

# The Real Best Practices to Save/Restore Activity's and Fragment's state. (StatedFragment is now deprecated)

Months ago I published an article related to Fragment State saving & restoring, [Probably be the best way \(?\) to save/restore Android Fragment's state so far](#). A lot of valuable feedback are received from Android developers all over the world. Thanks a ton to you all =)

Anyway `StatedFragment` causes a pattern breaking since it was designed to do the different way as Android is designed with an assumption that it might be easier for Android developer to understand Fragment's state saving/restoring if it acts just like Activity does (View's state and Instance state are handled at the same time). So I did an experiment by developed `StatedFragment` and see how is it going. Is it easier to understand? Is its pattern is more developer-friendly?

Right now, after 2 months of experiment, I think I got a result already. Although `StatedFragment` is a little bit easier to understand but it also comes with a pretty big problem. It breaks a pattern design of Android's View architecture. So I think it may causes a long time problem which is totally not good. Actually I also feel weird with my codes myself already...

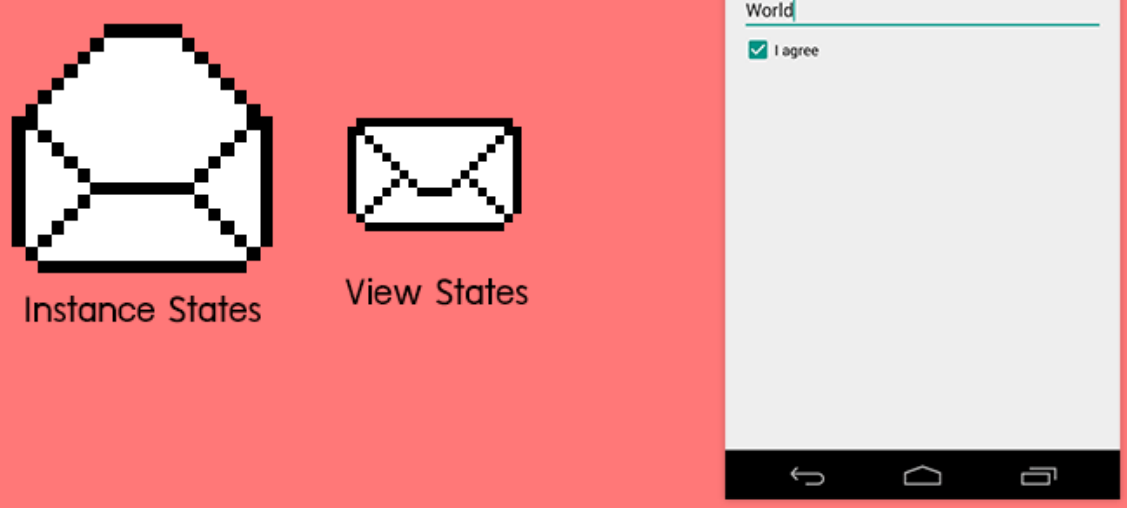
With this reason, **I decide to mark `StatedFragment` as deprecated from now on**. And as an apology for the mistake, I wrote this blog to show the real best practices visually how to save and restore Fragment's state in the way Android is designed. =)

## **Understand what happens while Activity's State is being Saved/Restored**

When Activity's `onSaveInstanceState` is called. Activity will automatically collect View's State from every single View in the View hierarchy. Please note that only View that is implemented View State Saving/Restoring internally that could be collected the data from. Once `onRestoreInstanceState` is called. Activity will send those collected data back to the View in the View hierarchy that provides the same `android:id` as it is collected from one by one.

Let's see it in visualization.

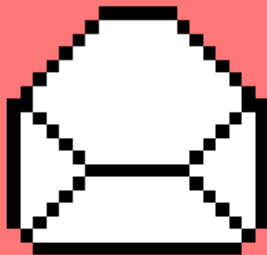
## Activity State Saving



The diagram illustrates the state saving process. On the left, a large envelope icon is labeled "Instance States", and a smaller envelope icon is labeled "View States". On the right, a screenshot of an Android application titled "Fragment State" is shown. The app's interface includes a text field containing the text "World" and a checked checkbox labeled "I agree". The screenshot also shows the Android navigation bar at the bottom and the system status bar at the top with the time 12:16.

# Activity State Restoring

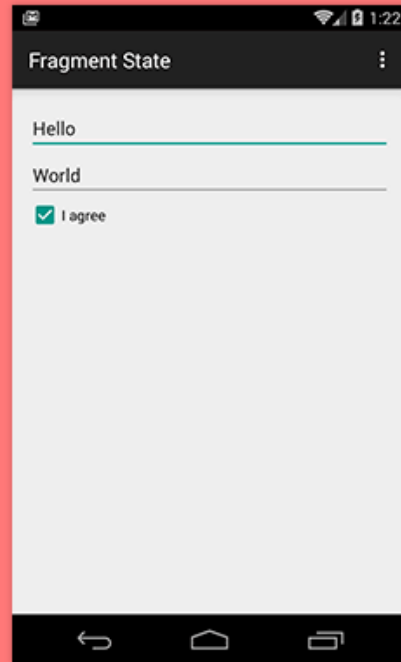
Return to View with same android:id



Instance States



View States



This is the reason why text typed inside EditText still persisted even though Activity is already destroyed and we didn't do anything special. There is no magic. Those View State are automatically collected and restored back.

And this is also the reason why those View without `android:id` defined isn't able to restore its View's state.

Although those View's state are automatically saved but the Activity's member variables are not. They will be destroyed along with Activity. You have to manually save and restore them through `onSaveInstanceState` and `onRestoreInstanceState` method.

```
public class MainActivity extends AppCompatActivity {  
  
    private int someVarA;  
    private String someVarB;  
  
    ...  
}
```

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("someVarA", someVarA);
    outState.putString("someVarB", someVarB);
}

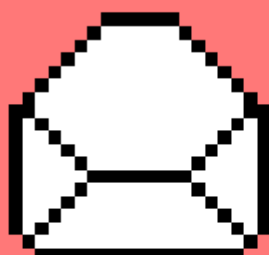
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    someVarA = savedInstanceState.getInt("someVarA");
    someVarB = savedInstanceState.getString("someVarB");
}
}
```

That's all what you have to do to restore Activity's Instance state and View state.

## **Understand what happens while Fragment's State is being Saved/Restored**

In case that Fragment is destroyed by the system. Everything will just happen exactly the same as Activity.

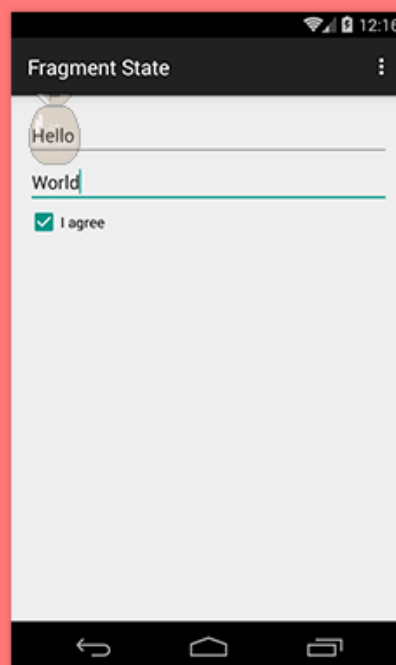
## Fragment State Saving



Instance States

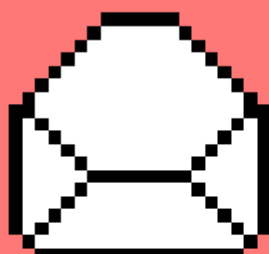


View States



## Fragment State Restoring

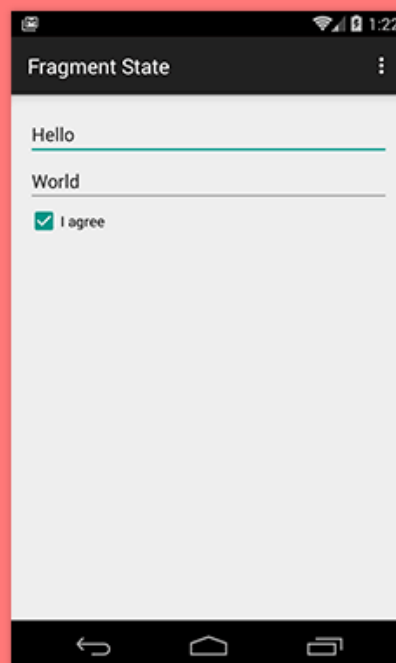
Return to View with same android:id



Instance States



View States



It means that every single member variables are also destroyed. You have to manually save and restore those variables

through `onSaveInstanceState` and `onActivityCreated` method respectively. Please note that there is no `onRestoreInstanceState` method inside `Fragment`.

```
public class MainFragment extends Fragment {

    private int someVarA;
    private String someVarB;

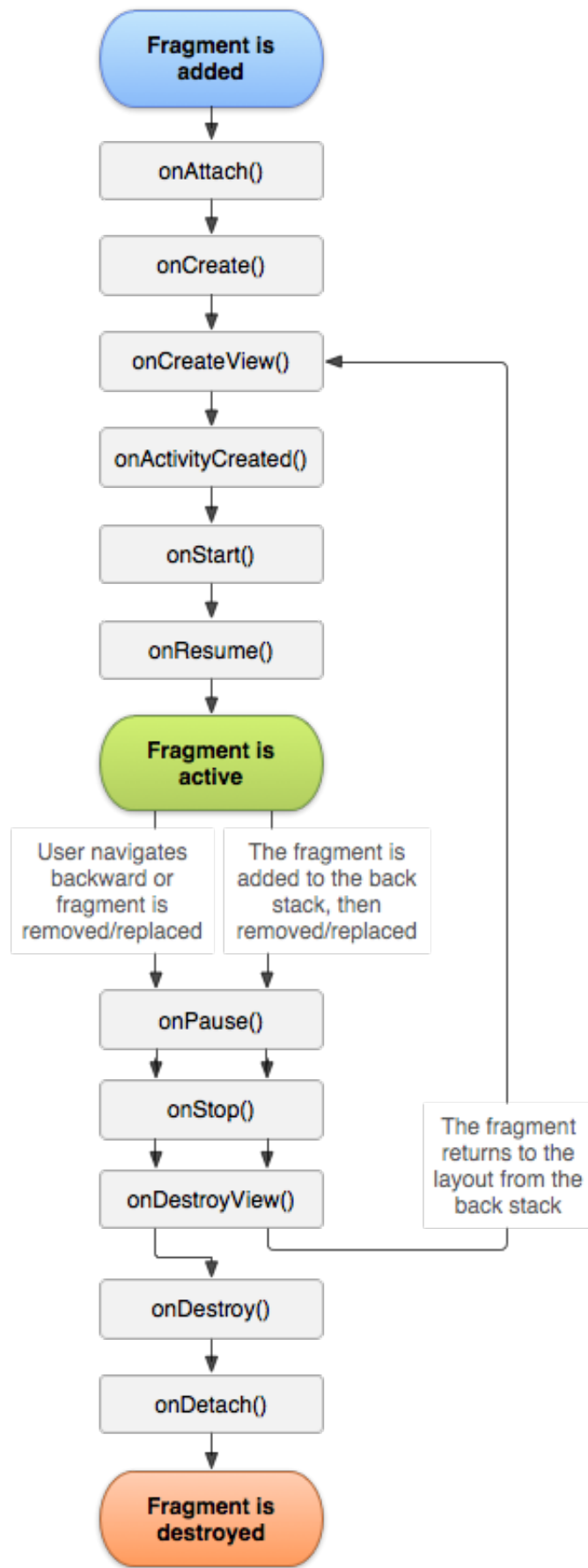
    ...

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putInt("someVarA", someVarA);
        outState.putString("someVarB", someVarB);
    }

    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        someVarA = savedInstanceState.getInt("someVarA");
        someVarB = savedInstanceState.getString("someVarB");
    }

}
```

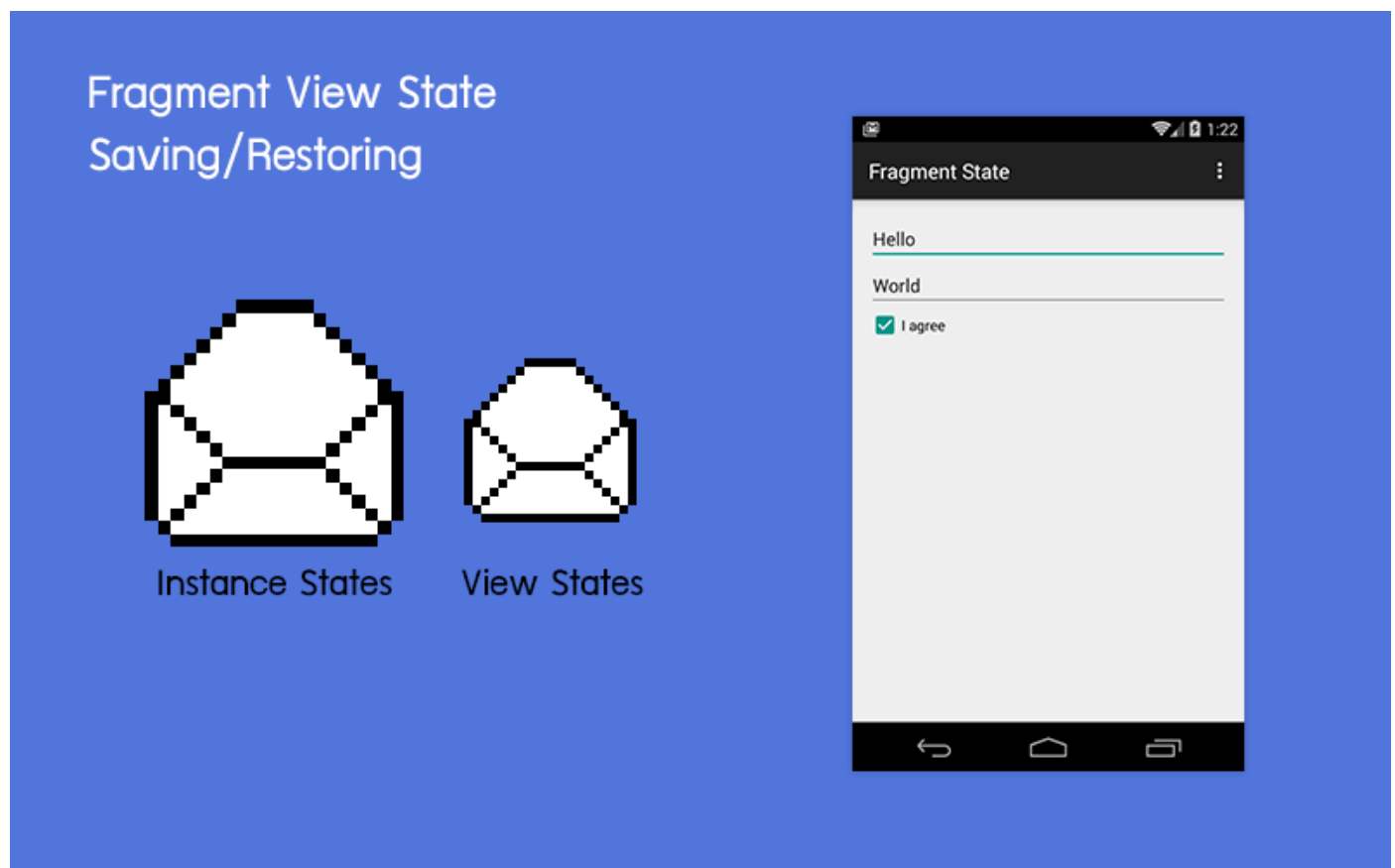
For `Fragment`, there is some special case that is different from `Activity` and I think that you need to know about it. **Once `Fragment` is returned from backstack, its `View` would be destroyed and recreated.**



**In this case, Fragment is not destroyed. Only View inside**

**Fragment does.** As a result, there is no any Instance State saving happens. But what happens to those View that is newly created by Fragment's lifecycle showed above?

Not a problem. Android is designed this way. View State Saving/Restoring are internally called inside Fragment in this case. As a result, every single View that is implemented a View State Saving/Restoring internally, for example `EditText` or `TextView` with `android:freezeText="true"`, will be automatically saved and restored the state. Causes it to display just perfectly the same as previous.



Please note that only View is destroyed (and recreated) in this case. Fragment is still there, just like those member variables inside. So you don't have to do anything with them. No any additional code is required.

```
public class MainFragment extends Fragment {
```



```
private int someVarA;
private String someVarB;

...

}
```

You might already notice that if every single View used in this Fragment are internally implemented a View Saving/Restoring. You have no need to do anything in this case since View's state will be automatically restored and member variables inside Fragment also still persist.

So the first condition of Fragment's State Saving/Restoring Best Practices is

...

## **Every single View used in your application must be internally implemented State Saving/Restoring**

Android provides a mechanic to View to save and restore View State internally through

`onSaveInstanceState` and `onRestoreInstanceState` method. It is developer's task to implement it.

```
public class CustomView extends View {

    ...

    @Override
    public Parcelable onSaveInstanceState() {
        Bundle bundle = new Bundle();

        return bundle;
    }

    @Override
    public void onRestoreInstanceState(Parcelable state) {
        super.onRestoreInstanceState(state);
```

```
}  
  
...  
  
}
```

Basically every single standard View such as EditText, TextView, Checkbox and etc. are all already internally implemented those things. Anyway you may need to enable it for some View for example you have to set `android:freezeText` to true for `TextView` to use the feature.

But if we talk about 3rd Party Custom View distributed all over the internet. I must say that many of them aren't implemented this part of code yet which may cause a big problem in real use.

If you decide to use any of 3rd Party Custom View, you have to be sure that it is already implemented View State Saving/Restoring internally or you have to create a subclass derived from that Custom View and implement `onSaveInstanceState/onRestoreInstanceState` yourself.

```
public class SomeBetterSmartButton extends SomeSmartButton {  
  
    ...  
  
    @Override  
    public Parcelable onSaveInstanceState() {  
        Bundle bundle = new Bundle();  
  
        return bundle;  
    }  
  
    @Override  
    public void onRestoreInstanceState(Parcelable state) {  
        super.onRestoreInstanceState(state);  
    }  
}
```

```
}  
  
...  
  
}
```

And if you create your own Custom View or Custom Viewgroup, don't forget to implement those two methods as well. It is really important that every single type of View used in the application is implemented this part.

And also don't forget to assign `android:id` attribute to every single View placed in the layout that you need to enable View State Saving and Restoring or it will not be able to restore the state at all.

```
<EditText  
    android:id="@+id/editText1"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />  
  
<EditText  
    android:id="@+id/editText2"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />  
  
<CheckBox  
    android:id="@+id/cbAgree"  
    android:text="I agree"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

We are now halfway there!

## **Clearly separate Fragment State from View State**

To make your code be clean and scalable, you have to separate Fragment State and View State from each other. If any property is belonged to View, do the state saving/restoring inside View. If any property is belonged to

Fragment, do it inside Fragment. Here is an example:

```
public class MainFragment extends Fragment {  
  
    ...  
  
    private String dataGotFromServer;  
  
    @Override  
    public void onSaveInstanceState(Bundle outState) {  
        super.onSaveInstanceState(outState);  
        outState.putString("dataGotFromServer", dataGotFromServer);  
    }  
  
    @Override  
    public void onActivityCreated(Bundle savedInstanceState) {  
        super.onActivityCreated(savedInstanceState);  
        dataGotFromServer = savedInstanceState.getString("dataGotFromServer");  
    }  
  
    ...  
  
}
```

Let me repeat again. Don't save View's State inside Fragment's onSaveInstanceState and vice versa.

That's all. It is the Best Practices on how to Save/Restore Activity's, Fragment's and View's State. Hope you find this piece of information useful =)

## **Goodbye StatedFragment, say Hi to NestedActivityResultFragment**

Please do the way described above to Save/Restore Activity's, Fragment's and View's State. And let me mark StatedFragment as **deprecated** now.

However StatedFragment's functionality to retrieve onActivityResult in

Nested Fragment is still good to go. To prevent any confusion in the future, I decide to separate that functionality to a new class


`NestedActivityResultFragment` available from v0.10.0 onwards.

More information about it is now

available at <https://github.com/nuuneoi/StatedFragment>. Please feel free to check it anytime !

Hope that the visualization in this blog helps you understand about the way to restore Activity's, Fragment's and View's State clearly. So sorry for the confusion in the previous article. ^^"

Author: **nuuneoi** (Android GDE, CTO & CEO at The Cheese Factory)



A full-stack developer with more than 6 years experience on Android Application Development and more than 12 years in Mobile Application Development industry. Also has skill in Infrastructure, Service Side, Design, UI&UX, Hardware, Optimization, Cooking, Photographing, Blogging, Training, Public Speaking and do love to share things to people in the world!