A callback for handling the processed response
```

```
         * @param httpCallback
         *             The callback for message processing
         */
        <T> void send(String url, final Callback<T> callback,
                      final HttpCallback<T> httpCallback);
}
```

(currently, only POST is needed). It receives a HttpCallback:

```
package ar.com.oxen.nibiru.mobile.core.api.http;

/**
 * Callback for processing HTTP messages.
 *
 * @param <T>
 *             The expected type for the response after parsing it
 */
public interface HttpCallback<T> {
        /**
         * Builds the request message, usually from some contextual data.
         *
         * @return A string with message body
         */
        String buildRequest();

        /**
         * Parses the response text in order to create an object representing th
         * response.
         *
         * @param responseMessage
         *             The response body
         * @return The object resultin from parsing process
         */
        T parseResponse(String responseMessage);
}
```

which mus supply methods for creating the request data and parsing the HTTP response.

This way, message creation is decoupled from HTTP messaging implementation.

6.2.3 Object serialization

When sending an object across the network, some kind of serialization is needed. For example, you could convert the object into a JSON or a XML stream. The package ar.com.oxen.nibiru.mobile.core.api.serializer provides the Serializer interface, which abstracts such process:

```
package ar.com.oxen.nibiru.mobile.core.api.serializer;

/**
 * Interface for serializing from/to object to/from String.
 */
public interface Serializer {
        /**
         * Converts from object to string.
         *
         * @param object
         *             The object
         * @return The string
         */
        String serialize(Object object);

        /**
         * Converts from string to object
         *
         * @param data
         *             The string
         * @param returnType
         *             The expected return type
         * @return The object
         */
        <T> T deserialize(String data, Class<T> returnType);

        /**
         * @return The encoding (json, xml, etc.)
         */
        String getEncoding();
}
```

6.2.4 Remote services

Remote services are also abstracted. They are represented by the RemoteService interface form ar.com.oxen.nibiru.mobile.core.api.service package:

```
package ar.com.oxen.nibiru.mobile.core.api.service;

import ar.com.oxen.nibiru.mobile.core.api.async.Callback;

/**
 * A remote service.
 */
public interface RemoteService {
```

```
        /**
         * Invokes a method on a remote service
         *
         * @param method
         *            The name of the method
         * @param requestDto
         *            The DTO used for creating request data
         * @param responseClass
         *            The expected response class
         * @param callback
         *            A callback for receiving the response
         */
        <T> void invoke(String method, Object requestDto, Class<T> responseClass
                        Callback<T> callback);
}
```

Remote service implementation is responsible for building the message body. The ar.com.oxen.nibiru.mobile.core.impl.service includes some common implementations:

- JsonRpcService, for JSON-RPC messages.
- RestService, for REST-like messages (a simple POST using the URL for the method name).

Both implementations rely on HttpManager.

6.2.5 User interface

The ar.com.oxen.nibiru.mobile.core.api.ui package includes interfaces for manipulating user interface. At this level, just an AlertManager is provided:

```
package ar.com.oxen.nibiru.mobile.core.api.ui;

/**
 * A manager for showing messages to user.
 */
public interface AlertManager {
        /**
         * Shows an informative message.
         *
         * @param message
         *            The message
         */
        void showMessage(String message);
```

```
        /**
         * Shows an error message.
         *
         * @param exception
         *             The exception which generated the error
         */
        void showException(Exception exception);
}
```

It allows showing messages to the user.

The remaining functionality is included in different subpackages.

Place management Navigation between views is represented through places. The ar.com.oxen.nibiru.mobile.core.api.ui.place package provides interfaces useful for accomplishing this task.

The Place interface abstracts this concept:

```
package ar.com.oxen.nibiru.mobile.core.api.ui.place;

import ar.com.oxen.nibiru.mobile.core.api.common.Configurable;
import ar.com.oxen.nibiru.mobile.core.api.common.Identifiable;

/**
 * A place.
 */
public interface Place extends Identifiable<String>, Configurable<Place> {
        /**
         * Navigates to a place (no push).
         */
        void go();

        /**
         * Navigates to a place.
         *
         * @param push
         *             True if previous place must be kept into the stack (allows
         *             returning later to the current place using
         *             {@link PlaceManager#back()}).
         */
        void go(boolean push);
}
```

Places are created using a PlaceManager instance:

```
package ar.com.oxen.nibiru.mobile.core.api.ui.place;
```

```
/**
 * A manager for handling places.
 */
public interface PlaceManager {
        /**
         * Creates a place.
         *
         * @param id
         *             The place id
         * @return The place
         */
        Place createPlace(String id);

        /**
         * Creates a place.
         *
         * @param id
         *             The place id
         * @return The place
         */
        Place createPlace(Enum<?> id);

        /**
         * Backs to previous place.
         */
        void back();
}
```

Together, they allows navigating using a DSL-like syntax, using method chaining. For example:

```
...
placeManager.createPlace(DefaultPlaces.HOME)
                        .addParameter("message", message)
                        .go(false);
...
```

As seen on the example, there is an enumeration with identifiers for common places:

```
package ar.com.oxen.nibiru.mobile.core.api.ui.place;

/**
 * Places commonly used in applications.
 */
public enum DefaultPlaces {
```

```
        INITIAL, HOME, LOGIN
}
```

Model-View-Presenter pattern The ar.com.oxen.nibiru.mobile.core.api.ui.mvp package provides interfaces for abstracting different implementations for MVP pattern.

Under this pattern, the UI logic is contained into a Presenter:

```
package ar.com.oxen.nibiru.mobile.core.api.ui.mvp;

import ar.com.oxen.nibiru.mobile.core.api.ui.place.Place;

/**
 * A presenter.
 *
 * @param <V>
 *            The view type
 */
public interface Presenter<V extends View> {
        /**
         * @return The associated view
         */
        V getView();

        /**
         * Startup method called when the presenter is initialized.
         *
         * @param place
         *            The calling place
         */
        void go(Place place);

        /**
         * Callback method called when leaving the presenter.
         */
        void onStop();
}
```

which holds a reference to a View:

```
package ar.com.oxen.nibiru.mobile.core.api.ui.mvp;

/**
 * A view.
 */
```

```
public interface View {
        /**
         * @return The underlying native implementation
         */
        Object asNative();
}
```

The view shouldn't contain logic. Its responsibilities should be limited to showing data (including internationalization) and firing events. This way, views can be easily replaced when changing the platform. And presenter keeps unchanged. This approach also allows taking advantage of native view capabilities.

When navigating to a specific place, a presenter for handling such place must be selected. This is accomplished by implementing a PresenterMapper:

```
package ar.com.oxen.nibiru.mobile.core.api.ui.mvp;

/**
 * A component for getting the presenter associated to a given place.
 */
public interface PresenterMapper {
        /**
         * Gets a presenter.
         *
         * @param place
         *              The associated place
         * @return The presenter
         * @throws NoPresenterFoundException
         *              If no presenter is mapped for such place
         */
        Presenter<?> getPresenter(String place) throws NoPresenterFoundException
}
```

The package also provides abstractions for accessing widget data. The TakesValue interface allows accessing data from a widget (which must be adapted according the UI framework):

```
package ar.com.oxen.nibiru.mobile.core.api.ui.mvp;

/**
 * Something that can holds a value.
 *
 * @param <T>
 *              The value type
 */
public interface TakesValue<T> {
        /**
         * @param value
```

```
         *              The value
         */
        void setValue(T value);

        /**
         * @return The value
         */
        T getValue();
}
```

While the HasChangeHandler represents a widget which can fire value change events.

```
package ar.com.oxen.nibiru.mobile.core.api.ui.mvp;

import ar.com.oxen.nibiru.mobile.core.api.handler.HandlerRegistration;

/**
 * Something that can notify change events.
 */
public interface HasChangeHandler {
        /**
         * Sets the change handler.
         *
         * @param changeHandler
         *              The change handler
         * @return A handler registration
         */
        HandlerRegistration setChangeHandler(ChangeHandler changeHandler);
}
```

Since many widgets can do both things (holding a value and firing change events), the HasValue interface combines both:

```
package ar.com.oxen.nibiru.mobile.core.api.ui.mvp;

/**
 * Something that is both, {@link TakesValue} and {@link HasChangeHandler}
 *
 * @param <T>
 *              The value type
 */
public interface HasValue<T> extends TakesValue<T>, HasChangeHandler {
}
```

In a similar way, widget having click event (such as buttons) are abstracted by HasClickHandler interface:

```
package ar.com.oxen.nibiru.mobile.core.api.ui.mvp;

import ar.com.oxen.nibiru.mobile.core.api.handler.HandlerRegistration;

/**
 * Something that can be clicked.
 */
public interface HasClickHandler {
        /**
         * Sets the click handler.
         *
         * @param clickHandler
         *             The click handler
         * @return A handler registration
         */
        HandlerRegistration setClickHandler(ClickHandler clickHandler);
}
```

The ar.com.oxen.nibiru.mobile.core.impl.mvp package provides base implementations for both, PresenterMapper and Presenter.

The BasePresenterMapper includes an inner class, Cbk, which allows an easy creation of callbacks. It injects the AlertManager automatically into the callback.

```
package ar.com.oxen.nibiru.mobile.core.impl.mvp;

import ar.com.oxen.nibiru.mobile.core.api.ui.AlertManager;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.Presenter;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.View;
import ar.com.oxen.nibiru.mobile.core.impl.async.BaseCallback;

/**
 * Base class for presenters
 *
 * @param <V>
 *            The view type
 */
abstract public class BasePresenter<V extends View> implements Presenter<V> {
        private V view;
        private AlertManager alertManager;

        public BasePresenter(V view, AlertManager alertManager) {
                super();
                this.view = view;
                this.alertManager = alertManager;
        }
```

```
        @Override
        public V getView() {
                return this.view;
        }

        @Override
        public void onStop() {
        }

        protected AlertManager getAlertManager() {
                return alertManager;
        }

        /**
         * Utility class for creating internal callbacks.
         *
         * @param <T>
         *              The callback return type
         */
        protected abstract class Cbk<T> extends BaseCallback<T> {
                public Cbk() {
                        super(alertManager);
                }
        }
}
```

Inside a presenter which extends BasePresenter, you can simply could run something like this:

```
...
geolocationManager.watchPosition(new Cbk<Position>() {
        @Override
        public void onSuccess(Position result) {
                //...
        }
});
...
```

6.2.6 User preferences

A simple API for storing user preferences can be found at ar.com.oxen.nibiru.mobile.core.api.preferences package.

The Preferences interface allows accessing the preferences. It extends Configurable, so preference data can be loaded or stored using its methods.

```
package ar.com.oxen.nibiru.mobile.core.api.preferences;

import ar.com.oxen.nibiru.mobile.core.api.common.Configurable;

/**
 * User preferences.
 */
public interface Preferences extends Configurable<Preferences> {

}
```

The ar.com.oxen.nibiru.mobile.core.impl.preferences package provides a base class for implementing preferences (AbstractPreferences) which provides useful functionality such as data conversion.

6.2.7 Geolocation

The ar.com.oxen.nibiru.mobile.core.api.geolocation provides accessing to location hardware. The main interface is GeolocationManager:

```
package ar.com.oxen.nibiru.mobile.core.api.geolocation;

import ar.com.oxen.nibiru.mobile.core.api.async.Callback;

/**
 * Manager for accessing geolocation information.
 */
public interface GeolocationManager {
        /**
         * Watches position changes.
         *
         * @param callback
         *             A callback for notifying positon changes
         */
        void watchPosition(Callback<Position> callback);
}
```

Location information is accessed using Position and Coordinates interfaces:

```
package ar.com.oxen.nibiru.mobile.core.api.geolocation;

/**
 * A location read at some moment.
 */
public interface Position {
        /**
```

```
         * @return The coordinates
         */
        public Coordinates getCoordinates();

        /**
         * @return The timestamp
         */
        public long getTimeStamp();
}
```

```
package ar.com.oxen.nibiru.mobile.core.api.geolocation;

/**
 * Location coordinates.
 */
public interface Coordinates {
        /**
         * @return The latitude
         */
        double getLatitude();

        /**
         * @return The longitude
         */
        double getLongitude();

        /**
         * @return The altitude
         */
        double getAltitude();

        /**
         * @return The accuracy
         */
        double getAccuracy();

        double getSpeed();
}
```

6.3 Generic functionality components

6.3.1 Security

The security module provides authentication and profile access functionality. In a future it should include authorization too.

Security components operate at many layers. They are explained in the following sections.

User Interface The package ar.com.oxen.nibiru.mobile.core.impl.ui.security provides a presenter for performing login:

```
package ar.com.oxen.nibiru.mobile.core.impl.ui.security;

import javax.inject.Inject;

import ar.com.oxen.nibiru.mobile.core.api.business.security.AuthenticationManage
import ar.com.oxen.nibiru.mobile.core.api.ui.AlertManager;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.ClickHandler;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.HasClickHandler;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.View;
import ar.com.oxen.nibiru.mobile.core.api.ui.place.DefaultPlaces;
import ar.com.oxen.nibiru.mobile.core.api.ui.place.Place;
import ar.com.oxen.nibiru.mobile.core.api.ui.place.PlaceManager;
import ar.com.oxen.nibiru.mobile.core.impl.mvp.BasePresenter;
import ar.com.oxen.nibiru.mobile.core.impl.ui.security.LoginPresenter.Display;

/**
 * Presenter for login screen.
 */
public class LoginPresenter extends BasePresenter<Display> {
        public interface Display extends View {
                String getUsername();

                String getPassword();

                HasClickHandler getLogin();

                void showLoginError();
        }

        private AuthenticationManager authenticationManager;
        private PlaceManager placeManager;

        @Inject
        public LoginPresenter(Display display, AlertManager alertManager,
                        AuthenticationManager authenticationManager,
                        PlaceManager placeManager) {
                super(display, alertManager);
                this.authenticationManager = authenticationManager;
                this.placeManager = placeManager;
        }
```

```
        @Override
        public void go(Place place) {
                getView().getLogin().setClickHandler(new ClickHandler() {

                        @Override
                        public void onClick() {
                                authenticationManager.login(getView().getUsernam
                                                .getPassword(), new Cbk<Boolean>
                                        @Override
                                        public void onSuccess(Boolean result) {
                                                if (result) {
                                                        placeManager.createPlace
                                                } else {
                                                        getView().showLoginError
                                                }
                                        }
                                });
                        }
                });
        }
}
```

Business Logic This is the core of security functionality. Its API can be found at ar.com.oxen.nibiru.mobile.core.api.business.security package.

The AuthenticationManager interface alloes authenticating an user:

```
package ar.com.oxen.nibiru.mobile.core.api.business.security;

import ar.com.oxen.nibiru.mobile.core.api.async.Callback;

/**
 * Manager for performing authentication operations.
 */
public interface AuthenticationManager {
        /**
         * Performs a login.
         *
         * @param username
         *              The username
         * @param password
         *              The password
         * @param callback
         *              A callback notifying true if the login was successful
         */
```

```
        void login(String username, String password, Callback<Boolean> callback)

        /**
         * Performs a logout.
         *
         * @param callback
         *            A callback notifying true if the login was successful
         */
        void logout(Callback<Boolean> callback);
}
```

Once the user is logged in, its information can be accessed through the Profile interface:

```
package ar.com.oxen.nibiru.mobile.core.api.business.security;

/**
 * A user profile.
 */
public interface Profile {
        /**
         * @return True if the provile is valid (authen ticated)
         */
        boolean isActive();

        /**
         * @return The username
         */
        String getUsername();

        /**
         * @return The first name
         */
        String getFirstName();

        /**
         * @return The last name
         */
        String getLastName();
}
```

Just inject a Profile into your component.

Finally, the HashManager enables hashing sensitive data, such as passwords.

```
package ar.com.oxen.nibiru.mobile.core.api.business.security;

/**
```

```
 * A manager for hashing operations.
 */
public interface HashManager {
        /**
         * Calculates a hash.
         *
         * @param data
         *              The data
         * @return The hash
         */
        String hash(String data);
}
```

The ar.com.oxen.nibiru.mobile.core.impl.business.security package provides generic implementations of these interfaces.

Persistence In order to authenticate the user when the device is offline, it is required to store authentication data locally.

The ar.com.oxen.nibiru.mobile.core.api.data.security package provides an API for storing security data.

The UserDao allows loading and storing user data using the DAO pattern:

```
package ar.com.oxen.nibiru.mobile.core.api.data.security;

import ar.com.oxen.nibiru.mobile.core.api.async.Callback;

/**
 * A DAO for users.
 */
public interface UserDao {
        /**
         * Loads an user.
         *
         * @param username
         *              The username to look for
         * @param callback
         *              A callback returning the user domain object
         */
        void findByName(String username, Callback<User> callback);

        /**
         * Creates a user
         *
         * @param username
         *              The user name
```

```
         * @param passwordHash
         *             The password hash
         * @param firstName
         *             First name
         * @param lastName
         *             Last name
         * @return The user domain object
         */
        User create(String username, String passwordHash, String firstName,
                    String lastName);

        /**
         * Deletes all the users.
         *
         * @param callback
         *             A callback
         */
        void deleteAll(Callback<Void> callback);
}
```

The user information is provided by the User interface:

```
package ar.com.oxen.nibiru.mobile.core.api.data.security;

/**
 * User domain object.
 */
public interface User {
        String getUsername();

        String getPasswordHash();

        String getFirstName();

        String getLastName();

        void setUsername(String username);

        void setPasswordHash(String passwordHash);

        void setFirstName(String firstName);

        void setLastName(String lastName);
}
```

Remote Services When possible, the device would perform the authentication against a remote service (on line). The ar.com.oxen.nibiru.mobile.core.api.service.security package provides such functionality.

Such remote authentication is done using the AuthenticationService

```
package ar.com.oxen.nibiru.mobile.core.api.service.security;

import ar.com.oxen.nibiru.mobile.core.api.async.Callback;

/**
 * A remote authentication service.
 */
public interface AuthenticationService {
        /**
         * Performs a remote login.
         *
         * @param username
         *              The username
         * @param password
         *              The password
         * @param callback
         *              A callback returning the user DTO
         */
        void login(String username, String password, Callback<UserDto> callback)
}
```

It transfers information, as usual, using DTOs.

```
package ar.com.oxen.nibiru.mobile.core.api.service.security;

/**
 * A DTO for transferring login request data.
 */
public interface LoginDto {
        String getUsername();

        String getPassword();

        void setUsername(String username);

        void setPassword(String password);
}
```

```
package ar.com.oxen.nibiru.mobile.core.api.service.security;

/**
 * A DTO for transferring user data.
```

```
 */
public interface UserDto {
        String getFirstName();

        String getLastName();

        void setFirstName(String firstName);

        void setLastName(String lastName);
}
```

The ar.com.oxen.nibiru.mobile.core.impl.service.security package provides a generic implementation for AuthenticationService and an annotation for configuring which remote service is the one used for authentication.

7 Android

7.1 Generic cross components

7.1.1 Application

No platform-specific bootstrap is required, so the only action made by ar.com.oxen.nibiru.mobile.android.app.Ar is calling the application entry point.

Application startup is performed by ar.com.oxen.nibiru.mobile.android.app.BootstrapActivity. You must configure this activity in the AndroidManifest.xml file as launcher activity.

7.1.2 Inversion of control

Dependency injection is based on RoboGuice. It is based on Guice and uses Guice standard modules for configuration.

This framework requires creating a string array resource in order to define the modules to be used. For example:

```
<resources>
        <string-array name="roboguice_modules">
                <item>ar.com.oxen.nibiru.mobile.android.ioc.DefaultAndroidModule
                <item>ar.com.oxen.nibiru.mobile.android.ioc.DefaultAndroidHardwa
                <item>ar.com.oxen.nibiru.mobile.core.ioc.DefaultSecurityModule</
                <item>ar.com.oxen.nibiru.mobile.android.ioc.DefaultSecurityModul
                <item>ar.com.oxen.nibiru.mobile.sample.android.Module</item>
        </string-array>
</resources>
```

An advantage of this approach is that such module list can be customized using Android resource selection mechanisms.

The ar.com.oxen.nibiru.mobile.android.ioc package contains many Guice modules for default configurations. It also contains generic providers.

7.2 Unified API components

7.2.1 Event handling

Event bus listening is implemented using android.content.BroadcastReceiver .
Events are thrown using android.content.Intent instances. The ar.com.oxen.nibiru.mobile.android.event package contains such implementations.

7.2.2 HTTP requests

HTTP requests are implemented using Apache HTTP Components, which are included in Android platform. Such implementation can be found on ar.com.oxen.nibiru.mobile.android.http package.

7.2.3 Object serialization

JSON serialization is provided using Jackson processor. The ar.com.oxen.nibiru.mobile.android.serializer package contains this implementation. JSON serialization is configured using an org.codehaus.jackson.map.ObjectMapper instance. If you need yo customize serialization, you can write a javax.inject.Provider<ObjectMapper> for this class.

7.2.4 User interface

User interface is divided into many packages, just like at the core module. The ar.com.oxen.nibiru.mobile.android.ui contains a android.widget.Toast based implementation for ar.com.oxen.nibiru.mobile.core.api.ui.AlertManager.

i18n The ar.com.oxen.nibiru.mobile.android.ui.i18n contains classes used for internationalization.

The ar.com.oxen.nibiru.mobile.android.ui.i18n.MessageInvocationHandler class is used, in conjuntion with a Java proxy, in order to read messages from a resource bundle according to method name. This way, interfaces for i18n messages can be used on both, GWT and Android. This approach unifies both models.

Place management Places are handled using android.content.Intent instances. Inside the ar.com.oxen.nibiru.mobile.android.ui.place package, the IntentPlace wraps an intent inside a Nibiru Mobile place, while the IntentPlaceManager implements a place manager which IntentPlace instances.

IntentPlace just fires an intent. The intent action is build by convention. Its structure is:

{application_package_name}.place.{place_name}

You must follow such convention when configurin AndroidManifest.xml file.

Model-View-Presenter pattern The ar.com.oxen.nibiru.mobile.android.ui.mvp contains different classes for implementing the MVP pattern. Most of them are just adapters. We will overlook them in order to focus on more important classes.

The main class is PresenterActivity:

```
package ar.com.oxen.nibiru.mobile.android.ui.mvp;

import javax.inject.Inject;

import roboguice.activity.RoboActivity;
import roboguice.inject.ContextScope;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ContextMenu.ContextMenuInfo;
import ar.com.oxen.nibiru.mobile.android.ui.place.IntentPlace;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.Presenter;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.PresenterMapper;
import ar.com.oxen.nibiru.mobile.core.api.ui.place.Place;

/**
 * An activity that delegates logic to a presenter.
 */
public class PresenterActivity extends RoboActivity {
        @Inject
        private PresenterMapper presenterMapper;

        @Inject
        protected ContextScope scope;

        private Presenter<?> presenter;
```

```
private AndroidView view;

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Place place = new IntentPlace(this.getIntent(), this);

        synchronized (ContextScope.class) {
                scope.enter(this);
                try {
                        this.presenter = this.presenterMapper.getPresent
                                        .getId());
                } finally {
                        scope.exit(this);
                }
        }

        this.view = (AndroidView) this.presenter.getView();
        this.setContentView(this.view.asNative());
        this.presenter.go(place);
}

@Override
protected void onStop() {
        super.onStop();
        this.presenter.onStop();
}

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
        return this.view.onPrepareOptionsMenu(menu);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
        return this.view.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
        return this.view.onOptionsItemSelected(item);
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
```

```
                        ContextMenuInfo menuInfo) {
                this.view.onCreateContextMenu(menu, v, menuInfo);
        }

        @Override
        public boolean onContextItemSelected(MenuItem item) {
                return this.view.onContextItemSelected(item);
        }
}
```

which delegates on the presenter. In order to get the presenter responsible for executing the logic, it asks to the presenter mapper using the place as parameter. This class also shows the view.

For each presenter, you must add an activity of this type, following the conventions explained in the previous section.

Regarding the view, there is a more specific interface (AndroidView), which allows handling Android events:

```
package ar.com.oxen.nibiru.mobile.android.ui.mvp;

import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuItem;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.View;

/**
 * {@link View} specialization that adds Android events.
 */
public interface AndroidView extends View {
        android.view.View asNative();

        boolean onPrepareOptionsMenu(Menu menu);

        boolean onCreateOptionsMenu(Menu menu);

        boolean onOptionsItemSelected(MenuItem item);

        void onCreateContextMenu(ContextMenu menu, android.view.View v,
                        ContextMenuInfo menuInfo);

        boolean onContextItemSelected(MenuItem item);
}
```

BaseAndroidView provides an empty implementation for this interface:

```
package ar.com.oxen.nibiru.mobile.android.ui.mvp;

import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

/**
 * Base class for Android based views.
 */
public abstract class BaseAndroidView implements AndroidView {
        @Override
        public boolean onPrepareOptionsMenu(Menu menu) {
                return false;
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                return false;
        }

        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
                return false;
        }

        @Override
        public void onCreateContextMenu(ContextMenu menu, View v,
                        ContextMenuInfo menuInfo) {
        }

        @Override
        public boolean onContextItemSelected(MenuItem item) {
                return false;
        }
}
```

7.2.5 User preferences

User preferences are stored using android.content.SharedPreferences. The implementation can be found at the ar.com.oxen.nibiru.mobile.android.preferences package.

7.2.6 Geolocation

The ar.com.oxen.nibiru.mobile.android.geolocation package contains a native location service implementation.

8 GWT

8.1 Generic cross components

8.1.1 Application

Due to different options (frameworks, deployment environments, etc.), application setup is a little more complicated on GWT.

For example, some frameworks, such as MGWT, are based on GWT activities and places. So, you may choose between using raw GWT activities and places or building you UI using MGWT. The only difference is how the root application widget is created. In order to abstract this, Nibiru Mobile provides the AppWidgetBootstrap interface:

```
package ar.com.oxen.nibiru.mobile.gwt.app;

import com.google.gwt.user.client.ui.IsWidget;

/**
 * Annotation for application root widget. The provider must return the app
 * widget with its corresponding Activity Manager configured.
 *
 */
public interface AppWidgetBootstrap {
        IsWidget createAppWidget();
}
```

The GwtPlacesBootstrap class receives an AppWidgetBootstrap and performs GWT places configuration. GwtAppWidgetBootstrap provides an AppWidgetBootstrap implementation that uses standard GWT widgets. On the ar.com.oxen.nibiru.mobile.mgwt module, the ar.com.oxen.nibiru.mobile.mgwt.app.MgwtAppWidgetBootstrap class provides a MGWT-based widget for place navigation.

The GwtPlacesBootstrap also receives a DatabaseBootstrap instance:

```
package ar.com.oxen.nibiru.mobile.gwt.app;

import ar.com.oxen.nibiru.mobile.core.api.async.Callback;

/**
```

```
 * Interface representing database creation process.
 */
public interface DatabaseBootstrap {
        /**
         * Creates the database.
         *
         * @param callback
         *             A callback notifying the process end
         */
        void createDatabase(Callback<Void> callback);
}
```

which creates the Web SQL database. However, your applicacion may not need/support a local database. Because of this, there is a dummy implementation, explained in the next section.

SmartGWT mobilehas a different approach. It is not based on GWT activities and places, so it has its own bootstrap: ar.com.oxen.nibiru.mobile.smartgwt.app.SmartGwtBootstrap. However, it still receives a DatabaseBootstrap in order to initialize the database.

Kendo UIperforms navigation between views defined in the same HTML page (you can navigate to external pages, but not to dynamically generated views on the same page). The Kendo UI module (still experimental) provides bootstrap classes which creates all the views on the startup and adds them to the host page. They are placed on ar.com.oxen.nibiru.mobile.kendoui.app package.

8.1.2 Data access

The ar.com.oxen.nibiru.mobile.gwt.data package contains components for accessing Web SQL database. The technologies used for this purpose are GwtMobilePersistence and persistence.js. The GwtMobileDatabaseBootstrap sets up a database using these technologies.

As explained in the previous section, if you aren't going to support Web SQL database (for example if you are targetting browsers that doesn't support this feature), you can use DummyDatabaseBootstrap.

8.1.3 Registration handling

Since GWT API provides a com.google.web.bindery.event.shared.HandlerRegistration class for un-registering handlers, the framework provides a ar.com.oxen.nibiru.mobile.gwt.handler.HandlerRegis which adapts to the Nibiru Mobile API.

8.1.4 Inversion of control

The ar.com.oxen.nibiru.mobile.gwt.ioc contains many GIN modules useful for default configurations. Also, it provides a base interface for creating injectors

based on GIN (GwtInjector).

On ar.com.oxen.nibiru.mobile.mgwt module, the ar.com.oxen.nibiru.mobile.mgwt.ioc package provides GIN modules for configuring dependency injection with MGWT. Similary, the ar.com.oxen.nibiru.mobile.smartgwt module contains such modules in the ar.com.oxen.nibiru.mobile.smartgwt.ioc package. The same applies for ar.com.oxen.nibiru.mobile.kendoui with the ar.com.oxen.nibiru.mobile.kendoui.ioc package.

8.2 Unified API components

8.2.1 Event handling

GWT event handling is implemented using a wrapper for com.google.web.bindery.event.shared.EventBus. This adapter and simple event/event handler implementations can be found at the ar.com.oxen.nibiru.mobile.gwt.event package.

8.2.2 HTTP requests

The com.google.gwt.http.client.RequestBuilder class is used in order to perform HTTP request. The adapter to Nibiru Mobile API is located at the ar.com.oxen.nibiru.mobile.gwt.http package. Its name is RequestBuilderHttpManager.

8.2.3 Object serialization

The package ar.com.oxen.nibiru.mobile.gwt.serializer contains GWT implementation for JSON serialization: AutoBeanSerializer. It is based in AutoBeans technology, included in GWT.

8.2.4 User interface

The ar.com.oxen.nibiru.mobile.gwt.ui package contains an alert manager implementation (GwtAlertManager) which uses Window.alert() in order to show messages.

The ar.com.oxen.nibiru.mobile.mgwt.ui from ar.com.oxen.nibiru.mobile.mgwt module provides a similar implementation, but using MGWT com.googlecode.mgwt.ui.client.dialog.Dialogs.

SmartGWT mobile has a non-working implementation, so it currently uses Window.alert(). It can be found at ar.com.oxen.nibiru.mobile.smartgwt.ui.mvp package.

Place management The ar.com.oxen.nibiru.mobile.gwt.ui.place package contains classes that adapt GWT's activities and places to Nibiru Mobile place management API. GwtPlaceManager is a place manager implementation that delegates on com.google.gwt.place.shared.PlaceController for navigation. It creates SimplePlace instances, which just wraps a GWT place.

DefaultActivityMapper and DefaultPlaceHistoryMapper provide generic implementations for both, ActivityMapper and PlaceHistoryMapper.

The ar.com.oxen.nibiru.mobile.mgwt module provides the ar.com.oxen.nibiru.mobile.mgwt.ui.place.DefaultAnim which just selects a random animation for place transition.

On the other hand, SmartGWT Mobile doesn't use activities and places. The ar.com.oxen.nibiru.mobile.smartgwt.ui.place package provides com.smartgwt.mobile.client.widgets.layout.NavS based navigation implementations (place and place manager).

Kendo UI support for place management can be found at ar.com.oxen.nibiru.mobile.kendoui.ui.place package.

Model-View-Presenter pattern The ar.com.oxen.nibiru.mobile.gwt.ui.mvp package contains classes (API implementations and adapteres) for implementing the MVP pattern.

The main class is PresenterActivity:

```
package ar.com.oxen.nibiru.mobile.gwt.ui.mvp;

import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.Presenter;
import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.View;
import ar.com.oxen.nibiru.mobile.core.api.ui.place.Place;

import com.google.gwt.activity.shared.AbstractActivity;
import com.google.gwt.event.shared.EventBus;
import com.google.gwt.user.client.ui.AcceptsOneWidget;
import com.google.gwt.user.client.ui.IsWidget;

/**
 * Presenter-based activity.
 */
public class PresenterActivity extends AbstractActivity {
        private Presenter<? extends View> presenter;
        private Place place;

        public PresenterActivity(Presenter<? extends View> presenter, Place plac
                super();
                this.presenter = presenter;
                this.place = place;
        }
```

```
        @Override
        public void start(AcceptsOneWidget containerWidget, EventBus eventBus) {
                IsWidget widget = (IsWidget) this.presenter.getView().asNative()
                containerWidget.setWidget(widget);
                this.presenter.go(this.place);
        }

        @Override
        public void onStop() {
                super.onStop();
                this.presenter.onStop();
        }
}
```

which delegates on the presenter. This activity receives the presenter responsible for executing the logic and the place from DefaultActivityMapper. This class also sets the display widget.

Regarding the view, a base class is provided:

```
package ar.com.oxen.nibiru.mobile.gwt.ui.mvp;

import ar.com.oxen.nibiru.mobile.core.api.ui.mvp.View;

import com.google.gwt.user.client.ui.Composite;

/**
 * Base class for GWT views.
 */
public abstract class BaseGwtView extends Composite implements View {

        @Override
        public Composite asNative() {
                return this;
        }

}
```

On the ar.com.oxen.nibiru.mobile.mgwt module, you can find MGWT adapters and base classes for MVP pattern inside the ar.com.oxen.nibiru.mobile.mgwt.ui.mvp package. In a similar way, the ar.com.oxen.nibiru.mobile.smartgwt.ui.mvp package from ar.com.oxen.nibiru.mobile.smartgwt module provides SmartGWT Mobile implementations for the same pattern.

Kendo UI support for MVP pattern can be found at ar.com.oxen.nibiru.mobile.kendoui.ui.mvp package.

8.2.5 User preferences

The ar.com.oxen.nibiru.mobile.gwt.preferences package contains a cookie-based preferences services (CookiesPreferences). In a future, a Web SQL preferences could be implemented.

8.2.6 Geolocation

Geolocation (as hardware access in general) is implemented using Apache Cordova (aka PhoneGap). Since this API is JavaScript-based, GWT-PhoneGap is used in order to access it from GWT.

Geolocation implementations are found in the ar.com.oxen.nibiru.mobile.gwt.geolocation package.

Part IV
Deployment

9 HTML

9.1 Standard Web server

If you aren't targetting mobile development (event when Nibiru Mobile has "mobile" in its name!), you can deploy it on an standard web server and run the app from a desktop browser.

Just compile and deploy it as you would do with any GWT application.We recommend you reading the GWT documentation regarding this aspect.

If you are targetting mobile development, you can even use a Web server + desktop browser in order to test quickly your application. You can even use GWT development mode in order to avoid full compilation when developing the application. Again, we recommend you reading GWT documentation related to this point. However, you can't test hardware-related fucntionality under a desktop browser (maybe geolocation is the exception, which is supported by some browsers).

9.2 Apache Cordova

In order to deploy to a mobile device (or a simulator), you must follow PhoneGap instructions for your platform. Typically, you will need creating a project for each platform and copying the GWT generated files inside each one.

9.3 Kendo UI considerations

Kendo UI Mobile has a commercial license. There is not OpenSource license. So, it can be included in Nibiru Mobile. When deploying an application built with this framework, you must include your Kendo UI Mobile licensed copy inside a web app directory and load them from HTML host page. For example:

```
<html>
        <head>
                ...
                <script src="nibiru_mobile_sample/js/jquery.min.js"></script>
                <script src="nibiru_mobile_sample/js/kendo.mobile.min.js"></scri
                <link href="nibiru_mobile_sample/styles/kendo.common.min.css" re
                <link href="nibiru_mobile_sample/styles/kendo.mobile.all.min.css
                <script type="text/javascript" src="nibiru_mobile_sample/nibiru_
        </head>
        <body>
                ...
        </body>
</html>
```

You can copy the files to the webapp manually or include them into a custom-made GWT module (inside a "public" directory). Using the second option, when including such module, the files will be copied automatically.

10 Android

Deploying as native Android application is quite simple. You must just create an Android application linked with Nibiru Mobile dependencies. You can even configure Maven in order to creating the Eclipse project configuration for Android (such as it is done on the example project).

The only issue is that Android Eclipse plugin requires linked libraries to be exported (otherwise, they aren't included when generating APK). In order to fix this, select the project -> Properties -> Java Build Path -> Order and Export and check all the dependencies.

Part V

License

The framework is distributed under Apache 2.0 license.