## About This Document

This document contains information on how to use an external Android library called ActionBarSherlock to bring the Ice Cream Sandwich version of the Android ActionBar to older versions of the platform.

To understand the contents of this article one should know the basics of Android application development including Fragments and ActionBar concepts.

## 1. Motivation for using ActionBarSherlock

With the release of the Honeycomb Android version dedicated to tablet devices, a new set of user interface features were introduced to provide better support for larger screen sizes. The most prominent additions were Fragments and ActionBar concepts.

The Fragment is best described as being a sub-Activity, though it does not extend Activity or Context classes. The main idea behind this concept is to separate functionalities from Activities and put them into a more specialized block to promote code reusability. You should keep in mind that using Activities in a regular manner is not discouraged and you can still use it.

Another concept introduced in API 11 is the ActionBar. Since ActionBar is a central place for the navigation in the new Android UI design guidelines, it is assumed to be visible most of the time during the lifetime of your app. You can customize its look and feel to provide a consistent branding experience so your Activities are easily recognizable.

When new features are added to Android there is always a chance you will find them in the Support Library when looking to use them on older platform versions. Unfortunately, only Fragments received such support and the ActionBar is available from API 11 only. ActionBarSherlock is supposed to lift this limit.

## 2. Features and differences

Since ActionBarSherlock's main purpose is to bring missing classes and methods to mature (pre-Honeycomb) devices, its APIs are designed to mimic the original ones. The library does not provide any new features, so regular Android documentation should be the starting point when trying to find information about ActionBar and Fragments.

The naming convention is similar to that of the native one, but some minor differences occur in class and method names. You should pay attention to import proper classes referring to the tables below for a representation of the major differences between the APIs.

| Android API package android.app | ActionBarSherlock package com.actionbarsherlock.app |
|---|---|
| ActionBar | ActionBar |
| Activity | SherlockActivity |
| DialogFragment | SherlockDialogFragment |
| ExpandableListView | SherlockExapandableListView |
| Fragment | SherlockFragment |
| FragmentActivity | SherlockFragmentActivity |
| ListActivity | SherlockListActivity |
| ListFragment | SherlockListFragment |
| PreferenceActivity | SherlockPreferenceActivity |

[Table 1] Class name differences in .app package between the Support Library and ActionBarSherlock APIs.

| Android API package android.view | ActionBarSherlock package com.actionbarsherlock.view |
|---|---|
| ActionMode | ActionMode |
| ActionProvider | ActionProvider |
| CollapsibleActionView | CollapsibleActionView |
| Menu | Menu |
| MenuInflater | MenuInflater |
| MenuItem | MenuItem |
| SubMenu | SubMenu |
| Window | Window |

[Table 2] Classes in .view package of the Support Library and ActionBarSherlock APIs.

| Android API package android.widget | ActionBarSherlock package com.actionbarsherlock.widget |
|---|---|
| ActivityChooserModel | ActivityChooserModel |
| ActivityChooserView | ActivityChooserView |
| SearchView | SearchView (API 8+) |
| ShareActionProvider | ShareActionProvider |
| SuggestionsAdapter | SuggestionsAdapter |

[Table 3] Classes in .widget package of the Support Library and ActionBarSherlock APIs.

With class names the major differences include Activity, Fragment and their subclasses. Thankfully, those changes are consistent and when you import a class you should check if there is a Sherlock-prefixed one first.

| Android API | ActionBarSherlock |
|---|---|
| Activity. getActionBar() | SherlockActivity. getSupportActionBar() |
| Activity. getMenuInflater() | SherlockActivity. getSupportMenuInflater() |
| Activity. invalidateOptionsMenu() | SherlockActivity. supportInvalidateOptionsMenu() |
| Activity. setProgress(int) | SherlockActivity. setSupportProgress(int) |
| Activity. setProgressBarIndeterminate(boolean) | SherlockActivity. setSupportProgressBarIndeterminate(boolean) |
| Activity. setProgressBarIndeterminateVisibility(boolean) | SherlockActivity. setSupportProgressBarIndeterminateVisibility(boolean) |
| Activity. setProgressBarVisibility(boolean) | SherlockActivity. setSupportProgressBarVisibility(boolean) |
| Activity. setSecondaryProgress(int) | SherlockActivity. setSupportSecondaryProgress(int) |

[Table 4] Method name differences in Activity class between the Support Library and ActionBarSherlock APIs.

| Android API | ActionBarSherlock |
|---|---|
| Fragment. getActivity() | SherlockFragment. getSherlockActivity() |

[Table 5] Method name differences in Fragment class between the Support Library and ActionBarSherlock APIs.

API differences include method names too. As with classes, you should make sure you use methods with Support- and Sherlock- prefixes.

The general rule in using ActionBarSherlock and the Support Library (which the ActionBarSherlock actually tries to extend) is represented in the diagram below.

[Diagram 1] Which APIs to look for when trying to use a class or a method when using ActionBarSherlock.

## 3. Using ActionBarSherlock

As stated in this document, ActionBarSherlock does not provide any new features, as its main purpose is to bring missing ActionBar-related classes to the Support Library. The library design explicitly requires its users to import ActionBarSherlock's packages whenever they are available.

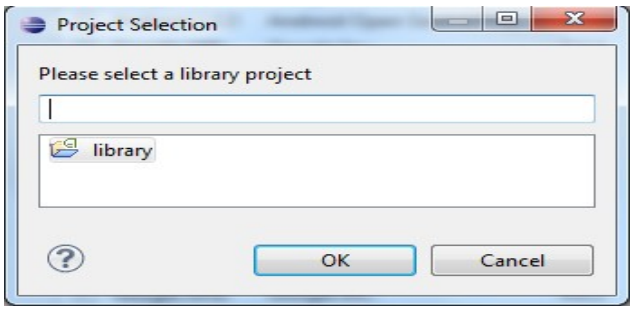
For example, you should import com.actionbarsherlock.app.SherlockFragmentActivity instead of the android.support.v4.app.FragmentActivity even if it appears that your application is working fine.

The main reason for such a requirement is that ActionBarSherlock uses native implementation if available and falls back to its own implementation on older platforms. If you do not specify those imports correctly, it can turn out that your app runs properly on Ice Cream Sandwich or Jelly Bean but fails on older versions. The other implication regarding this mechanism is the need to provide style names twice when customizing Sherlock's themes. It is highly advised to test your app during development stages on different API versions to eliminate potential problems early on.

### 3.1. Installation requirements

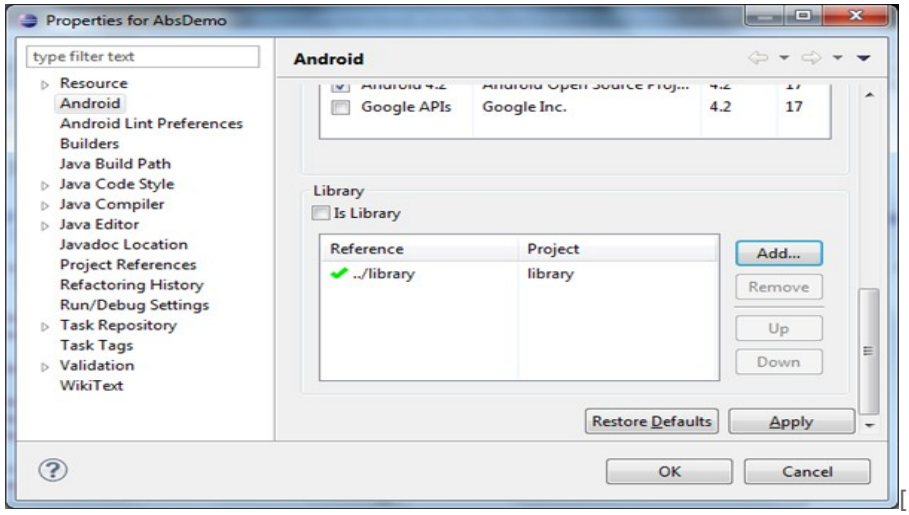
ActionBarSherlock is distributed as an Android library project meaning that you have to import the existing project into Eclipse (File > Import > General > Existing Projects into Workspace).

After importing ActionBarSherlock library you can quickly reference the library in your project by going to Project > Properties > Android > Library and pressing the Add button. You should see a list of Android library projects that can be referenced.




[Image 1] A list of Android library projects that can be referenced by your app.

After selecting the library you should see a screen similar to the one below.



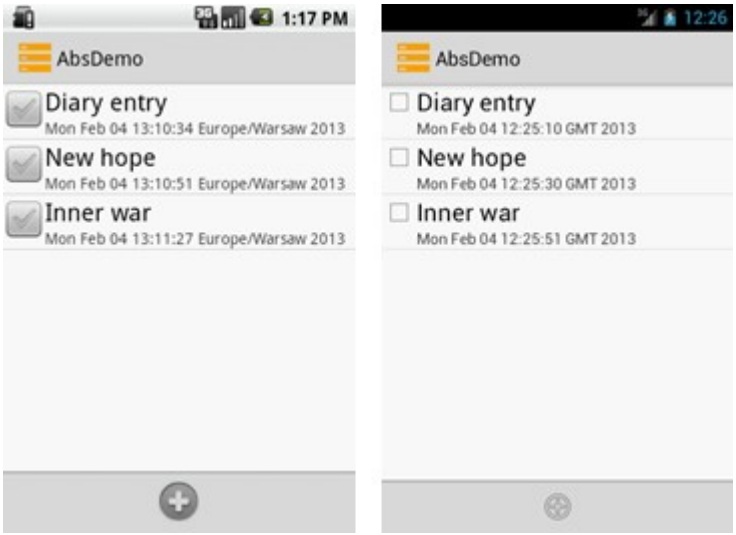

[Image 2] ActionBarSherlock's library successfully referenced in your project.

Now you should be able to import classes from the ActionBarSherlock library.

The demo application also uses the Ormlite library to keep the notes created with the application. You should download proper jar files (ormlite-core-4.42.jar and ormlite-android-4.42.jar at the time of writing this article) from the Ormlite site. You can reference those Ormlite jar files by simply copying them to the libs directory of your project.

## 3.2. Demo app

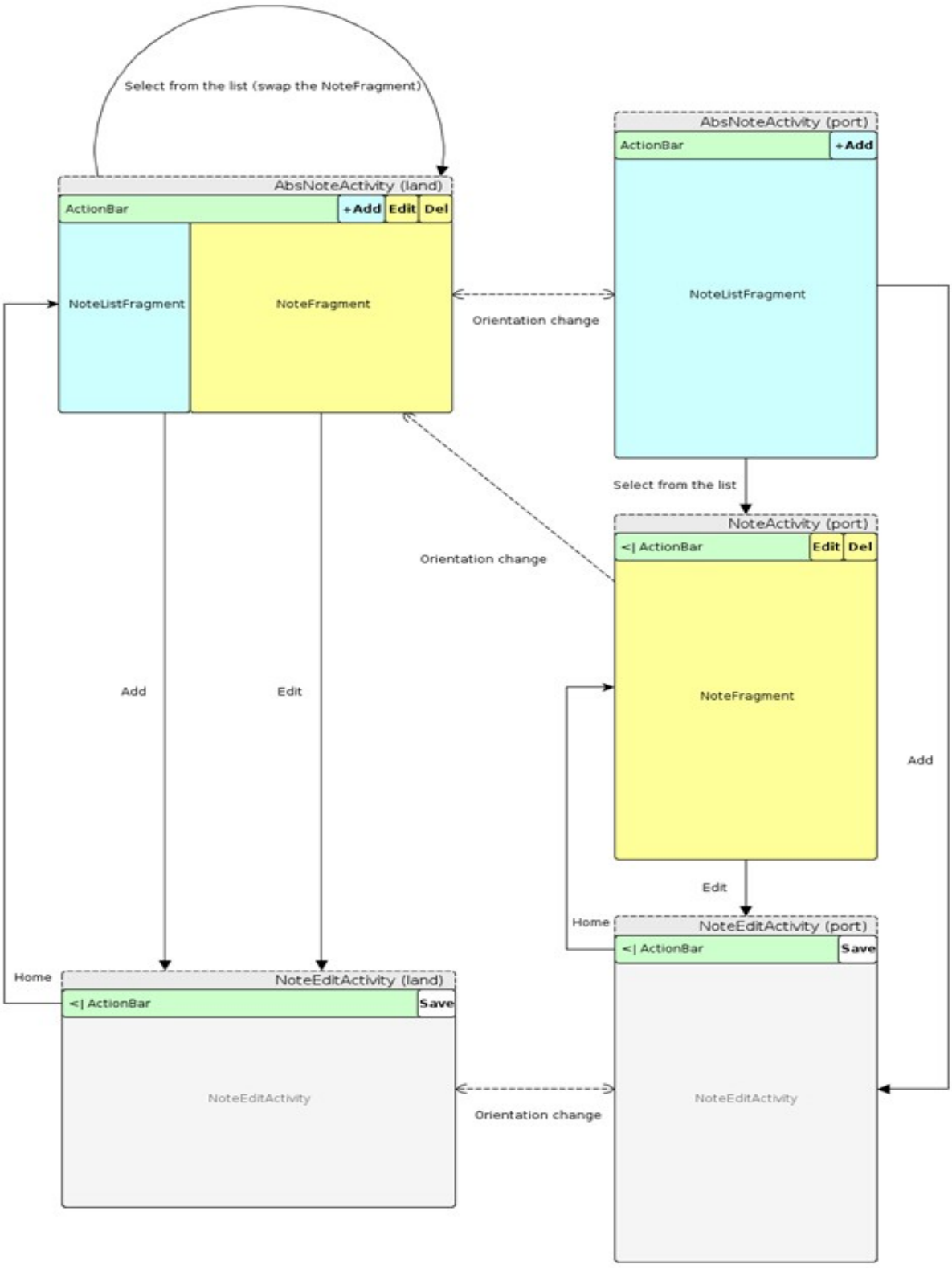
This article provides sample usage of the ActionBarSherlock library based on the AbsDemo app developed for the purpose of this document. The application's main functionality is to store, edit and delete simple notes.




[Image 3] AbsDemo's main screen running on Android 2.1 (Éclair – left) and 4.2 (Jelly Bean – right) in portrait mode.

The screen flow is presented in the Diagram 2. It consists of 3 Activities and 2 Fragments:

- AbsNoteActivity - the main one which has an optional feature when running in landscape mode to show two Fragments at once (both NoteListFragment holding a list of notes and NoteFragment that shows a specified note)
- NoteActivity - used if the AbsNoteActivity does not show the NoteFragment (in single pane mode only)
- NoteEditActivity - does not hold any Fragments. It is a regular Activity that has explicitly specified View hierarchy that you can use to edit a note.




[Diagram 2] AbsDemo UI flow. Note that ActionItems added to the ActionBar depend on the visible Fragment.

## 3.3. AbsNoteActivity

Our main Activity shows a few of the concepts available in the ActionBarSherlock library:

- Fragments – a reusable and modular section of an Activity (please refer to the Android Fragments guideline for a full explanation)
- ActionBar (and Action items) – a centralized navigation pattern in the modern Android UI (a guide is also available in the Android documents about Action Bar)
- ActionMode – a special mode of interaction with a group of elements that appears in place of the ActionBar (a complete reference can also be found in the Menus section of the Android documentation)

In the code fragment below you can see that AbsNoteActivity implements few interfaces, most notably the ActionMode.Callback and two custom ones:

- NoteListFragment.OnNoteSelectedListener
- NoteAdapter.OnNoteCheckedListener.

```
src/com/sprc/absdemo/AbsNoteActivity.java:

public class AbsNoteActivity extends SherlockFragmentActivity implements Callback,
OnNoteSelectedListener, OnNoteCheckedListener, LoaderCallbacks<List<Note>> {
...
```

[Code 1] AbsNoteActivity is a subclass of SherlockFragmentActivity. Additionally, it implements ActionMode's Callback interface and LoaderManager's LoaderCallbacks.

The ActionBar is used as a replacement of the regular menu button behavior, as it provides a set of ActionItems. You can use the regular options API to add new items to the ActionBar as shown in the code snippet below.

```
src/com/sprc/absdemo/AbsNoteActivity.java:

...

@Override
public boolean onCreateOptionsMenu(Menu menu) {

    // Use the regular *Activity.onCreateOptionsMenu method to inflate your menu
    // Warning: remember to import proper MenuInflater

    MenuInflater inflater = getSupportMenuInflater();
    inflater.inflate(R.menu.menu_list, menu);

    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.menu_item_delete:
```

```
        // Handle the option item click here

        return true;
    default:
        return false;
    }
}
```

[Code 2] Adding new ActionItems and handling their click events.

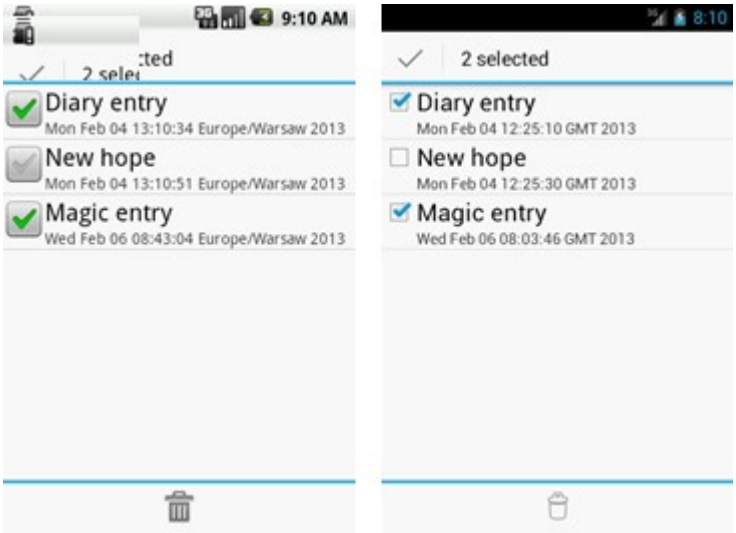
Using ActionMode on the other hand is straightforward. You only have to implement four methods from the ActionMode.Callback interface – onCreateActionMode, onPrepareAcitonMode, onActionItemClicked and onDestroyActionMode. Showing the ActionMode is done by calling the Activity.startActionMode(ActionMode.Callback) method and to close the ActionMode just call the ActionMode.finish().

```
src/com/sprc/absdemo/AbsNoteActivity.java:

...

@Override
public boolean onCreateActionMode(ActionMode mode, Menu menu) {

    // Inflate your ActionMode items using a MenuInflater and hold
    // the ActionMode reference.
    // Warning: make sure you import the com.actionbarsherlock.view.MenuInflater

    MenuInflater inflater = getSupportMenuInflater();
    inflater.inflate(R.menu.action_mode_list, menu);

    mActionMode = mode;

    return true;
}

@Override
public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
    return false;
}

@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
    switch (item.getItemId()) {
    ...
```

```
    case R.id.action_item_delete:

        // Handle item clicks in switch-case statements

        ...

        mode.finish();
        return true;
    default:
        return false;
    }
}

@Override
public void onDestroyActionMode(ActionMode mode) {

    // Empty the reference to ActionMode and do a clean up

    mActionMode = null;

    ...
}
```

[Code 3] ActionMode.Callback interface implementation in the AbsNoteActivity.

Additionally, AbsNoteActivity uses the Loader concept from the API 11 provided by the Support Library, which in our case assists loading a Note list from the database. As this feature is not strictly related to ActionBarSherlock, please refer to the Loader guidelines in the Android documentation for more details regarding this topic.




[Image 4] ActionMode is properly shown on Android 2.1 (Éclair – left) as well as on 4.2 (Jelly Bean – right).

### 3.4. NoteListFragment

In order to make your Fragments reusable you should design them with loose coupling in mind. It means that you should not store a reference to the Activity that hosts your Fragment explicitly, but provide an interface that should be implemented by your Fragment's user instead.

```
src/com/sprc/absdemo/NoteListFragment.java:

public interface OnNoteSelectedListener {
     void onNoteSelected(Note n, int position, long id);
}

src/com/sprc/absdemo/AbsNoteActivity.java:

@Override
protected void onCreate(Bundle savedInstanceState) {
     ...
     NoteListFragment fragmentNoteList = getNoteListFragment();
     fragmentNoteList.addOnNoteSelectedListener(this);
     ...
}

@Override
public void onNoteSelected(Note n, int position, long id) {
     if (mActionMode != null)
          mActionMode.finish();

     mPaneMode.onNoteSelected(n, position, id);
}
```

[Code 4] Adding the NoteListFragment's OnNoteSelectedListener so it receives a callback when a note is clicked in the NoteListFragment.

### 3.5. NoteFragment

The concept of Fragment reusability helps to put the NoteFragment in two Activities without too much code duplication. This way we can show the NoteFragment in the dual-pane mode in AbsNoteActivity as well as in NoteActivity.

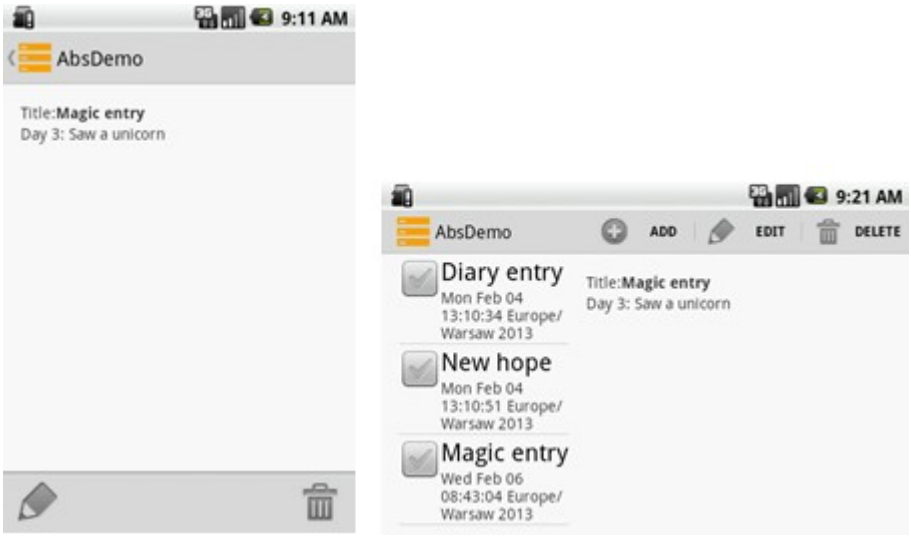
Creating ActionItems related to a Fragment is possible through a similar interface to the one available in the Activity. You should override onCreateOptionsMenu from the Fragment class and instead use the MenuInflater provided as the method's argument. There is a pitfall in this solution though. In order for the onCreateOptionsMenu method to be called at all, you have to invoke the setHasOptionsMenu(true) in your onCreate first.

```
src/com/sprc/absdemo/NoteFragment.java:
```

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

    // Remember to call this method to enable onCreateOptionsMenu

    setHasOptionsMenu(true);
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    inflater.inflate(R.menu.menu_note, menu);
}
```

[Code 5] Creating options menu in the NoteFragment so options related to this particular Fragment get attached to the ActionBar of whatever Activity is hosting the Fragment.

You should note that the NoteFragment does not handle option clicks by itself. As those events get propagated to the onOptionsItemSelected method of the Activity, we can handle them there.




[Image 5] NoteFragment is used in two different Activities – in NoteActivity (left) and AbsNoteActivity (right). It also provides its specific action items (Edit and Delete) to the ActionBar.

## 3.6. Dual-pane mode support

Since the UI requirements for our AbsDemo introduced a dual-pane layout to better utilize the screen size on larger devices, now is the right moment to design a proper abstraction layer for the task.

The AbsNoteActivity's instantiates one of the PaneMode implementations (DualPaneMode or OnePaneMode) in the onCreate method. Both of them extend the abstract PaneMode class that requires its subclasses to implement a couple of specific methods. This way we can just perform a single check for the appropriate mode and use its corresponding logic. For instance, in dual pane mode we want to swap the NoteFragment that shows the note contents in the right side of the

layout, but in the OnePaneMode we have to run a separate Activity that will host the NoteFragment.

```
src/com/sprc/absdemo/AbsNoteActivity:

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

    ...

    // Instantiate the proper PaneMode logic helper

    if (getResources().getBoolean(R.bool.dual_pane))
        mPaneMode = new DualPaneMode(this);
    else
        mPaneMode = new OnePaneMode(this);

    ...

    mPaneMode.onCreate(savedInstanceState);
}


@Override
public void onNoteSelected(Note n, int position, long id) {

    if (mActionMode != null)
        mActionMode.finish();

    mPaneMode.onNoteSelected(n, position, id);
}

private static abstract class PaneMode {

    protected final WeakReference<AbsNoteActivity> mWeakActivity;

    public PaneMode(AbsNoteActivity activity) {
        mWeakActivity = new WeakReference<AbsNoteActivity>(activity);
    }

    public abstract void onCreate(Bundle savedInstanceState);

    public abstract void onNoteSelected(Note n, int position, long id);
```

```
    public abstract void onAddNote();

    ...
}

private static class OnePaneMode extends PaneMode {

    ...

    @Override
    public void onNoteSelected(Note n, int position, long id) {

        // In OnePaneMode we swap start the NoteActivity to show the NoteFragment

        AbsNoteActivity activity = mWeakActivity.get();

        Intent intent = new Intent(activity, NoteActivity.class);
        intent.putExtra(activity.getString(R.string.key_id), id);
        activity.startActivityForResult(intent, NoteActivity.REQUEST_SHOW_NOTE);
    }

    ...
}

private static class DualPaneMode extends PaneMode {

    ...

    @Override
    public void onNoteSelected(Note n, int position, long id) {

        // In DualPaneMode we swap the NoteFragment

        AbsNoteActivity activity = mWeakActivity.get();

        NoteFragment currentFragment = activity.getNoteFragment();

        if (currentFragment == null || currentFragment.getNoteId() != id) {
            // Let's swap the NoteFragment with the selected one


activity.getSupportFragmentManager().beginTransaction().replace(R.id.fragment_note,
```

```
NoteFragment.newInstance(n)).commit();
        }
    }

    ...
}
```

[Code 6] Instantiating the proper PaneMode implementation based on the R.bool.dual_pane resource. The Activity will perform a different action based on the current PaneMode instance referenced by mPaneMode.

This kind of abstraction layer can be helpful especially if you plan to add different fragment layout modes to your app. It can also make the code more readable since we eliminate the if-statements.

The dual-pane mode has a specific implication in the UI flow. When we are in portrait mode and display a NoteActivity showing a single NoteFragment, switching the orientation should make the dual-pane mode visible instead. This means we have to go to the AbsNoteActivity in such a case. This is performed in the NoteActivity's onCreate method where we call finish() when R.bool.dual_pane is true and AbsNoteActivity is popped from the back stack.

```
src/com/sprc/absdemo/AbsNoteActivity:

public class NoteActivity extends SherlockFragmentActivity {

    ...

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

        ...

        if (getResources().getBoolean(R.bool.dual_pane)) {
            // Let's get back to the AbsNotectivity since it should show both
            // fragments at the same time

            Intent data = new Intent();
            data.putExtra(EXTRA_NOTE_ID, id);
            setResult(0, data);
            finish();
        }

        ...

    }
    ...
```

[Code 7] Adding the NoteListFragment's OnNoteSelectedListener so it receives a callback when a note is clicked in the NoteListFragment.

For testing purposes the AbsDemo app shows the dual-pane mode only in landscape mode with the smallest width of the screen as at least 180dp. Since the sw<X>dp modifier was added in API 13 the dual-pane mode will be only available on newer platforms. As an exercise you can adjust those resource modifiers to display the dual-pane mode on older platforms too (hint: use size-specific modifiers, i.e. small, normal, large or xlarge). This can help you better understand how those modifiers are handled.

## 3.7. NoteActivity

In our AbsDemo we want to host our NoteFragment in portrait mode on its own. This implies that we should create a separate Activity to handle such case. It is possible to put the Fragment in AbsNoteActivity, but it would make the code much less readable and error-prone.

As already mentioned, NoteActivity is finished whenever we switch to dual-pane mode. The other interesting part of this Activity is that it changes the Home icon of the ActionBar to a clickable button. We handle it in the onOptionsItemSelected method with the android.R.id.home as the id.

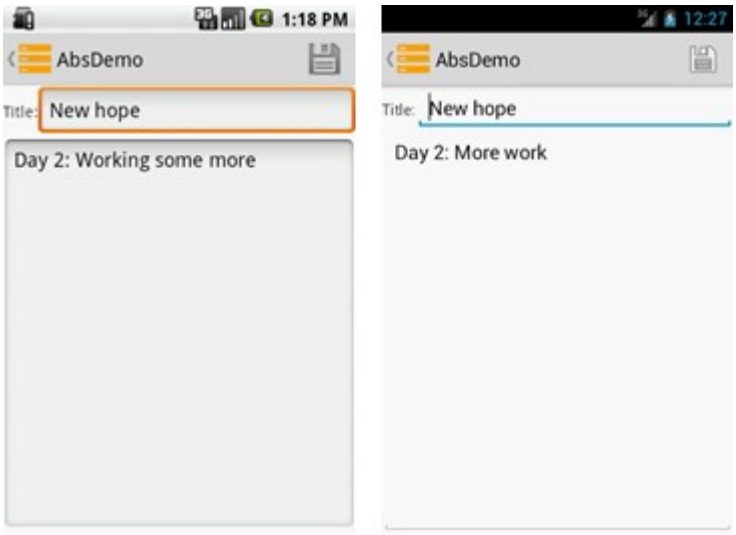
```
src/com/sprc/absdemo/NoteActivity:

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

    ...
    if (getSupportActionBar() != null)
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
}

...
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case android.R.id.home:
        // ActionBar's Home button clicked
        ...
        return true;
    ...
    default:
        return false;
    }
}
...
```

[Code 8] Changing the application icon into a clickable button is done with a single call to setDisplayHomeAsUpEnabled(true).

### 3.8. NoteEditActivity

The NoteEditActivity is an example of mixing the concepts of Activities and Fragments in a single application UI flow. We could have used a specialized Fragment to edit a note and just attach it to this Activity, but since we do not need to reuse this functionality anywhere else we implement it directly in NoteEditActivity.

Just as in the NoteActivity it shows its icon as a button that removes it from the back stack and shows the previous one. Additionally, it handles the option items clicks to perform the save operation.




[Image 6] NoteEditActivity running on Android 2.1 (Éclair – left) and 4.2 (Jelly Bean – right) in portrait mode.

## 4. Custom themes and styles

The themes mechanism in ActionBarSherlock is similar to API version. If your app is running on an older platform version it will use the fallback solution provided by ActionBarSherlock. On the other hand, when a newer API is available (at least 14) it will use the native ones. This introduces one simple drawback for the user of the library.

You have to remember to use the themes provided by ActionBarSherlock in order for the library to work correctly. Fortunately, it mimics the themes mechanism and lets you customize them when needed. You should remember to make your app use the ActionBarSherlock themes as shown in the code snippet below.

```
AndroidManifest.xml:

<application
    ...
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.sprc.absdemo.AbsNoteActivity"
        android:label="@string/app_name"
        android:uiOptions="splitActionBarWhenNarrow" >
```

```
        ...
    </activity>
    ...
</application>

res/values/styles.xml:

<resources
    xmlns:android=http://schemas.android.com/apk/res/android
    xmlns:tools="http://schemas.android.com/tools">

    <style name="AppTheme" parent="@style/Theme.Sherlock.Light">
        <!-- Override style settings here -->
    </style>

</resources>
```

[Code 9] The AbsDemo application uses the theme from ActionBarSherlock and provides a place to customize it in the styles.xml resource file. Additionally, the AbsNoteActivity has an uiOption set to split the ActionBar items when there is not enough space to show them.

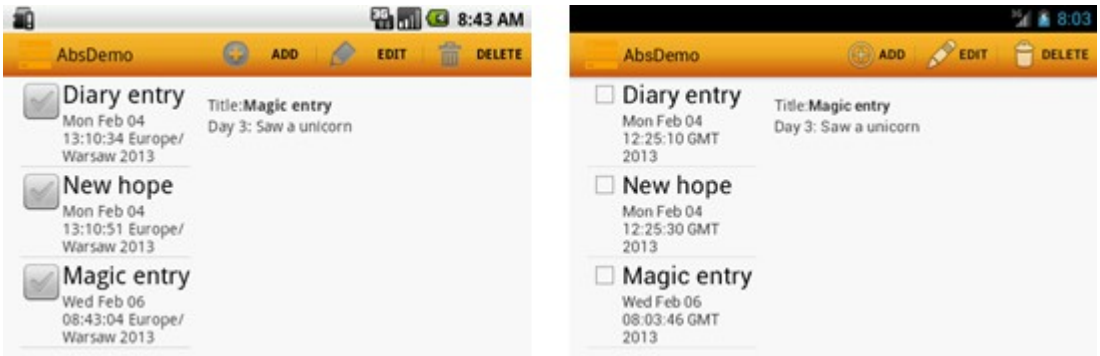
As an exercise we will show you how to quickly modify the theme by changing the default ActionBar background. Pay special attention to the res/values/styles.xml file as it contains the trickiest part of ActionBarSherlock theme customization, namely the need to provide attribute names twice for different namespaces.

```
res/values/colors.xml:

<resources>
...
    <color name="actionbar_startcolor">#CC5511</color>
    <color name="actionbar_centercolor">#EEAA20</color>
    <color name="actionbar_endtcolor">#FFBB33</color>
</resources>

res/drawable/actionbar_background.xml:

<shape xmlns:android="http://schemas.android.com/apk/res/android"
       xmlns:tools="http://schemas.android.com/tools" >

    <gradient
        android:angle="90"
        android:centerColor="@color/actionbar_centercolor"
        android:centerX="50%"
        android:endColor="@color/actionbar_endcolor"
```

```
        android:startColor="@color/actionbar_startcolor" />

</shape>

res/values/styles.xml:

<resources xmlns:android=http://schemas.android.com/apk/res/android
            xmlns:tools="http://schemas.android.com/tools">

    <style name="AppTheme" parent="@style/Theme.Sherlock.Light">
        <item name="android:actionBarStyle">@style/MyActionBar</item>
        <item name="actionBarStyle">@style/MyActionBar</item>
    </style>

    <style name="MyActionBar" parent="@style/Widget.Sherlock.Light.ActionBar.Solid">
        <item name="android:background">@drawable/actionbar_background</item>
        <item name="background">@drawable/actionbar_background</item>
    </style>

</resources>
```

[Code 10] You can customize ActionBarSherlock's themes easily by using the parenting mechanism of the styles.

Since ActionBarSherlock uses native and fallback approaches to providing themes it requires overriding style items twice. By providing the prefix android: (e.g. android:background) we declare that this particular item will be used by the native theme mechanism, while an empty namespace (e.g. background) means that the element will be used by the fallback solution. If you ever see that on one platform your theme looks fine, while on the other you can't see your changes, it may mean you need to revisit the above guidelines again.




[Image 7] Customized ActionBar on Android 2.1 (Éclair – left) and 4.2 (Jelly Bean – right) in landscape mode.

## 5. Summary and tips

Below you can find a list of some common guidelines regarding ActionBarSherlock that can help you

develop high quality code.

- •Using improper imports is one of the main causes of errors related to using the Support Library and the ActionBarSherlock. That is why you should pay special attention when using their classes and methods for the first time. It is advised to check if there are corresponding support-prefixed methods and use them instead of the native ones.
- •In order to minimize the chances of errors you should test your application on other platform versions during the development process. It can prove crucial in keeping your deadlines if you can quickly determine the cause of problems.
- •In case of Sherlock's themes customization you should remember to declare your style overrides twice – one for the native implementation (resource attribute names with @android: prefixes) and one for the fallback mechanism from the ActionBarSherlock.
- •Since ActionBarSherlock is an open source library and you are supposed to use it as an Android library project, you have easy access to its source code and resource declarations. Whenever you have some doubts regarding, for instance, the logic behind propagation of options item click events or customizing themes you can always check the inner mechanism of the library. This is especially important in case of customizing Sherlock themes when you can look into library's resources and find the list of potential elements you can override.

## 6. References

You can find more information about ActionBarSherlock at http://actionbarsherlock.com. It is also strongly advised to look into the official Android documentation at http://d.android.com regarding Fragments and the ActionBar if you have any doubts using them.

Ref:http://developer.samsung.com/android/technical-docs/UI-unification-with-older-Android-versions-using-ActionBarSherlock