TagBy v2 SDK

This document provides information about TagBy v2 SDK components for and Android application.

Documents

The core concept around TagBy SDK is a document which represents an offer configured using TagBy web manager. Before starting to develop any TagBy application you must decide which part of the application is configurable using web manager's designer and which is fixed inside the application. The document consist of configurable items which are further used by the application to apply the offer to the application UI.

For most of the applications, you need to customise the UI items which are called widgets, like buttons, images, cameras and so on. These are described in the section **Widgets**.

Sometimes the application requires settings configurable via the TagBy web manager and these settings affect the application UI or behaviour but are not directly placed on the designer. This includes for instance any kind of popup messages or winning chance parameters for a reward application. These are placed in the section called **Settings**.

In your application you define the structure of your document by inheriting from **TagByDocument** class. The most common document properties are referenced from the instances of this class. And this is the place where you should put your custom document properties.

Under the hood the document properties are always stored as JSON objects. You should get familiar with the **JSONObject** API from the Android documentation.

For most cases you don't have to deal explicitly with anything but widgets and settings from your document. The basic functionality is delivered by **com.tagby.sdk.simpleapp** Android library, which let you focus

Offer

Every document provides basic information about the offer represented by the document. You can reference the offer from here:

TagByOffer TagByDocument.getOffer()

These are the members of the **TagByOffer** class:
- **String getId()** - gets the unique offer identifier
- **String getPreviewUrl()** - gets the offer preview image url, this is presented by the application of the offer selection screen

Settings

Additionally the document contains various settings. These are split into two categories:
- social networks settings: you can find the information about this in the **Settings** section
- other basic settings: the information about these is below.

In order to reference basic document settings use:

TagBySettings TagByDocument.getSettings()

There are not too many items in the settings:
- TagByOrientation getOrientation()

- **boolean blockScreenLocking()**: determines id the device is allowed to go into sleep mode when the application is running. Most of the times you would like to block the sleep.
- **boolean nfcEnabled()**: determines how the user can identify yourselves within the app. Currently there are only two modes supported:
  - NFC / RFID: the user can us its NFC card
  - barcode / QR: the user can scan its QR code (it could be the QR code delivered to the user when the user has registered with any social network).

Please consider that some devices come without NFC or this module could be simply disabled.

Background

The very first UI element that can affect the user experience is the application background referenced from here:

TagByBackground TagByDocument.getBackground().

The background could be defined:
- as a solid color: **Integer getBackgroundColor()**
- as an image: **String getUrl()**

The above two cannot be combined and the background image url takes precedence if both are defined. As every image it is very important to optimise the application performance but providing background images in compressed format (like JPG) not PNG or BMP.

Social networks

After the user plays with your app, he can share his experience on on of the social networks: Facebook, Twitter or email (please note, currently only Facebook is supported). The social network settings can be found here:

TagBySocialNetworks TagByDocument.getSocialNetworks()

For each of the social network type there is a corresponding method in TagBySocialNetworks:
- **TagByFacebook getFacebook()**
  - **boolean isSupported()**: determines if Facebook is supported in this offer
- **TagByTwitter getTwitter()**
  - **boolean isSupported()**: determines if Facebook is supported in this offer
- **TagByEmail getEmail()**
  - **boolean isSupported()**: determines if Facebook is supported in this offer

The above options are configured on the web manager designer sharing options page.

Widgets

The most important part of the document are widgets. Each widget type represents a specific type of the UI element that is configured using TagBy web manager designer. It is up to you what widgets build up your application.

The are various type of widgets supported by the platform. You can find the following members of the **com.tagby.sdk.documents.widgets.TagByWidgetType**:

- **VIEW**
- **LABEL**
- **BUTTON**
- **IMAGE**
- **LOGO_IMAGE**
- **CAMERA_PREVIEW**
- **BARCODE_PREVIEW**
- **YOUTUBE**

For every widget supported, TagBy SDK v2 delivers corresponding controller (see **com.tagby.sdk.app.controllers** namespace) which will help you style your Android application UI elements based on the widget settings. Every controller introduces features specific to the UI element, but the common behaviour matches the one delivered by **TagByView** controller (see next section).

View (TagByView)

This is the simplest widget that will help you apply position and the background colour to any Android control (derived from android.view.View). In order to apply the widget to your view use TagByViewController like below:

new TagByViewController(mPlayButton, document.getPlayButton()).apply()

As you can see, configuring the application based on the document settings is quite simple. All you have to do is to reference Android UI element (mPlayButton above) and related document widget (getPlayButton) and let the controller do the job.

Some widgets can be missing in the document (if you decide to have so), so the controller by default hides any element for which there is no widget.

Label (TagByLabel)

This widget is a specialised version for Android **TextView** and delivers features like below:
- label text: see **TagByLabel.getText()**
- label text alignment: see **TagByLabel.getTextAlignment()**
- label font: see **TagByLabel.getFont()**

As before you don't have to know how to apply the properties of the widget to TextView, the SDK comes with:

new TagByLabelController(labelView, document.getInfoLabel()).apply();

The code above is quite the same as for TagByView. This pattern is reused for every other widget / controller.

Button

This widget is a specialised version for Android **Button** and delivers features like below:
- button text: see **TagByButton.getText()**
- button font: see **TagByButton.getFont()**

and the controller what works with this widget:

new TagByButtonController(mShareButton, document.getShareButton()).apply();

Image (TagByImage)

This widget works with two views delivered with the SDK:
- **NetworkImageView**
- **NetworkImageProgressView**

The former one is a standard Google Volley image control, while the latter one enhances the former one by displaying the progress indicator until the image is downloaded from the network.

Depending on the type of the Android view used, use one of the following controllers:

new TagByImageController(imageView, document.getImage()).apply();

or

new TagByProgressImageController(imageView, document.getImage()).apply();

Logo image (TagByLogoImage)

It is a specialised image that holds customer logo. Its behaviour is the same as TagByImage. You can also use the same controllers and Android UI views. As the web portal designer treats the customer logo a little bit different than regular image, we have introduced a specialised widget type. The sample application components don't treat logo image in any kind of a different way as images.

Camera preview (TagByCameraPreview)

The very first more complex widget type is camera. It is responsible for displaying a camera preview on the application screen. Please notice, this widget type is used for displaying and taking pictures not scanning barcode / QR.

The controller for this widget, namely **TagByCameraPreviewController** is responsible for rendering the camera preview or taking pictures. The controller needs to build more complex Android view hierarchy, so it needs to be applied to **FrameLayout** Android view like below:

new TagByCameraPreviewController(getActivity(), cameraPreview, document.getTakePhotoCameraPreview());

Please note, the controller isn't applied immediately after creation. This is because this controller uses a shared camera source, so it must be managed with more care. Basically you should call the following methods:
- **TagByCameraPreviewController.resume ()**
- **TagByCameraPreviewController.pause ()**

in appropriate places in your app (probably in the Activity / Fragment lifecycle methods named the same).

The code above will let you hide, show and place your camera preview within your app. In order to take pictures, the controller delivers additional method:

void takePicture(final PictureCallback callback)

Barcode preview (TagByBarcodePreview)

This widget is very similar to the previous one and below is the controller that deals with this kind of the widget:

new TagByBarcodePreviewController<TagByDocument>((TagByActivity<TagByDocument>)getActivity(), cameraView, document.getBarcodePreview());

Additionally this item automatically wires up into the TagBy application in order to deliver the information about any scanned tag (barcode or QR). So you don't need to worry scanning the tags. Just place the **FrameLayout** widget onto your Android XML layout file and use the controller.

YouTube (TagByYouTube)

The last widget type supported is YouTube. It is responsible for rendering Youtube videos on **YouTubePlayerFragment** from Google YouTube SDK.

new TagByYoutTubeController (youTubeFragment, document.getYouTube()).

TagBy web server API

The SDK comes from the API that works with the TagBy web server for retrieving documents, sharing info and so on. Most of the time you don't have to used this API directly as the base application libraries deals with the most scenarios. For referencing purposes here is the list of available commands:

- **ScheduleCommand**: retrieves the documents / offers configured for your device
- **CheckTagCommand**: checks if the scanned tag is known by the platform and retrieves the information about associated user.
- **EventCommand**: stores an analytics event
- **PublishCommand**: shares / publishes the information on the chosen social network

How to build a TagBy application?

Building TagBy application is quite simple as most of the features are delivered by the TagBy components. Basically you need the following components to start with:

- **TagBy SDK** components:
  - **com.tagby.sdk.jar** library
  - **crashlytics.jar** library
  - **YouTubeAndroidPlayerApi.jar** library
  - **com.tagby.sdk.resources** Android project
- **TagBy simple application** components:
  - **it.sephiroth.android.library** Android project
  - **com.tagby.sdk.simpleapp** Android project

The simple application components deliver everything that you can see after you have clicked **Share** in any of the TagBy delivered applications (like **Photobooth**, **Promo** or **Slot machine**). What differentiates these applications is the very first screen displayed after the offer has been selected. This is where you put your application design, we will name this a main fragment screen.

The process requires you to implement for components:

- document
- Android application
- Android activity
- Android main fragment screen
- Android manifest file.

Document

Here you define your widgets like below:

```
package com.example.demoapp;

import org.json.JSONException;
import org.json.JSONObject;

import android.os.Parcel;
import android.os.Parcelable;

import com.tagby.sdk.documents.TagByDocument;
import com.tagby.sdk.documents.widgets.json.TagByButton;
import com.tagby.sdk.documents.widgets.json.TagByImage;
import com.tagby.sdk.documents.widgets.json.TagByLabel;
import com.tagby.sdk.exceptions.TagByException;

/**
 * This is the class that represents your document / offer configuration.
 * <p>
 * You should put here everything that comes from the server and it is configured on the web portal.
 * It includes UI widgets but also the non-UI settings that are used to change your application
 * behavior.
 * </p>
 * <p>
 * Additionally in order to meet {@link Parcelable} requirements, some additional
 * members like constructor must be implemented.
 * </p>
 */
public class MyDocument extends TagByDocument {

	public final static String WIDGET_LOGO = "logo-image";
	public final static String WIDGET_IMAGE = "custom-image";
	public final static String WIDGET_INFO_LABEL = "info-label";
	public final static String WIDGET_SHARE_BUTTON = "share-button";

	public MyDocument(JSONObject data) throws JSONException, TagByException {
		super(data);
	}

	public MyDocument(Parcel parcel) throws JSONException, TagByException {
		this(new JSONObject(parcel.readString()));
	}

	public static final Parcelable.Creator<MyDocument> CREATOR = new
Parcelable.Creator<MyDocument>() {
    public MyDocument createFromParcel(Parcel in) {
	try {
		return new MyDocument(in);
	}
	catch (JSONException exception) {
		return null;
```

```
            }
            catch (TagByException exception) {
                    return null;
            }
        }

        public MyDocument[] newArray(int size) {
            return new MyDocument[size];
        }
    };

    /**
     * This is an example of a mandatory logo Image widget.
     */
    public TagByImage getLogo() {
                    return super.getWidgets().getItem(WIDGET_LOGO);
            }

    /**
     * This is an example of a optional Image widget.
     */
    public TagByImage getImage() {
                    return super.getWidgets().optItem(WIDGET_IMAGE);
            }

    /**
     * Another example of optional label. IF this item is missing
     * in the document it will be hidden by associated widget controller.
     */
    public TagByLabel getInfoLabel() {
                    return super.getWidgets().optItem(WIDGET_INFO_LABEL);
            }

    /**
     * And this button widget must be always presents.
     */
    public TagByButton getShareButton() {
                    return super.getWidgets().getItem(WIDGET_SHARE_BUTTON);
            }
}
```

Android application

In this component, all you have to do, is to convert the list of the TagByDocument instances delivered to the application into an application specific **MyDocument** instances:

```
package com.example.demoapp;

import java.util.ArrayList;
import java.util.List;

import com.tagby.sdk.app.TagByApplication;
import com.tagby.sdk.app.TagByApplicationDocuments;
import com.tagby.sdk.documents.TagByDocument;

public class MyApplication extends TagByApplication<MyDocument>{
	/**
	 * This method will be called by Tag'By runtime components after the offers are fetched from the server.
	 */
	@Override
	protected TagByApplicationDocuments<MyDocument> createDocuments(List<TagByDocument> documents) throws Exception {
		List<MyDocument> myDocuments = new ArrayList<MyDocument>(documents.size());

		for(TagByDocument document: documents) {
			// we don't want to display the offers that have no social network configured
			// if your application doesn't work well with all social network providers
			// you must take it into account here
			if (document.getSocialNetworks().anySupported() == false)
				continue;

			myDocuments.add(
				new MyDocument (document.getData()));
		}

		// additionally we get rid of any NFC enabled offers if the application is run on non-NFC device
		myDocuments = super.filterOutNfcEnabled(myDocuments);

		return new TagByApplicationDocuments<MyDocument> (myDocuments);
	}
}
```

Android activity

The activity controls the whole application process. Here you are notified about selected offer or when the user has closed the sharing screens. Here you implement what to do after the document has been selected and what UI to display:

```
package com.example.demoapp;

import com.example.demoapp.R;
import com.tagby.sdk.api.PublishCommand;
import com.tagby.sdk.app.MessagePresenter;
import com.tagby.sdk.app.TagByApplication;
import com.tagby.sdk.app.TagByAsyncTask;
import com.tagby.sdk.resources.ResourceUtils;
import com.tagby.sdk.simpleapp.SimpleTagByActivity;
import com.tagby.sdk.simpleapp.fragments.MessageFragment;

import android.app.FragmentManager;
import android.os.Handler;
import android.util.Log;

public class MainActivity extends SimpleTagByActivity<MyDocument> {
	private final static String TAG_MAIN_FRAGMENT = "mainFragment";
	private final static String TAG_MESSAGE = "message";

	private final Handler mHandler = new Handler();

	/**
	 * This is the main entry point after the document has been selected.
	 * It is the application responsibility to present document UI.
	 */
	@Override
	public void onDocumentSelected(MyDocument document) throws Exception {
		super.onDocumentSelected(document);

		final FragmentManager fm = getFragmentManager();
		fm.beginTransaction()
		.replace(R.id.main_frame, MyMainFragment.newInstance(document), TAG_-
MAIN_FRAGMENT)
		.addToBackStack(TAG_MAIN_FRAGMENT)
		.commit();
	}

	/**
	 * Called by the base class after share now screen has been closed.
	 * Depending on the application type it could be required to update some UI here.
	 * For this demo application we restore the document switch and share buttons.
	 */
	@Override
	public void onShareClosed() throws Exception {
		super.onShareClosed();
```

```
		MyMainFragment fragment = (MyMainFragment)getFragmentManager().findFrag-
mentByTag(TAG_MAIN_FRAGMENT);
		if (fragment != null) {
			fragment.restart();
		}
	}

	/**
	 * Called by the base class after the user has identified himself and his friends and decides
to share.
	 */
	@Override
	public void onShareNow() throws Exception {
		Log.d(TAG, "onShareNow");

		super.onShareNow();

		// we call the base class method in order
		// to go back to the main fragment screen
		super.onShareClosed();

		new PublishTask().execute();
	}

	private void closeSharedMessage () throws Exception {
		// close "Sharing now..." message
		getFragmentManager().popBackStack();
		((MyMainFragment)getFragmentManager().findFragmentByTag(TAG_-
MAIN_FRAGMENT)).restart();
	}

	@Override
	protected void safelyOnBackPressed() throws Exception {
		Log.d(TAG, "safelyOnBackPressed");

		// we would like to block BACK button as far as the message button is displayed
		FragmentManager fm = getFragmentManager();
		if (fm.getBackStackEntryCount() != 0) {
			if (fm.findFragmentByTag(TAG_MESSAGE) != null)
				return;
		}

		super.safelyOnBackPressed();
	}

	/**
	 * Publishes the message.
	 */
	private class PublishTask extends TagByAsyncTask<Void, Void, Void> {
		protected PublishTask() {
			super(MainActivity.this, ResourceUtils.publishing(MainActivity.this), true);
		}

		@Override
		protected void onPreExecute() {
```

```
                super.onPreExecute();

                getFragmentManager()
                .beginTransaction()
                .add(R.id.main_frame,
MessageFragment.newInstance(getString(R.string.sharing_message)), TAG_MESSAGE)
                .addToBackStack(null)
                .commit();
        }

        @Override
        protected Void safelyDoInBackground(Void... params) throws Exception {
                new PublishCommand(
                        MainActivity.this.getRemoteDeviceId(),
                        MainActivity.this.getOfferId(),
                        null,
                        null,
                        null,
                        TagByApplication.getInstance().getUsers().getUsers()).execute();

                return null;
        }

        @Override
        protected void onFailure(Exception exception) {
                super.onFailure(exception);

                try {
                        // close "Sharing now..." message
                        getFragmentManager().popBackStack();

((MyMainFragment)getFragmentManager().findFragmentByTag(TAG_MAIN_FRAGMENT)).restart(
);
                } catch (Exception e) {
                        MessagePresenter.getInstance().showException(e);
                }
        }

        @Override
        protected void onSuccess(Void unused) {
                // replace "Sharing now..." message with "Thank you"

((MessageFragment)getFragmentManager().findFragmentByTag(TAG_MESSAGE)).setMessage(g
etString(R.string.shared_message));

                mHandler.postDelayed(
                        new Runnable() {
                                @Override
                                public void run() {
                                        try {
                                                closeSharedMessage();
                                        }
                                        catch (Exception exception) {
                                MessagePresenter.getInstance().showException(exception);
                                        }
```

```
                }}, 3000);
            }
        }
}
```

Android main fragment screen

In your main fragment, you consume the document widgets and apply the settings to your UI items:

```
package com.example.demoapp;

import android.content.Context;
import android.os.Bundle;
import android.util.DisplayMetrics;
import android.util.Log;
import android.view.Display;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.TextView;

import com.example.demoapp.R;
import com.tagby.sdk.app.TagByApplication;
import com.tagby.sdk.app.TagByDocumentFragment;
import com.tagby.sdk.app.TagByOnClickListener;
import com.tagby.sdk.app.controllers.DocumentBackgroundController;
import com.tagby.sdk.app.controllers.TagByButtonController;
import com.tagby.sdk.app.controllers.TagByLabelController;
import com.tagby.sdk.app.controllers.TagByProgressImageController;
import com.tagby.sdk.simpleapp.SimpleTagByActivity;
import com.tagby.sdk.ui.NetworkImageProgressView;
import com.tagby.sdk.utils.Check;

/**
 * This is the main application fragment responsible for rendering application specific main screen.
 *
 * The contents of this fragment are either bundled within the app or configured on via Tag'By Manager
 * using offer designer.
 */
public class MyMainFragment extends TagByDocumentFragment<MyDocument>{
	// we need to keep the reference to these controls
	// as we would like to hide / show it based on the application state
	private View mSwitchDocument;
	private Button mShareButton;

	public static MyMainFragment newInstance (MyDocument document) {
		Check.Argument.isNotNull(document, "document");

		MyMainFragment fragment = new MyMainFragment();

		fragment.setArguments(fragment.createBundle(document));

		return fragment;
	}
```

```
	/**
	 * In this method we need to setup the fragment based on the selected offer.
	 */
	@Override
	protected View safelyOnCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) throws Exception {
		Log.d(TAG, "safelyOnCreateView");

		// retrieve the current offer
		final MyDocument document = getDocument();

		WindowManager wm = (WindowManager) TagByApplication.getInstance().getSys-
temService(Context.WINDOW_SERVICE);
		Display display = wm.getDefaultDisplay();
		DisplayMetrics metrics = new DisplayMetrics();
		display.getMetrics(metrics);

		View rootView = inflater.inflate(R.layout.fragment_main, container, false);
		NetworkImageProgressView logoView =
(NetworkImageProgressView)rootView.findViewById(R.id.fragment_main_logo);
		NetworkImageProgressView imageView =
(NetworkImageProgressView)rootView.findViewById(R.id.fragment_main_image);
		TextView labelView = (TextView)rootView.findViewById(R.id.fragment_main_label);
		mSwitchDocument =
rootView.findViewById(R.id.fragment_main_switch_document);
		mShareButton = (Button)rootView.findViewById(R.id.fragment_main_share);

		// the following lines configure fragment widgets as designed in the offer
		// do not forget to call 'apply' for every newly created controller
		DocumentBackgroundController.Instance.apply(rootView, document.get-
Background(), metrics);
		new TagByButtonController(mShareButton, document.getShareButton()).ap-
ply();
		new TagByProgressImageController(logoView, document.getLogo()).apply();
		new TagByProgressImageController(imageView,
document.getImage()).apply();
		new TagByLabelController(labelView, document.getInfoLabel()).apply();

		mSwitchDocument.setOnClickListener(
				new TagByOnClickListener() {
					@Override
					protected void safelyOnClick(View v) throws Exception {
						setControls (false);

						((SimpleTagByActivity<?>)getActivity()).onDocu-
mentSwitch();
					}
				});

		mShareButton.setOnClickListener(
			new TagByOnClickListener() {
				@Override
				protected void safelyOnClick(View v) throws Exception {
					setControls (false);
```

```
                                        ((SimpleTagByActivity<?>)getActivity()).onShare();
                                }
                        });

                return rootView;
        }

        /*
         * Called by the {@link MainActivity}
         */
        public void restart() {
                setControls (true);
        }

        private void setControls(boolean enabled) {
                mSwitchDocument.setVisibility(enabled ? View.VISIBLE : View.GONE);
                mShareButton.setVisibility(enabled ? View.VISIBLE : View.GONE);
        }
}
```

Android manifest file

And finally you need to add some permissions and activities to your **AndroidManifets.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.demoapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.NFC" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <uses-feature android:name="android.hardware.camera" />
    <uses-feature
        android:name="android.hardware.nfc"
        android:required="true" />

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="16" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:name="com.example.demoapp.MyApplication"
        android:theme="@style/MyAppTheme" >
        <activity
            android:name="com.example.demoapp.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name="com.tagby.sdk.app.TagByWebViewActivity"/>
        <activity android:name="com.tagby.sdk.app.TagByTagRegistrationActivity"/>
        <activity android:name="com.tagby.sdk.app.TagByManualRegistrationActivity"/>

        <meta-data android:name="com.crashlytics.ApiKey" android:value="Your Crashlytics
API Key"/>
    </application>
</manifest>
```