Using Adapters for Data Binding

To apply an Adapter to an AdapterView-derived class, you call the View's setAdapter method, passing in an Adapter instance, as shown in the snippet below:

```
ArrayList<String> myStringArray = new ArrayList<String>();
ArrayAdapter<String> myAdapterInstance;

int layoutID = android.R.layout.simple_list_item_1;
myAdapterInstance = new ArrayAdapter<String>(this, layoutID, myStringArray);
myListView.setAdapter(myAdapterInstance);
```

This snippet shows the most simplistic case, where the array being bound is a string and the List View items are displayed using a single Text View control.

The first of the following examples demonstrates how to bind an array of complex objects to a List View using a custom layout. The second shows how to use a Simple Cursor Adapter to bind a query result to a custom layout within a List View.

Customizing the To-Do List ArrayAdapter

This example extends the To-Do List project, storing each item as a ToDoItem object that includes the date each item was created.

You will extend ArrayAdapter to bind a collection of ToDoItem objects to the ListView and customize the layout used to display each List View item.

1. Return to the To-Do List project. Create a new ToDoItem class that stores the task and its creation date. Override the toString method to return a summary of the item data.

```
package com.paad.todolist;
import java.text.SimpleDateFormat;
import java.util.Date;
public class ToDoItem {
String task;
Date created;
public String getTask() {
return task;
}
public Date getCreated() {
return created;
}
public ToDoItem(String _task) {
this(_task, new Date(java.lang.System.currentTimeMillis()));
}
public ToDoItem(String _task, Date _created) {
task = _task;
created = _created;
}
@Override
public String toString() {
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yy");
String dateString = sdf.format(created);
return "(" + dateString + ") " + task;
}
}
```

2. Open the ToDoList Activity, and modify the ArrayList and ArrayAdapter variable types to store ToDoItem objects rather than Strings. You'll then need to modify the onCreate method to update the corresponding variable initialization. You'll also need to update the onKeyListener handler to support the ToDoItem objects.

```
private ArrayList<ToDoItem> todoItems;
private ListView myListView;
private EditText myEditText;
private ArrayAdapter<ToDoItem> aa;
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
// Inflate your view
setContentView(R.layout.main);
// Get references to UI widgets
myListView = (ListView)findViewById(R.id.myListView);
myEditText = (EditText)findViewById(R.id.myEditText);
todoItems = new ArrayList<ToDoItem>();
int resID = R.layout.todolist_item;
aa = new ArrayAdapter<ToDoItem>(this, resID, todoItems);
myListView.setAdapter(aa);
myEditText.setOnKeyListener(new OnKeyListener() {
public boolean onKey(View v, int keyCode, KeyEvent event) {
if (event.getAction() == KeyEvent.ACTION_DOWN)
if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
ToDoItem newItem;
newItem = new ToDoItem(myEditText.getText().toString());
todoItems.add(0, newItem);
myEditText.setText("");
aa.notifyDataSetChanged();
cancelAdd();
return true;
}
return false;
}
});
registerForContextMenu(myListView);
}
```

3. If you run the Activity, it will now display each to-do item, as shown in Figure 5-3.

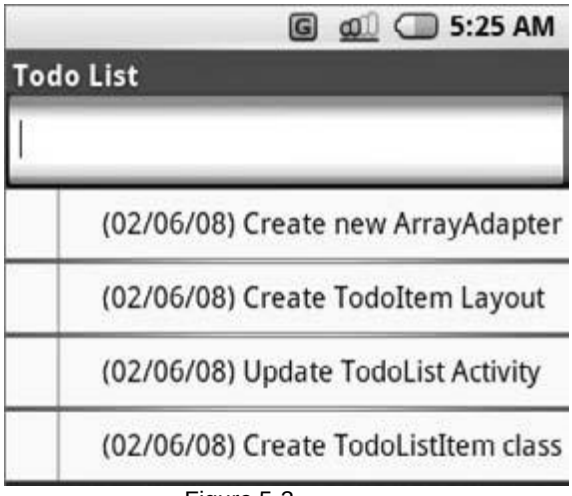

Figure 5-3

4. Now you can create a custom layout to display each to-do item. Start by modifying the custom layout you created in Chapter 4 to include a second TextView. It will be used to show the creation date of each to-do item.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
```

```
android:layout_height=”fill_parent”
android:background=”@color/notepad_paper”>
<TextView
android:id=”@+id/rowDate”
android:layout_width=”wrap_content”
android:layout_height=”fill_parent”
android:padding=”10dp”
android:scrollbars=”vertical”
android:fadingEdge=”vertical”
android:textColor=”@color/notepad_text”
android:layout_alignParentRight=”true”
/>
<TextView
android:id=”@+id/row”
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
android:padding=”10dp”
android:scrollbars=”vertical”
android:fadingEdge=”vertical”
android:textColor=”@color/notepad_text”
android:layout_alignParentLeft=”@+id/rowDate”
/>
</RelativeLayout>
```

5. Create a new class (ToDoItemAdapter) that extends an ArrayAdapter with a ToDoItem-specifi c variation. Override getView to assign the task and date properties in the ToDoItem object to the Views in the layout you created in Step 4.

```
import java.text.SimpleDateFormat;
import android.content.Context;
import java.util.*;

import android.view.*;
import android.widget.*;
public class ToDoItemAdapter extends ArrayAdapter<ToDoItem> {
int resource;
public ToDoItemAdapter(Context _context,
int _resource,
List<ToDoItem> _items) {
super(_context, _resource, _items);
resource = _resource;
}
@Override
public View getView(int position, View convertView, ViewGroup parent)
{
LinearLayout todoView;
ToDoItem item = getItem(position);
String taskString = item.getTask();
Date createdDate = item.getCreated();
SimpleDateFormat sdf = new SimpleDateFormat(“dd/MM/yy”);
String dateString = sdf.format(createdDate);
if (convertView == null) {
todoView = new LinearLayout(getContext());
String inflater = Context.LAYOUT_INFLATER_SERVICE;
LayoutInflater vi;
vi = (LayoutInflater)getContext().getSystemService(inflater);
vi.inflate(resource, todoView, true);
} else {
todoView = (LinearLayout) convertView;
}
TextView dateView = (TextView)todoView.findViewById(R.id.rowDate);
TextView taskView = (TextView)todoView.findViewById(R.id.row);
dateView.setText(dateString);
taskView.setText(taskString);
return todoView;
}
}
```

6. Finally, replace the ArrayAdapter declaration with a ToDoItemAdapter. private ToDoItemAdapter aa;
Within onCreate, replace the ArrayAdapter<String> instantiation with the new ToDoItemAdapter.
aa = new ToDoItemAdapter(this, resID, todoItems);

7. If you run your Activity, it should appear as shown in the screenshot in Figure 5-4.

Figure 5-4

Using the SimpleCursorAdapter

The SimpleCursorAdapter lets you bind columns from a Cursor to a List View using a custom layout defi nition.

The SimpleCursorAdapter is constructed by passing in the current context, a layout resource, a Cursor, and two arrays: one that contains the names of the columns to be used and a second (equally sized) array that has resource IDs for the Views to use to display the corresponding column's data value.

The following skeleton code shows how to construct a SimpleCursorAdapter to display contact information:

```
String uriString = "content://contacts/people/";
Cursor myCursor = managedQuery(Uri.parse(uriString), null, null, null, null);
String[] fromColumns = new String[] {People.NUMBER, People.NAME};
int[] toLayoutIDs = new int[] { R.id.nameTextView, R.id.numberTextView};
SimpleCursorAdapter myAdapter;
myAdapter = new SimpleCursorAdapter(this,
R.layout.simplecursorlayout,
myCursor,
fromColumns,
toLayoutIDs);
myListView.setAdapter(myAdapter);
```

The Simple Cursor Adapter was used previously in this chapter when creating the Contact Picker example. You'll learn more about Content Providers and Cursors in Chapter 6, where you'll also find more SimpleCursorAdapter examples.