Part 2 - Using the Contacts ContentProvider

Writing code to use access data exposed by a ContentProvider doesn’t require a reference to the ContentProvider class at all. Instead a Uri is used to create a cursor over the data exposed by the ContentProvider. Android uses the Uri to search the system for the application that has exposed a ContentProvider with that identifier. The Uri is a string, typically in a reverse-DNS format such as this com.android.contacts/data.

Rather than having to remember this string, the Android *Contacts* provider exposes its metadata in the android.provider.ContactsContract class. This class is used to determine both the Uri of the ContentProvider and also the names of the tables and columns that can be queried.

Some data types also require special permission to access. The built-in contacts list requires the android.permission.READ_CONTACTS permission in the AndroidManifest.xml file.

There are three ways to create a cursor from the Uri:

ManagedQuery() – The preferred approach in Android 2.3 (API Level 10) and earlier, a ManagedQuery returns a cursor and also automatically manages refreshing the data and closing the cursor. This method is deprecated in Android 3.0 (API Level 11).

ContentResolver.Query() – Returns an unmanaged cursor, which means it must be refreshed and closed explicitly in code.

CursorLoader().LoadInBackground() – Introduced in Android 3.0 (API Level 11), CursorLoader is now the preferred way to consume a ContentProvider. CursorLoader queries a ContentResolver on a background thread so the UI isn’t blocked. This class can be accessed in older versions of Android using the v4 compatibility library.

Each of these methods has the same basic set of inputs:

Uri – The fully qualified name of the ContentProvider.

Projection – Specification of which columns to select for the cursor.

Selection – Similar to a SQL WHERE clause.

SelectionArgs – Parameters to be substituted in the Selection.

SortOrder – Columns to sort by.

Creating Inputs for a Query

The ContactsProvider sample code performs a very simple query against Android’s built-in Contacts provider. You do not need to know the actual Uri or column names, all the information required to query the Contacts ContentProvider is available as constants exposed by the ContactsContract class.

Regardless of which method is used to retrieve the cursor, these same objects are used as parameters as shown in the ContactsProvider/ContactsAdapter.cs file:

```
var uri = ContactsContract.Contacts.ContentUri;
string[] projection = {
   ContactsContract.Contacts.InterfaceConsts.Id,
   ContactsContract.Contacts.InterfaceConsts.DisplayName,

ContactsContract.Contacts.InterfaceConsts.PhotoId,

};
```

For this example, the selection, selectionArgs and sortOrder will be ignored by setting them to null.

Creating a Cursor from a Content Provider Uri

Once the parameter objects have been created, they can be used in one of the following three ways:

Using a Managed Query

Applications targeting Android 2.3 (API Level 10) or earlier should use this method:

var cursor = activity.ManagedQuery(uri, projection, null, null, null);

This cursor will be managed by Android so you do not need to close it.

Using ContentResolver

Accessing ContentResolver directly to get a cursor against a ContentProvider can be done like this:

```
var cursor = activity.ContentResolver(uri, projection, null, null, null);
```

This cursor is unmanaged, so it must be closed when no longer required. Ensure that the code closes a cursor that is open, otherwise an error will occur.

```
cursor.Close();
```

Alternatively you can call StartManagingCursor() and StopManagingCursor() to ‘manage’ the cursor. Managed cursors are automatically deactivated and re-queried when activities are stopped and restarted.

Using CursorLoader

Applications built for Android 3.0 (API Level 11) or newer should use this method:

```
var loader = new CursorLoader (activity, uri, projection, null, null, null);
var cursor = (ICursor)loader.LoadInBackground();
```

The CursorLoader ensures that all cursor operations are done on a background thread, and can intelligently re-use an existing cursor across activity instances when an activity is restarted (eg. due to a configuration change) rather that reload the data again.

Earlier Android versions can also use the CursorLoader class by using the v4 support libraries [doc ref?].

Displaying the Cursor Data with a Custom Adapter

To display the contact image we'll use a custom adapter, so that we can manually resolve the PhotoId reference to an image file path.

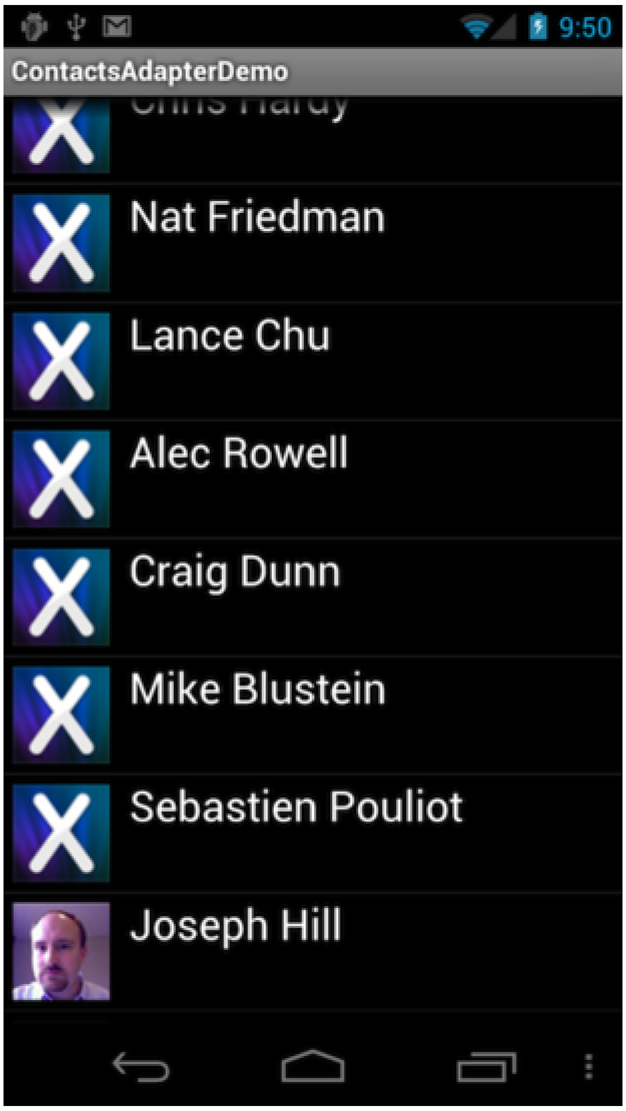
To display data with a custom adapter, the example uses a CursorLoader to retrieve all the Contact data into a local collection in the FillContacts method from ContactsProvider/ContactsAdapter.cs:

```
void FillContacts ()
{
   var uri = ContactsContract.Contacts.ContentUri;
   string[] projection = {
       ContactsContract.Contacts.InterfaceConsts.Id,
       ContactsContract.Contacts.InterfaceConsts.DisplayName,
       ContactsContract.Contacts.InterfaceConsts.PhotoId
  };
   // CursorLoader introduced in Honeycomb (3.0, API11)
   var loader = new CursorLoader(activity, uri, projection, null, null, null);
   var cursor = (ICursor)loader.LoadInBackground();
   contactList = new List<Contact> ();
   if (cursor.MoveToFirst ()) {
      do {
          contactList.Add (new Contact{
              Id = cursor.GetLong (cursor.GetColumnIndex (projection [0])),
              DisplayName = cursor.GetString (cursor.GetColumnIndex (projection [1])),
              PhotoId = cursor.GetString (cursor.GetColumnIndex (projection [2]))
          });
       } while (cursor.MoveToNext());
   }
}
```

Then implement the BaseAdapter's methods using the contactList collection. The adapter is implemented just as it would be with any other collection – there is no 'special handling' here because the data is sourced from a ContentProvider:

```
Activity activity;
public ContactsAdapter (Activity activity)
{
   this.activity = activity;
   FillContacts ();
}
public override int Count {
   get { return contactList.Count; }
}
public override Java.Lang.Object GetItem (int position)
{
  return null; // could wrap a Contact in a Java.Lang.Object to return it here if needed
}
public override long GetItemId (int position)
{
   return contactList [position].Id;
}
public override View GetView (int position, View convertView, ViewGroup parent)
{
   var view = convertView ?? activity.LayoutInflater.Inflate (Resource.Layout.ContactListItem, parent, false);
   var contactName = view.FindViewById<TextView> (Resource.Id.ContactName);
   var contactImage = view.FindViewById<ImageView> (Resource.Id.ContactImage);
   contactName.Text = contactList [position].DisplayName;
   if (contactList [position].PhotoId == null) {
       contactImage = view.FindViewById<ImageView> (Resource.Id.ContactImage);
       contactImage.SetImageResource (Resource.Drawable.ContactImage);
   } else {
       var contactUri = ContentUris.WithAppendedId (ContactsContract.Contacts.ContentUri, contactList [position].
Id);
       var contactPhotoUri = Android.Net.Uri.WithAppendedPath (contactUri, Contacts.Photos.ContentDirectory);
       contactImage.SetImageURI (contactPhotoUri);
   }
   return view;
}
```

The image is displayed (if it exists) using the Uri to the image file on the device. The application looks like this:



9:50

ContactsAdapterDemo

Chris Hardy

Nat Friedman

Lance Chu

Alec Rowell

Craig Dunn

Mike Blustein

Sebastien Pouliot

Joseph Hill

Using a similar code pattern your application can access a wide variety of system data including the user’s photos, videos and music. Some data types require special permissions to be requested in the project’s Properties (which are stored in AndroidManifest.xml).

Displaying the Cursor Data with a SimpleCursorAdapter

The cursor could also be displayed with a SimpleCursorAdapter (although only the name will be displayed, not the photo). This code demonstrates how to use a ContentProvider with SimpleCursorAdapter (this code does not appear in the sample):

```
var uri = ContactsContract.Contacts.ContentUri;
string[] projection = {
   ContactsContract.Contacts.InterfaceConsts.Id,
   ContactsContract.Contacts.InterfaceConsts.DisplayName
};
var loader = new CursorLoader (this, uri, projection, null, null, null);
var cursor = (ICursor)loader.LoadInBackground();
var fromColumns = new string[] {ContactsContract.Contacts.InterfaceConsts.DisplayName};
var toControlIds = new int[] {Android.Resource.Id.Text1};
adapter = new SimpleCursorAdapter (this, Android.Resource.Layout.SimpleListItem1, cursor, fromColumns,
toControlsIds);
listView.Adapter = adapter;
```

Refer toListViews and Adapters for further information on implementing SimpleCursorAdapter.

Next: Part 3 - Creating a Custom ContentProvider

Source URL:
http://docs.xamarin.com/guides/android/platform_features/intro_to_content_providers/part_2_-_using_the_contacts_contentprovier