Saving Simple Application Data

There are two lightweight techniques for saving simple application data for Android applications — Shared Preferences and a pair of event handlers used for saving Activity instance details. Both mechanisms use a name/value pair (NVP) mechanism to store simple primitive values.

Using SharedPreferences, you can create named maps of key/value pairs within your application that can be shared between application components running in the same Context.

Shared Preferences support the primitive types Boolean, string, fl oat, long, and integer, making them an ideal way to quickly store default values, class instance variables, the current UI state, and user preferences. They are most commonly used to persist data across user sessions and to share settings between application components.

Alternatively, Activities offer the onSaveInstanceState handler. It's designed specifi cally to persist the UI state when the Activity becomes eligible for termination by a resource-hungry run time.

The handler works like the Shared Preference mechanism. It offers a Bundle parameter that represents a key/value map of primitive types that can be used to save the Activity's instance values. This Bundle is then made available as a parameter passed in to the onCreate and onRestoreInstanceState method handlers.

This UI state Bundle is used to record the values needed for an Activity to provide an identical UI following unexpected restarts.

Creating and Saving Preferences

To create or modify a Shared Preference, call getSharedPreferences on the application Context, passing in the name of the Shared Preferences to change. Shared Preferences are shared across an application's components but aren't available to other applications.

To modify a Shared Preference, use the SharedPreferences.Editor class. Get the Editor object by calling edit on the Shared Preferences object you want to change. To save edits, call commit on the Editor, as shown in the code snippet below.

```
public static final String MYPREFS = "mySharedPreferences";
protected void savePreferences(){
// Create or retrieve the shared preference object.
int mode = Activity.MODE_PRIVATE;
SharedPreferences mySharedPreferences = getSharedPreferences(MYPREFS, mode);
// Retrieve an editor to modify the shared preferences.
SharedPreferences.Editor editor = mySharedPreferences.edit();
// Store new primitive types in the shared preferences object.
editor.putBoolean("isTrue", true);
editor.putFloat("lastFloat", 1f);
editor.putInt("wholeNumber", 2);
editor.putLong("aNumber", 3l);
editor.putString("textEntryValue", "Not Empty");
// Commit the changes.
editor.commit();
}
```

Retrieving Shared Preferences

Accessing saved Shared Preferences is also done with the getSharedPreferences method. Pass in the name of the Shared Preference you want to access, and use the type-safe get<type> methods to extract
saved values.

Each getter takes a key and a default value (used when no value is available for that key), as shown in the skeleton code below:

```
public void loadPreferences() {
// Get the stored preferences
int mode = Activity.MODE_PRIVATE;
```

```
SharedPreferences mySharedPreferences = getSharedPreferences(MYPREFS,
mode);
// Retrieve the saved values.
boolean isTrue = mySharedPreferences.getBoolean("isTrue", false);
float lastFloat = mySharedPreferences.getFloat("lastFloat", 0f);
int wholeNumber = mySharedPreferences.getInt("wholeNumber", 1);
long aNumber = mySharedPreferences.getLong("aNumber", 0);
String stringPreference;
stringPreference = mySharedPreferences.getString("textEntryValue",
"");
}
```

Saving the Activity State

If you want to save Activity information that doesn't need to be shared with other components (e.g., class instance variables), you can call Activity.getPreferences() without specifying a preferences name. Access to the Shared Preferences map returned is restricted to the calling Activity; each Activity supports a single unnamed SharedPreferences object.

The following skeleton code shows how to use the Activity's private Shared Preferences:

```
protected void saveActivityPreferences(){
// Create or retrieve the activity preferences object.
SharedPreferences activityPreferences =
getPreferences(Activity.MODE_PRIVATE);
// Retrieve an editor to modify the shared preferences.
SharedPreferences.Editor editor = activityPreferences.edit();
// Retrieve the View
TextView myTextView = (TextView)findViewById(R.id.myTextView);
// Store new primitive types in the shared preferences object.
editor.putString("currentTextValue",
myTextView.getText().toString());
// Commit changes.
editor.commit();
}
```

Saving and Restoring Instance State

To save Activity instance variables, Android offers a specialized alternative to Shared Preferences. By overriding an Activity's onSaveInstanceState event handler, you can use its Bundle parameter to save instance values. Store values using the same get and put methods as shown for Shared Preferences, before passing the modifi ed Bundle into the superclass's handler, as shown in the following code snippet:

```
private static final String TEXTVIEW_STATE_KEY = "TEXTVIEW_STATE_KEY";

@Override
public void onSaveInstanceState(Bundle outState) {
// Retrieve the View
TextView myTextView = (TextView)findViewById(R.id.myTextView);
// Save its state
outState.putString(TEXTVIEW_STATE_KEY,
myTextView.getText().toString());
super.onSaveInstanceState(outState);
}
```

This handler will be triggered whenever an Activity completes its Active life cycle, but only when it's not being explicitly fi nished. As a result, it's used to ensure a consistent Activity state between active life cycles of a single user session.

The saved Bundle is passed in to the onRestoreInstanceState and onCreate methods if the application is forced to restart during a session. The following snippet shows how to extract values from the Bundle and use them to update the Activity instance state:

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

```
setContentView(R.layout.main);
TextView myTextView = (TextView)findViewById(R.id.myTextView);
String text = "";
if (icicle != null && icicle.containsKey(TEXTVIEW_STATE_KEY))
text = icicle.getString(TEXTVIEW_STATE_KEY);
myTextView.setText(text);
}
```

It's important to remember that onSaveInstanceState *is called only when an Activity becomes inactive, but not when it is being closed by a call to fi nish or by the user pressing the Back button.*

Saving the To-Do List Activity State

Currently, each time the To-Do List example application is restarted, all the to-do items are lost and any text entered into the text entry box is cleared. In this example, you'll start to save the application state of the To-Do list application across sessions.

The instance state in the ToDoList Activity consists of three variables:

- ❑ Is a new item being added?
- ❑ What text exists in the new item entry textbox?
- ❑ What is the currently selected item?

Using the Activity's default Shared Preference, you can store each of these values and update the UI when the Activity is restarted.

Later in this chapter, you'll learn how to use the SQLite database to persist the to-do items as well. This example is a fi rst step that shows how to ensure a seamless experience by saving Activity instance details.

1. Start by adding static String variables to use as preference keys.

```
private static final String TEXT_ENTRY_KEY = "TEXT_ENTRY_KEY";
private static final String ADDING_ITEM_KEY = "ADDING_ITEM_KEY";
private static final String SELECTED_INDEX_KEY = "SELECTED_INDEX_KEY";
```

2. Next, override the onPause method. Get the Activity's private Shared Preference object, and get its Editor object. Using the keys you created in Step 1, store the instance values based on whether a new item is being added and any text in the "new item" Edit Box.

```
@Override
protected void onPause(){
super.onPause();
// Get the activity preferences object.
SharedPreferences uiState = getPreferences(0);
// Get the preferences editor.
SharedPreferences.Editor editor = uiState.edit();
// Add the UI state preference values.
editor.putString(TEXT_ENTRY_KEY, myEditText.getText().toString());
editor.putBoolean(ADDING_ITEM_KEY, addingNew);
// Commit the preferences.
editor.commit();
}
```

3. Write a restoreUIState method that applies the instance values you recorded in Step 2 when the application restarts. Modify the onCreate method to add a call to the restoreUIState method at the very end.

```
@Override
public void onCreate(Bundle icicle) {
[ ... existing onCreate logic ... ]
restoreUIState();
}
private void restoreUIState() {
// Get the activity preferences object.
SharedPreferences settings = getPreferences(Activity.MODE_PRIVATE);
// Read the UI state values, specifying default values.
String text = settings.getString(TEXT_ENTRY_KEY, "");
Boolean adding = settings.getBoolean(ADDING_ITEM_KEY, false);
// Restore the UI to the previous state.
```

```
if (adding) {
addNewItem();
myEditText.setText(text);
}
}
```

4. Record the index of the selected item using the onSaveInstanceState / onRestore InstanceState mechanism. It's then only saved and applied if the application is killed without the user's explicit instruction.

```
@Override
public void onSaveInstanceState(Bundle outState) {
outState.putInt(SELECTED_INDEX_KEY,
myListView.getSelectedItemPosition());
super.onSaveInstanceState(outState);
}
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
int pos = -1;
if (savedInstanceState != null)
if (savedInstanceState.containsKey(SELECTED_INDEX_KEY))
pos = savedInstanceState.getInt(SELECTED_INDEX_KEY, -1);
myListView.setSelection(pos);
}
```

When you run the To-Do List application, you should now see the UI state persisted across sessions. That said, it still won't persist the to-do list items — you'll add this essential piece of functionality later in the chapter.