Service:

<http://developer.android.com/guide/components/services.html>

1. Create a new "service" package and a new [IntentService](http://developer.android.com/reference/android/app/IntentService.html" \t "_blank) called SunshineService.
2. Declare the SunshineService in the AndroidManifest.xml.<service android:name=".service.SunshineService"/>
3. Move the code that fetches weather data from FetchWeatherTask'sdoInBackground into your new SunshineService's onHandleIntentmethod.
4. Initiate the service using an explicit intent within the updateWeather method in ForecastFragment.

Reference this [Guide on Services](http://developer.android.com/guide/components/services.html).

To solve this quiz, after creating the new service package and SunshineService class:

1. Copy over the code from FetchWeatherTask’s doInBackgroundmethod to the SunshineService's onHandleIntent.
2. In SunshineService, add a Log Tag and a String constant for the Intent Extra that will be added to the Intent that starts SunshineService. This Intent Extra will be used to pass in the location.
3. public static final String LOCATION\_QUERY\_EXTRA = "lqe";
4. private final String LOG\_TAG = SunshineService.class.getSimpleName();
5. Clean up onHandleIntent by replacing return null; withreturn void;. Note that Services have their own context, so there is no need to set or pass in a context to SunshineService. Instead, just use this.
6. Create your service and start it with an explicit intent. This should be done in the updateWeather method in ForecastFragment:
7. Intent intent = new Intent(getActivity(), SunshineService.class);
8. intent.putExtra(SunshineService.LOCATION\_QUERY\_EXTRA,
9. Utility.getPreferredLocation(getActivity()));
10. getActivity().startService(intent);
11. Don't forget to get the location from the intent before you execute the rest of the code in onHandleIntent:String locationQuery = intent.getStringExtra(LOCATION\_QUERY\_EXTRA);

Alarms

1. Add a public static inner class AlarmReceiver which extendsBroadcastReceiver to SunshineService. It should overrideonReceive. The onReceive method will get triggered when a Broadcast Intent is fired.
2. Since this is a new component, register it in the AndroidManifest:

<receiver android:name=".service.SunshineService$AlarmReceiver" android:enabled="true"/>

Your goal is to set the AlarmManager so that it fires the code inside of AlarmReceiver. Your code should be placed in updateWeather and it should:

* Start the code in AlarmReceiver five seconds after updateWeather is called (for testing purposes this time is short).
* It should only fire once.
* It should wake up the phone if it is asleep.

Hint 1: The AlarmManager fires PendingIntents. PendingIntents are wrappers for explicit Intents. You can use an explicit intent to trigger AlarmReceiver. Your goal here is to set up the explicit intent, wrap it in a PendingIntent, and set the constraints under which your PendingIntent should be fired with an AlarmManager.

Hint 2: PendingIntents are not created with constructors, rather, as stated in the [documentation](http://developer.android.com/reference/android/app/PendingIntent.html) “Instances of this class are created with getActivity(Context, int, Intent, int), getActivities(Context, int, Intent[], int), getBroadcast(Context, int, Intent, int), and getService(Context, int, Intent, int)”.

For more information:

[Set a Repeating Alarm](https://developer.android.com/training/scheduling/alarms.html#set)

[PendingIntent](http://developer.android.com/reference/android/app/PendingIntent.html)

[Lesson 3 : Broadcast Intents](https://www.udacity.com/course/viewer#!/c-ud853/l-1474559101/m-1631278613)

[BroadcastReceiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html)

Transferring Data Without Draining the Battery

<http://developer.android.com/training/efficient-downloads/index.html>

[DevBytes: Efficient Data Transfers](https://www.youtube.com/watch?v=cSIB2pDvH3E&list=PLWz5rJ2EKKc-VJS9WQlj9xM_ygPopZ-Qd)

# Optimizing Downloads for Efficient Network Access

<http://developer.android.com/training/efficient-downloads/efficient-network-access.html>

# Minimizing the Effect of Regular Updates

<http://developer.android.com/training/efficient-downloads/regular_updates.html>

# Redundant Downloads are Redundant

<http://developer.android.com/training/efficient-downloads/redundant_redundant.html>

# Modifying your Download Patterns Based on the Connectivity Type

<http://developer.android.com/training/efficient-downloads/connectivity_patterns.html>

# Transferring Data Using Sync Adapters

<http://developer.android.com/training/sync-adapters/index.html>

# T Sync Adapters

hroughout this course you’ve been learning about different ways to handle Sunshine’s need to do data processing and network calls off of the main thread. For the network call to the Open Weather Map API, the best solution is to use a SyncAdapter. Why? Well SyncAdapters are meant to keep local data on the device in sync with data on the web. This is exactly what we’re doing with Open Weather Map. We’re storing a local cache of weather data from Open Weather Map, with the goal of fast load times, offline functionality and not necessitating a network call every time we load an activity.

Additionally, using a Sync Adapter will intelligently maintain a balance between keeping our local weather data cache up to date and saving the battery by being smart about when to place network calls.

Many of the normal use cases for sync adapters involve syncing the app's local data with information from an online account. For example, in an email app, a sync adapter might be used to pull down a user’s emails at regular intervals. Of course to do this the email app would need to store the user’s account information, such as a username and password, so that the app could log in and grab the new messages. To manage this, sync adapters each have a concept of a user’s account, tied with the sync adapter.

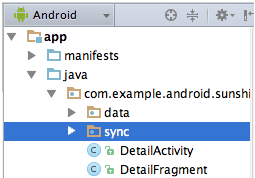
Does Open Weather Map need a user account? No. But we still need to implement the classes to handle accounts.

Does Open Weather Map need a user account? No. But we still need to implement the classes to handle accounts.

### Instructions

**1. Copy over the sync package**

You can copy this package from [here](https://github.com/udacity/Sunshine-Version-2/tree/lesson_6_sync_adapter_starter_code). Put it on the same level as your data package. This package includes the following classes:



* SunshineAuthenticator - This extends AbstractAccountAuthenticator. This code manages authenticatingwith Sunshine’s background service (as in Open Weather Map). Since Open Weather Map needs no authentication, this is literally the stub code from the [Developer Guide on sync adapters](http://developer.android.com/training/sync-adapters/creating-authenticator.html).
* SunshineAuthenticatorService - This will allow the framework to access our SunshineAuthenticator. This is also almost exactly the same as the stub code from the [Developer Guide on sync adapters](http://developer.android.com/training/sync-adapters/creating-authenticator.html).
* SunshineSyncService - This service provides framework access to your SunshineSyncAdapter. This too is almost exactly the same as the stub code from the [Developer Guide on sync adapters](http://developer.android.com/training/sync-adapters/creating-sync-adapter.html).
* SunshineSyncAdapter - This is the actual sync adapter; it extends AbstractThreadedSyncAdapter. Note the following methods:
  + onPerformSync - This is the important method which is overridden when you subclass AbstractThreadedSyncAdapter. This is what happens when a sync occurs. In our case, we’ll eventually want to move the network code here.
  + getSyncAccount() - This is a helper method we’ve coded for you. Whenever you request a sync, you need a sync account. This is in case requesting the sync requires a login (in our situation, the Open Weather API doesn’t, but we still have our dummy account). This method gives us one of our dummy accounts.
  + syncImmediately() - This is a helper method we’ve coded for you. It tells the system to perform a sync with our sync adapter immediately.

**2. Add the following two xml files to the res/xml folder**

You can copy these files from [here](https://github.com/udacity/Sunshine-Version-2/tree/lesson_6_sync_adapter_starter_code/xml). This includes:

* authenticator.xml - This resource file plugs your authenticator into your sync adapter and account frameworks by providing it with some of the meta data it needs.
* syncadapter.xml - This resource file defines the settings associated with your SyncAdapter.

**3. Update your strings.xml and AndroidManifest**

* Add the following [gist](https://gist.github.com/udacityandroid/6248a0d3c9e75a565992) to the strings.xml.

**Note: Make sure all references to content\_authority match the string you define here.** The string sync\_account\_type suggests that the account is specific to our SunshineApp. The content\_authority string is the Uri authority for your content provider.

We’re going to make four changes to our AndroidManifest.xml.

* Change 1 - Add our new SunshineAuthenticator and SunshineAuthenticatorService classes to the AndroidManifest.xml.

[AndroidManifest.xml Gist -- see file 1](https://gist.github.com/udacityandroid/335d2fcf012d8e59c6fd)

* Change 2 - We’re going to attach our WeatherProvider to the SyncAdapter. Change the provider tag for WeatherProvider in the AndroidManifest to the code in the Gist below.

Code to **replace** provider tag in AndroidManifest.xml:

[AndroidManifest.xml Gist -- see file 2](https://gist.github.com/udacityandroid/335d2fcf012d8e59c6fd)

android:exported="false" means that only our app can see our content provider.  
android:syncable="true" mean that this content provider will be synced with a server.  
Note how we now use the strings.xml content\_authority to signify our content\_authority.

* Change 3 - Add additional permissions at the start of AndroidManifest.xml.

[AndroidManifest.xml Gist -- see file 3](https://gist.github.com/udacityandroid/335d2fcf012d8e59c6fd)

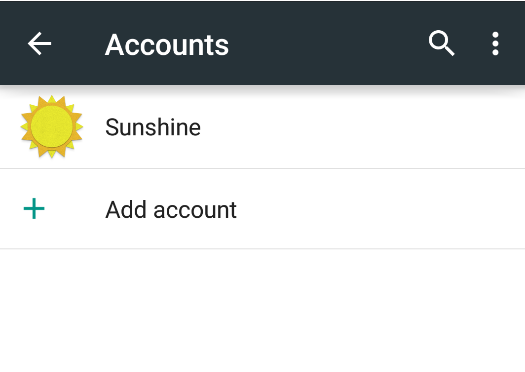
* Change 4 - Finally, register the SunshineSyncService in the AndroidManifest.xml.

[AndroidManifest.xml Gist -- see file 4](https://gist.github.com/udacityandroid/335d2fcf012d8e59c6fd)

**4. Test that it works**

At this point you can test if everything is working properly:

1. In the SunshineSyncAdapter’s onPerformSync method, add a Log statement.
2. Comment out other code in the updateWeather function in ForecastFragment and replace it with a call to the syncImmediately helper function.
3. Run your app and verify that the Log statement in onPerformSync prints to the console when you Refresh.
4. You can also go into your phone’s Settings and see that the dummy Sunshine account is now there.



You can also check your changes using this [diff](https://github.com/udacity/Sunshine-Version-2/compare/6.02_alarms...6.03_implement_sync_adapter).

**Goals**

* Make sure that updateWeather in ForecastFragment starts a sync.
* Move the synchronization code from onHandleIntent in SunshineService to onPerformSync and fix any errors.
* Make onPerformSync directly access the preference for location.

Hint : You already have a helper method in SunshineSyncAdapter that will run a sync.

# Running a Sync Adapter

<http://developer.android.com/training/sync-adapters/running-sync-adapter.html>

Basically you’ll need to copy the code from the SunshineService class intoonPerformSync. In doing so, you should grab the context by callinggetContext. The following code will get you the location:

String locationQuery = Utility.getPreferredLocation(getContext());

Finally, if you haven’t already, update updateWeather to syncImmediately.

## Scheduled Synchronization

[Diff for this node](https://github.com/udacity/Sunshine-Version-2/compare/6.04_finish_sync_adapter...6.05_scheduled_sync).

Here you will be creating the logic to, when you start the app, check if an account has been created for Sunshine and if not, create a new account and configure a periodic sync to start.

### Instructions

**1. Cleaning up the Code**

* If you haven’t already, delete FetchWeatherTask and the SunshineService. They are now just older, unused classes which do the same thing that the SunshineSyncAdapter class is doing.
* Clean up the AndroidManifest.xml and remove SunshineService and the AlarmReceiver from the xml.
* In onPreferenceChange in the SettingsActivity, call syncImmediately() instead of creating and executing a FetchWeatherTask.

**2. Add and update methods in SunshineSyncAdapter**

Add the following methods which can be found in [this gist.](https://gist.github.com/udacityandroid/7230489fb8cb3f46afee)

* configurePeriodicSync
* onAccountCreated
* initializeAdapter

Also add the line onAccountCreated() to the method getSyncAccount after you find that a new account has been created. You can copy and replace using the method from [this gist](https://gist.github.com/udacityandroid/e6a90828de84c8b439d9)

**3. Add Constants for Sync Time**

At the top of SunshineSyncAdapter add the following constants for sync time. Devices capable of doing flexible syncs will sync more often.

// Interval at which to sync with the weather, in milliseconds.

// 60 seconds (1 minute) \* 180 = 3 hours

public static final int SYNC\_INTERVAL = 60 \* 180;

public static final int SYNC\_FLEXTIME = SYNC\_INTERVAL/3;

**4. Start everything in MainActivity**

Go to MainActivity. At the end of MainActivity call: SunshineSyncAdapter.initializeSyncAdapter(this);

### The control flow

1. Your MainActivity is created and the sync adapter is initialized.
2. During initialization, getSyncAccount is called.
3. getSyncAccount will create a new account if no sunshine.example.com account exists. If this is the case,onAccountCreated will be called.
4. onAccountCreated configures the periodic sync and calls for an immediate sync. At this point, Sunshine will sync with the Open Weather API either every 3 hours (if the build version is less than KitKat) or everyone 1 hour (if the build version is greater than or equal to KitKat)

### Test if it Works

If you'd like to quickly test that your periodic sync code is working:

1. Change the SYNC\_INTERVAL time; set it to something low, for example, 30 seconds.
2. Add a Log statement in onPerformSync
3. Uninstall Sunshine or remove the Sunshine account.
4. Finally run Sunshine again and confirm syncs are being called. You will want to undo these changes and uninstall Sunshine again when done so that your phone is **not** syncing every 30 seconds.

Move onwards for a quick quiz...

Continue to Quiz