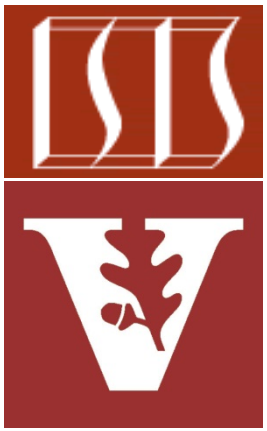
# Android Content Providers:

# Asynchronous Access to Content Providers

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



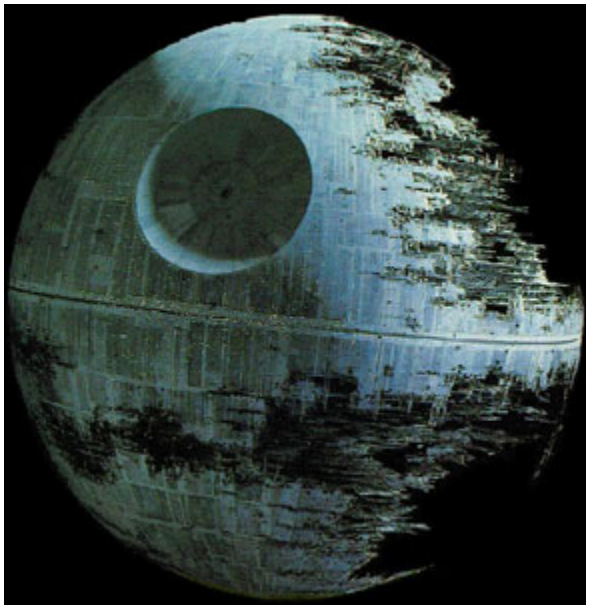
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

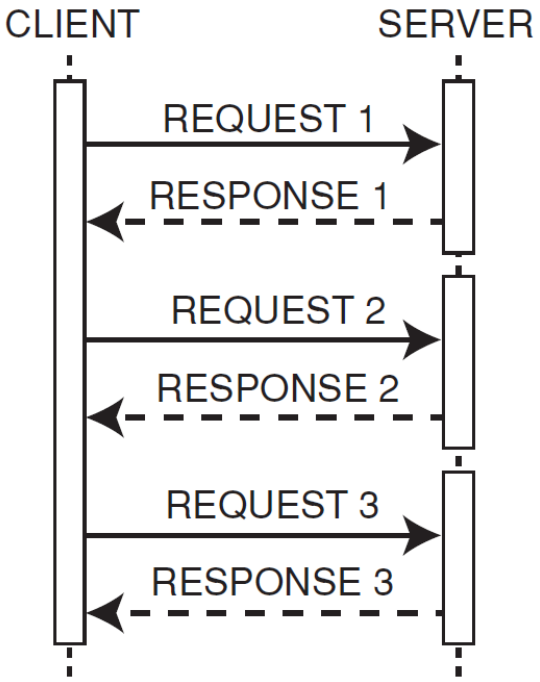
# Learning Objectives in this Part of the Module

- Understand the motivation for accessing Content Providers asynchronously

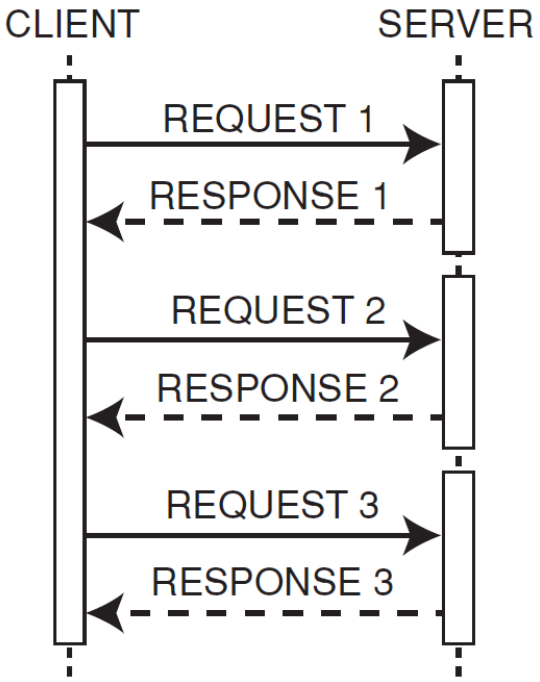
# Async Access to Content Providers

- All Activities thus far have invoked synchronous two-way calls to query ContentResolver/Provider
  - *Pros*: "Intuitive" since it maps nicely onto conventional "request/response method call interactions



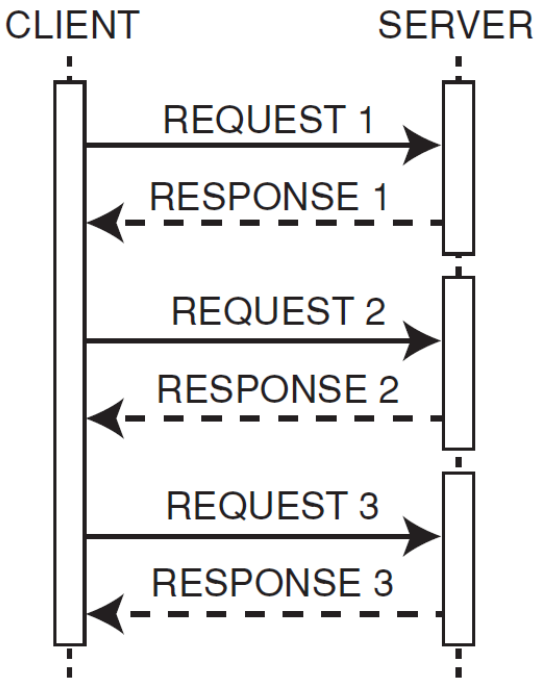
# Async Access to Content Providers

- All Activities thus far have invoked synchronous two-way calls to query ContentResolver/Provider
  - *Pros*: "Intuitive" since it maps nicely onto conventional "request/response method call interactions
  - *Cons*:
    - Doesn't leverage inherent parallelism in the system



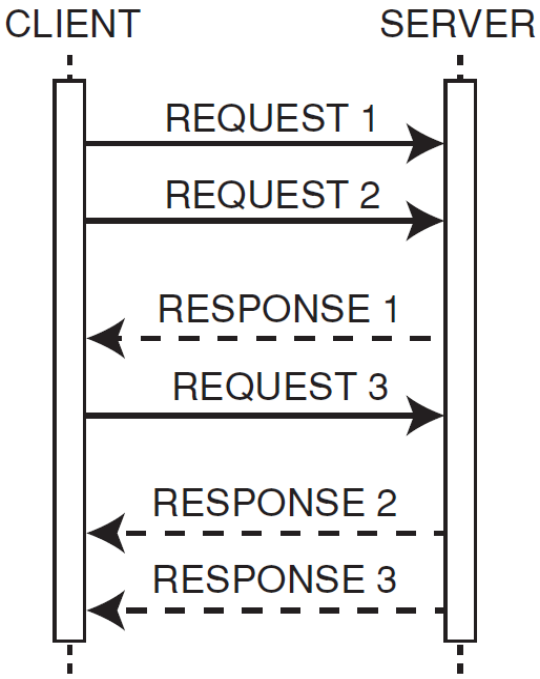
# Async Access to Content Providers

- All Activities thus far have invoked synchronous two-way calls to query ContentResolver/Provider
  - *Pros*: "Intuitive" since it maps nicely onto conventional "request/response method call interactions
  - *Cons*:
    - Doesn't leverage inherent parallelism in the system
    - Blocks the caller when performing queries on the UI Thread
      - Blocking is problematic for lengthy operations, such as loading data



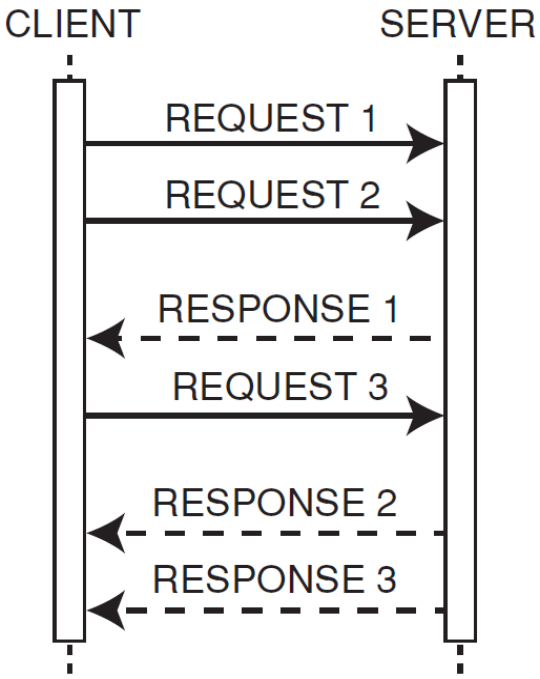
# Async Access to Content Providers

- All Activities thus far have invoked synchronous two-way calls to query ContentResolver/Provider
- An alternative on Android involves the use of two-way asynchronous operations
  - *Pros*: Leverages inherent parallelism more effectively & doesn't block the UI Thread



# Async Access to Content Providers

- All Activities thus far have invoked synchronous two-way calls to query ContentResolver/Provider
- An alternative on Android involves the use of two-way asynchronous operations
  - *Pros*: Leverages inherent parallelism more effectively & doesn't block the UI Thread
  - *Cons*: Can be hard to program unless you understand asynchrony patterns
    - e.g., *Proactor* & *Asynchronous Completion Token*



# Async Access to Content Providers

- All Activities thus far have invoked synchronous two-way calls to query ContentResolver/Provider
- An alternative on Android involves the use of two-way asynchronous operations
- Asynchronous Android models include
  - Use a CursorLoader to query the ContentResolver & return a Cursor

public class Summary: Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 11**

## CursorLoader

extends AsyncTaskLoader<D>

java.lang.Object
↳ android.content.Loader<D>
↳ android.content.AsyncTaskLoader<D>
↳ android.content.CursorLoader

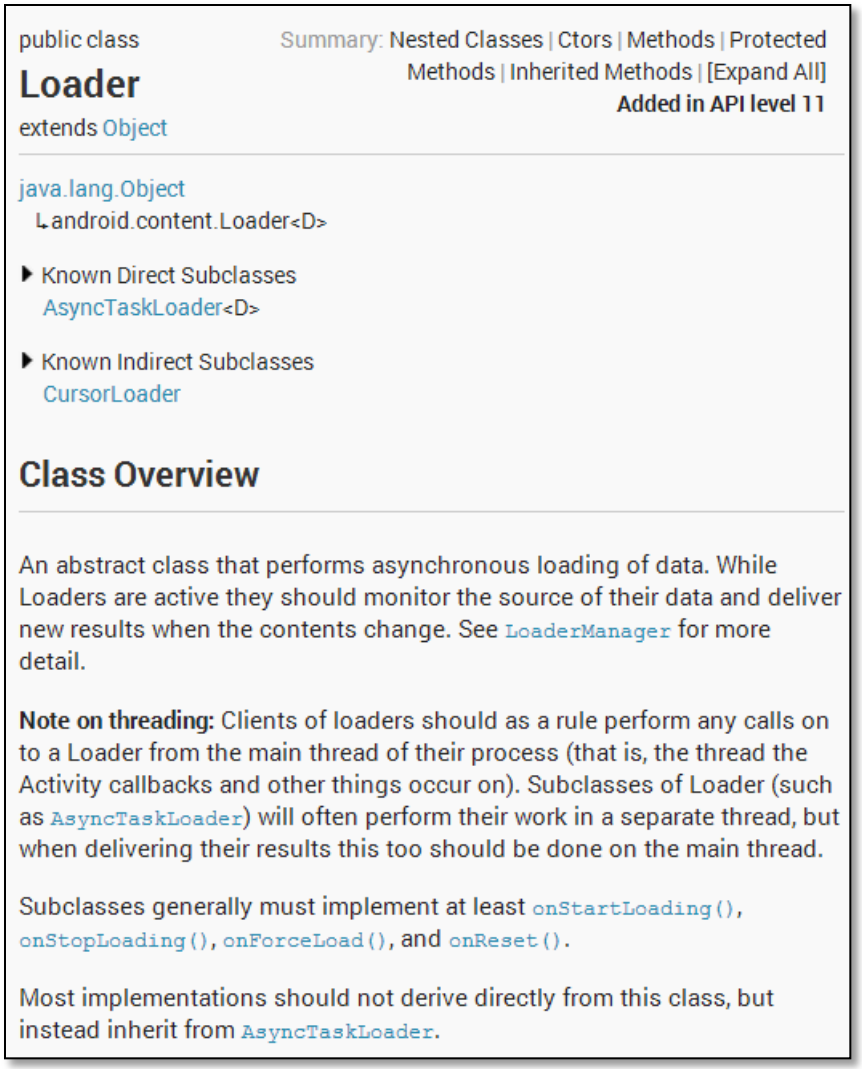
### Class Overview

A loader that queries the `ContentResolver` and returns a `Cursor`. This class implements the `Loader` protocol in a standard way for querying cursors, building on `AsyncTaskLoader` to perform the cursor query on a background thread so that it does not block the application's UI.

A CursorLoader must be built with the full information for the query to perform, either through the `CursorLoader(Context, Uri, String[], String, String[], String)` or creating an empty instance with `CursorLoader(Context)` and filling in the desired paramters with `setUri(Uri)`, `setSelection(String)`, `setSelectionArgs(String[])`, `setSortOrder(String)`, and `setProjection(String[])`.

# Async Access to Content Providers

- All Activities thus far have invoked synchronous two-way calls to query ContentResolver/Provider
- An alternative on Android involves the use of two-way asynchronous operations
- Asynchronous Android models include
  - Use a CursorLoader to query the ContentResolver & return a Cursor
    - Implements Loader protocol to query cursors & perform the cursor query on a background thread to not block the App's UI



public class
**Loader**
extends Object

Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]
**Added in API level 11**

java.lang.Object
↳android.content.Loader<D>

▸ Known Direct Subclasses
AsyncTaskLoader<D>

▸ Known Indirect Subclasses
CursorLoader

## Class Overview

An abstract class that performs asynchronous loading of data. While Loaders are active they should monitor the source of their data and deliver new results when the contents change. See `LoaderManager` for more detail.

**Note on threading:** Clients of loaders should as a rule perform any calls on to a Loader from the main thread of their process (that is, the thread the Activity callbacks and other things occur on). Subclasses of Loader (such as `AsyncTaskLoader`) will often perform their work in a separate thread, but when delivering their results this too should be done on the main thread.

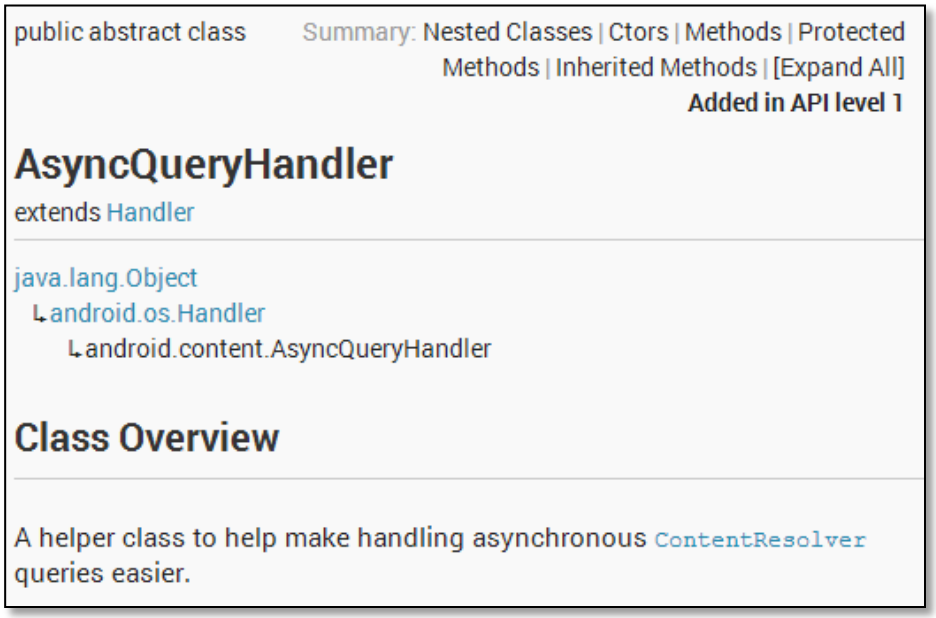
Subclasses generally must implement at least `onStartLoading()`, `onStopLoading()`, `onForceLoad()`, and `onReset()`.

Most implementations should not derive directly from this class, but instead inherit from `AsyncTaskLoader`.

developer.android.com/reference/android/content/Loader.html

# Async Access to Content Providers

- All Activities thus far have invoked synchronous two-way calls to query ContentResolver/Provider
- An alternative on Android involves the use of two-way asynchronous operations
- Asynchronous Android models include
  - Use a CursorLoader to query the ContentResolver & return a Cursor
  - Use an AsyncQueryHandler to make async ContentResolver queries easier



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

Added in API level 1

**AsyncQueryHandler**

extends Handler

java.lang.Object
↳ android.os.Handler
↳ android.content.AsyncQueryHandler

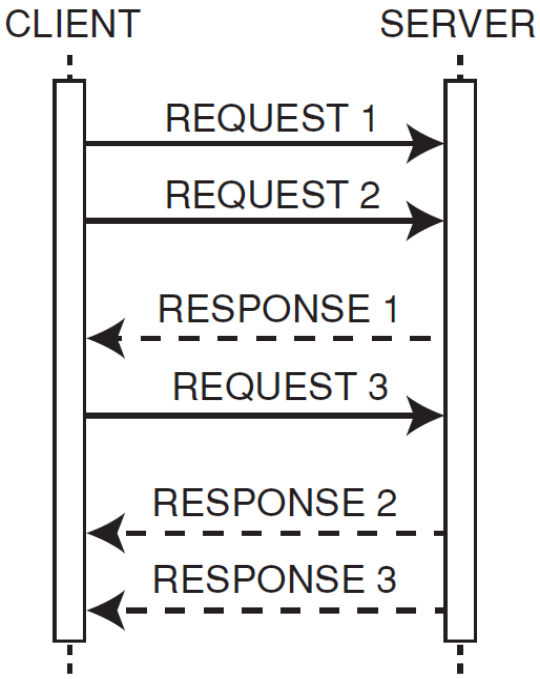
**Class Overview**

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html

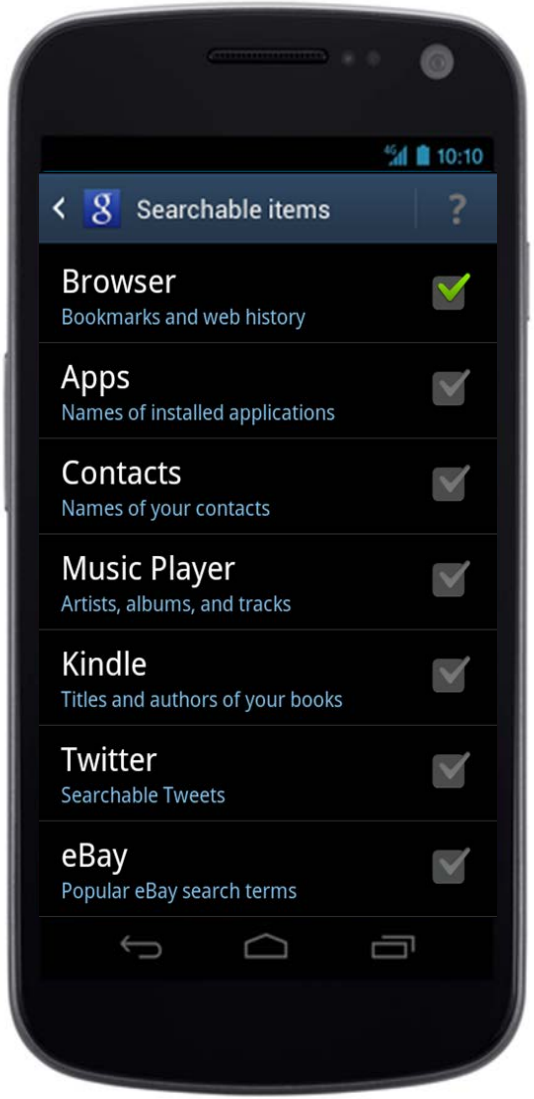
# Summary

- Asynchrony is a powerful technique that Android supports to optimize access to ContentProviders



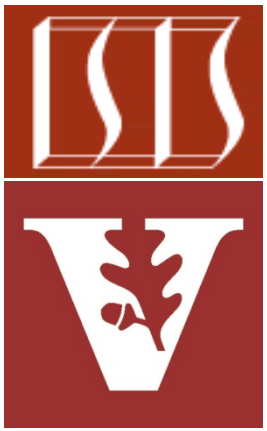
# Summary

- Asynchrony is a powerful technique that Android supports to optimize access to Content Providers
- Asynchronous access to Content Providers is very common in Android Apps, e.g.:
  - Browser
  - Calendar
  - Contacts
  - Email
  - MMS/SMS



# Android Content Providers: Programming with the LoaderManager

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

# Learning Objectives in this Part of the Module

- Understand how to access Content Providers asynchronously via the LoaderManager framework & CursorLoaders

# Overview of Loader

- An abstract class that performs asynchronous loading of data
  - Loaders ensure that all cursor operations are done “in the background”, thereby eliminating the possibility of blocking the UI Thread

public class
**Loader**
extends Object

Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]
**Added in API level 11**

java.lang.Object
↳android.content.Loader<D>

▸ Known Direct Subclasses
AsyncTaskLoader<D>

▸ Known Indirect Subclasses
CursorLoader

## Class Overview

An abstract class that performs asynchronous loading of data. While Loaders are active they should monitor the source of their data and deliver new results when the contents change. See `LoaderManager` for more detail.

**Note on threading:** Clients of loaders should as a rule perform any calls on to a Loader from the main thread of their process (that is, the thread the Activity callbacks and other things occur on). Subclasses of Loader (such as `AsyncTaskLoader`) will often perform their work in a separate thread, but when delivering their results this too should be done on the main thread.

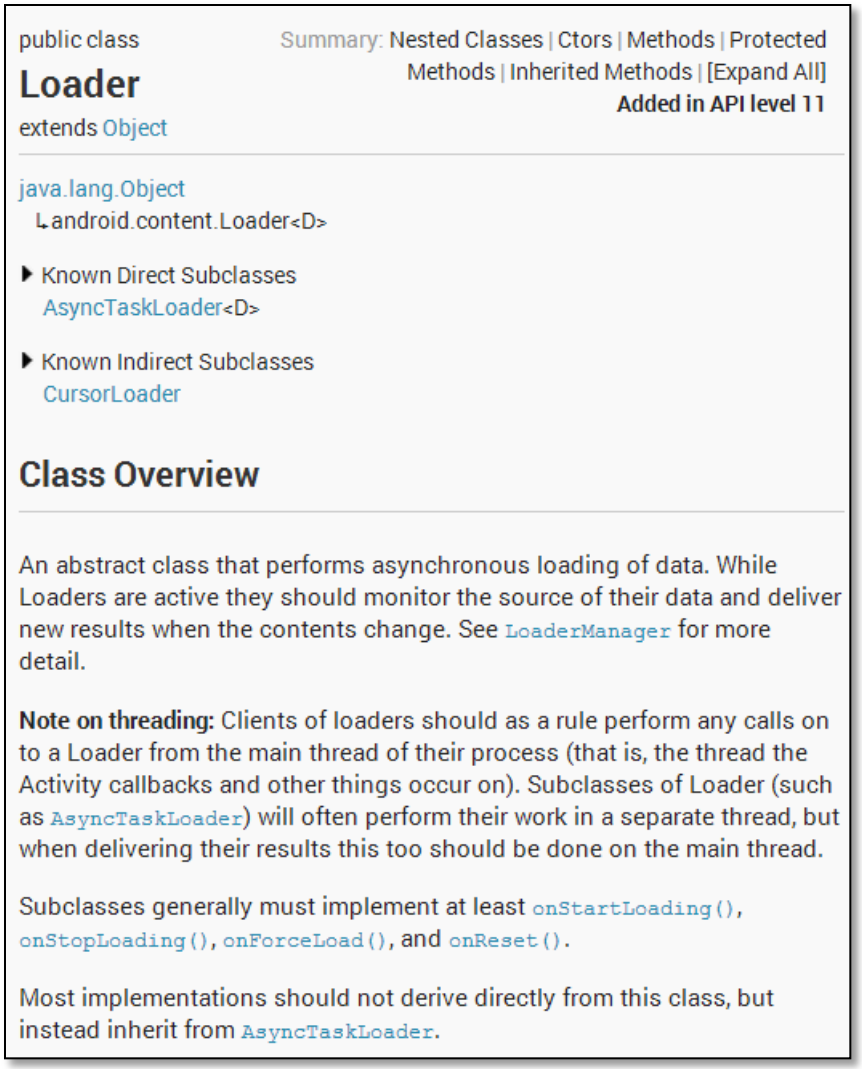
Subclasses generally must implement at least `onStartLoading()`, `onStopLoading()`, `onForceLoad()`, and `onReset()`.

Most implementations should not derive directly from this class, but instead inherit from `AsyncTaskLoader`.

developer.android.com/reference/android/content/Loader.html

# Overview of Loader

- An abstract class that performs asynchronous loading of data
  - Loaders ensure that all cursor operations are done “in the background”, thereby eliminating the possibility of blocking the UI Thread
  - While Loaders are active they monitor the source of their data & deliver new results when the contents change



public class
**Loader**
extends Object

Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]
**Added in API level 11**

java.lang.Object
↳android.content.Loader<D>

▸ Known Direct Subclasses
AsyncTaskLoader<D>

▸ Known Indirect Subclasses
CursorLoader

## Class Overview

An abstract class that performs asynchronous loading of data. While Loaders are active they should monitor the source of their data and deliver new results when the contents change. See `LoaderManager` for more detail.

**Note on threading:** Clients of loaders should as a rule perform any calls on to a Loader from the main thread of their process (that is, the thread the Activity callbacks and other things occur on). Subclasses of Loader (such as `AsyncTaskLoader`) will often perform their work in a separate thread, but when delivering their results this too should be done on the main thread.

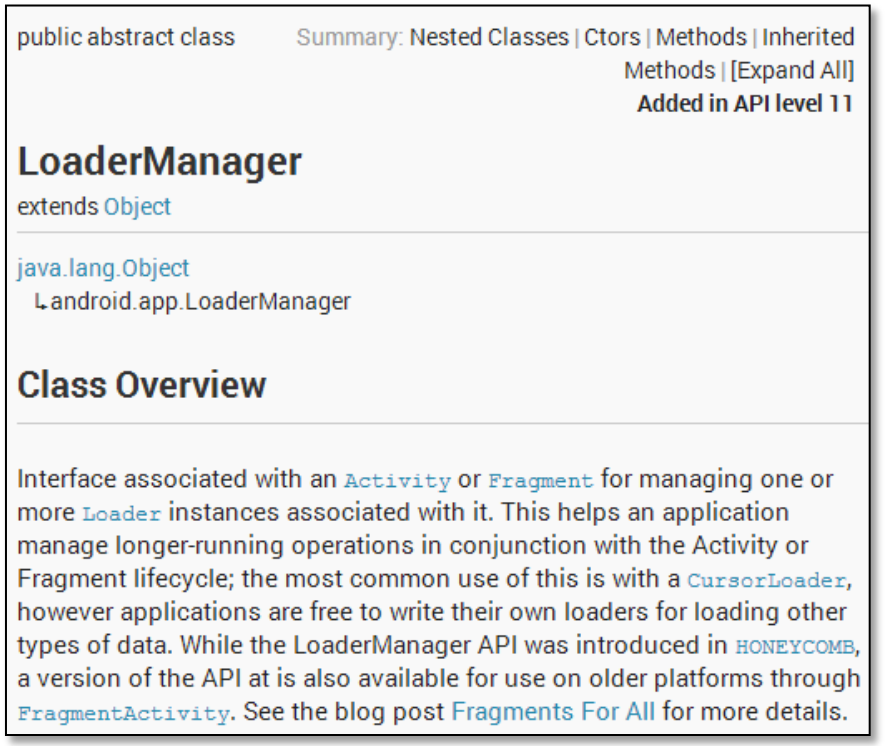
Subclasses generally must implement at least `onStartLoading()`, `onStopLoading()`, `onForceLoad()`, and `onReset()`.

Most implementations should not derive directly from this class, but instead inherit from `AsyncTaskLoader`.

developer.android.com/reference/android/content/Loader.html

# Overview of LoaderManager

- Interface associated with an Activity or Fragment for managing one or more Loader instances associated with it
  - e.g., a LoaderManager is in charge of starting, stopping, retaining, restarting, & destroying its Loaders



public abstract class Summary: Nested Classes | Ctors | Methods | Inherited Methods | [Expand All]

**Added in API level 11**

## LoaderManager

extends Object

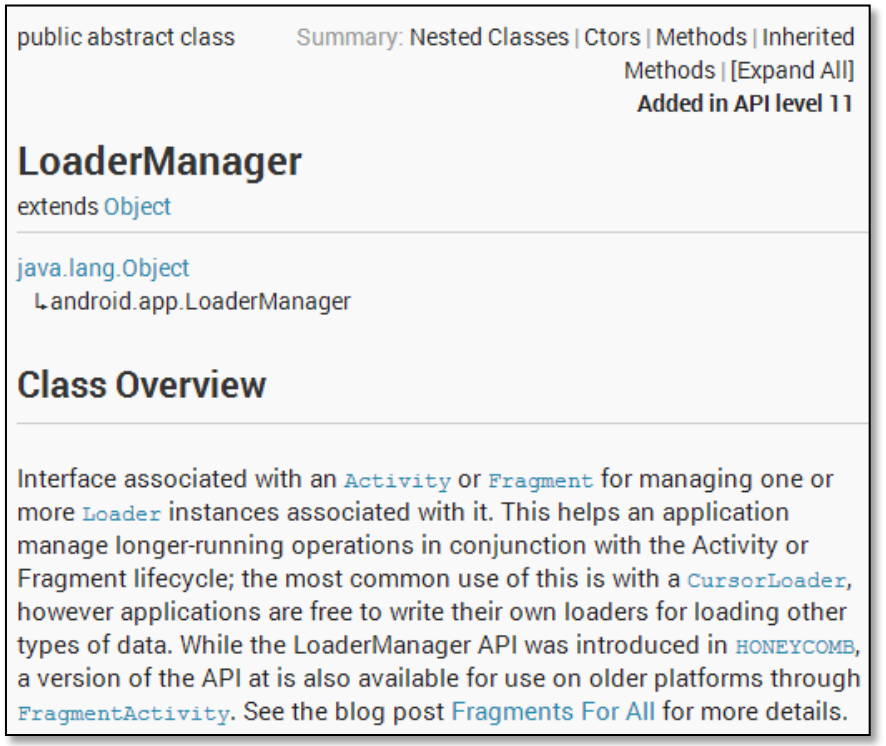
java.lang.Object
↳android.app.LoaderManager

### Class Overview

Interface associated with an `Activity` or `Fragment` for managing one or more `Loader` instances associated with it. This helps an application manage longer-running operations in conjunction with the Activity or Fragment lifecycle; the most common use of this is with a `CursorLoader`, however applications are free to write their own loaders for loading other types of data. While the LoaderManager API was introduced in `HONEYCOMB`, a version of the API at is also available for use on older platforms through `FragmentActivity`. See the blog post Fragments For All for more details.

# Overview of LoaderManager

- Interface associated with an Activity or Fragment for managing one or more Loader instances associated with it
- When a Loader is managed by a LoaderManager it retains its existing cursor data across the Activity or Fragment instance
  - e.g., when a restart occurs due to a configuration change the cursor needn't perform unnecessary, potentially expensive re-queries



public abstract class Summary: Nested Classes | Ctors | Methods | Inherited Methods | [Expand All]

**Added in API level 11**

## LoaderManager

extends Object

java.lang.Object
↳android.app.LoaderManager

## Class Overview

Interface associated with an `Activity` or `Fragment` for managing one or more `Loader` instances associated with it. This helps an application manage longer-running operations in conjunction with the Activity or Fragment lifecycle; the most common use of this is with a `CursorLoader`, however applications are free to write their own loaders for loading other types of data. While the LoaderManager API was introduced in `HONEYCOMB`, a version of the API at is also available for use on older platforms through `FragmentActivity`. See the blog post Fragments For All for more details.

# Overview of CursorLoader

- A CursorLoader runs an asynchronous query in the background against a ContentProvider
  - It returns the result to the Activity or Fragment from which it was called

public class Summary: Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]
**Added in API level 11**

## CursorLoader

extends AsyncTaskLoader<D>

java.lang.Object
↳ android.content.Loader<D>
↳ android.content.AsyncTaskLoader<D>
↳ android.content.CursorLoader

## Class Overview

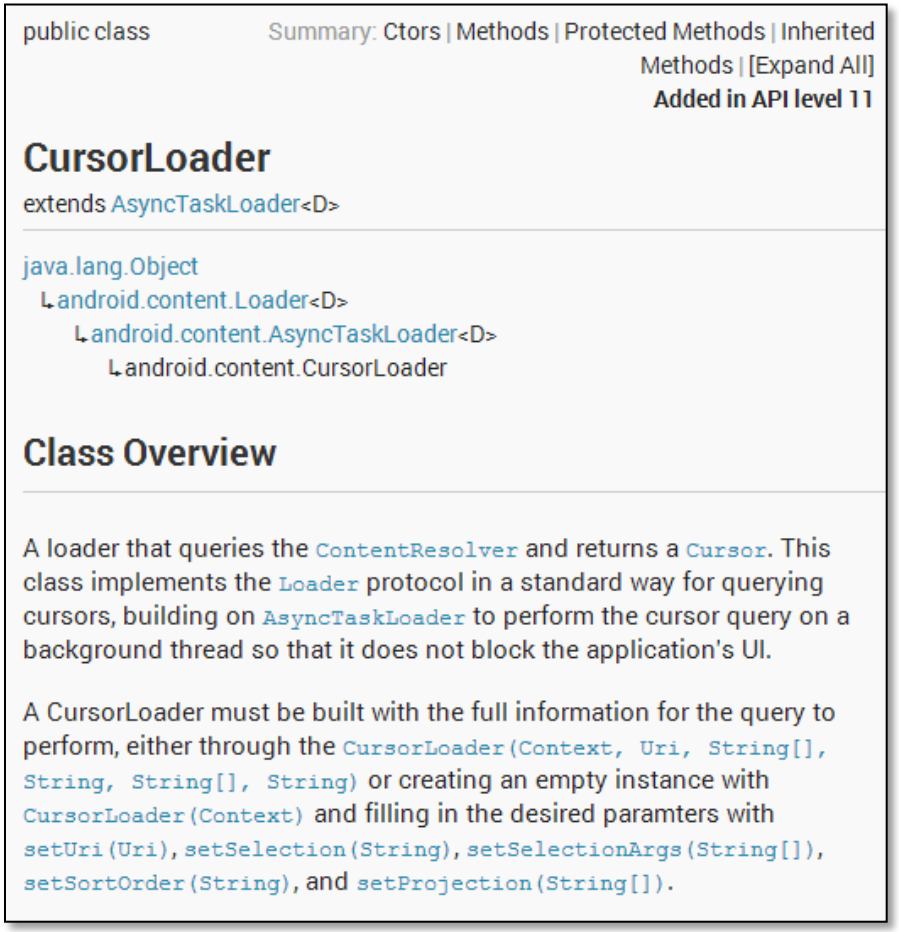
A loader that queries the `ContentResolver` and returns a `Cursor`. This class implements the `Loader` protocol in a standard way for querying cursors, building on `AsyncTaskLoader` to perform the cursor query on a background thread so that it does not block the application's UI.

A CursorLoader must be built with the full information for the query to perform, either through the `CursorLoader(Context, Uri, String[], String, String[], String)` or creating an empty instance with `CursorLoader(Context)` and filling in the desired paramters with `setUri(Uri)`, `setSelection(String)`, `setSelectionArgs(String[])`, `setSortOrder(String)`, and `setProjection(String[])`.

developer.android.com/reference/android/content/CursorLoader.html

# Overview of CursorLoader

- A CursorLoader runs an asynchronous query in the background against a ContentProvider
  - It returns the result to the Activity or Fragment from which it was called
  - It does not block the App's UI
    - The Activity or Fragment can thus continue to interact with the user which the query is ongoing



public class Summary: Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 11**

## CursorLoader

extends AsyncTaskLoader<D>

java.lang.Object
↳ android.content.Loader<D>
↳ android.content.AsyncTaskLoader<D>
↳ android.content.CursorLoader

## Class Overview

A loader that queries the `ContentResolver` and returns a `Cursor`. This class implements the `Loader` protocol in a standard way for querying cursors, building on `AsyncTaskLoader` to perform the cursor query on a background thread so that it does not block the application's UI.

A CursorLoader must be built with the full information for the query to perform, either through the `CursorLoader(Context, Uri, String[], String, String[], String)` or creating an empty instance with `CursorLoader(Context)` and filling in the desired paramters with `setUri(Uri)`, `setSelection(String)`, `setSelectionArgs(String[])`, `setSortOrder(String)`, and `setProjection(String[])`.

developer.android.com/reference/android/content/CursorLoader.html

# Overview of CursorLoader

- A CursorLoader runs an asynchronous query in the background against a ContentProvider
- A CursorLoader can be built with the full info for the query to perform

```
// Create a new CursorLoader
// with the query parameters
public CursorLoader
  makeCursorLoader
    (Context context,
     Uri uri) {
    return new CursorLoader
      (context,
       uri, null, null,
       null, null);
}
```

developer.android.com/reference/android/content/CursorLoader.html

# Overview of CursorLoader

- A CursorLoader runs an asynchronous query in the background against a ContentProvider
- A CursorLoader can be built with the full info for the query to perform
- You can also create empty instance with CursorLoader(Context) & fill in desired parameters with
  - setUri()
  - setSelection()
  - setSelectionArgs()
  - setSortOrder()
  - setProjection()

```
// Create a new CursorLoader
// with the query parameters
public CursorLoader
  makeCursorLoader
    (Context context,
     Uri uri) {
    CursorLoader cl =
      new CursorLoader
        (context);
    // Set the desired URI
    cl.setUri(uri);
}
```

# Using LoaderManager & CursorLoader

- To use the LoaderManager & CursorLoader have your Activity implement the LoaderManager.LoaderCallbacks<Cursor> class & override the following methods:
  - onCreateLoader() – Hook method that instantiates & returns a new Loader

```
public class DownloadActivity
       extends Activity
       implements LoaderCallbacks
                           <Cursor> {
  public Loader<Cursor>
    onCreateLoader(int id, Bundle){
      return makeCursorLoader
        (getApplicationContext(),
         ImageProvider.IMAGES_URI);
  }
```

# Using LoaderManager & CursorLoader

- To use the LoaderManager & CursorLoader have your Activity implement the LoaderManager.LoaderCallbacks<Cursor> class & override the following methods:
  - onCreateLoader() – Hook method that instantiates & returns a new Loader
  - onLoadFinished() – Hook method called when a previously created loader has finished its loading
    - LoaderManager can callback to onLoadFinished() method each time ContentProvider's data is updated

```
public class DownloadActivity
       extends Activity
       implements LoaderCallbacks
                          <Cursor> {
  public Loader<Cursor>
    onCreateLoader(int id, Bundle){
      ...
  }

  public void onLoadFinished
    (Loader<Cursor> a, Cursor c) {
      loadImageFromCursor(c);
  }
```

developer.android.com/reference/android/app/LoaderManager.LoaderCallbacks.html

# Using LoaderManager & CursorLoader

- To use the LoaderManager & CursorLoader have your Activity implement the LoaderManager.LoaderCallbacks<Cursor> class & override the following methods:
  - onCreateLoader() – Hook method that instantiates & returns a new Loader
  - onLoadFinished() – Hook method called when a previously created loader has finished its loading
  - onLoaderReset() – Hook method called when a created loader is being reset, making its data unavailable

```
public class DownloadActivity
       extends Activity
       implements LoaderCallbacks
                          <Cursor> {
  public Loader<Cursor>
    onCreateLoader(int id, Bundle){
      ...
  }

  public void onLoadFinished
    (Loader<Cursor> a, Cursor c) {
      ...
  }

  public void onLoaderReset
            (Loader<Cursor> loader) {
    onClickReset(null);
  }
```

# Example of LoaderManager ContentProvider

- Shows how to use LoaderManager & CursorLoader to implement a ContentProvider that is accessed asynchronously

# Example of LoaderManager ContentProvider

- Shows how to implement a ContentProvider that is accessed asynchronously
- Stores the DataRecord objects in a HashMap

# Example of LoaderManager ContentProvider

- Shows how to implement a ContentProvider that is accessed asynchronously
- Stores the DataRecord objects in a HashMap
- Supports all the ContentProvider "CRUD" operations
  - All of which are implemented as synchronized Java methods

# Example of LoaderManager ContentProvider

- Shows how to implement a ContentProvider that is accessed asynchronously
- Stores the DataRecord objects in a HashMap
- Supports all the ContentProvider "CRUD" operations
- Client Activity accesses the ContentProvider using asynchronous two-way calls made via a LoaderManager & CursorLoader

# ContactProviderActivityAsync Example

```
public class ContactProviderActivityAsync
             extends ListActivity
             implements LoaderManager.LoaderCallbacks<Cursor> {
```

This Activity handles callbacks from the LoaderManager

```
  private static final int LOADER_ID = 0;
```

The loader's unique id

```
  private LoaderManager.LoaderCallbacks<Cursor> mCallbacks;
```

The callbacks through which we interact with the LoaderManager

```
  private SimpleCursorAdapter mAdapter;

  ...
```

The adapter that binds our data to the ListView

# ContactProviderActivityAsync Example

```
public class ContactProviderActivityAsync
             extends ListActivity
             implements LoaderManager.LoaderCallbacks<Cursor> {
  public void onCreate(Bundle savedInstanceState) {
```

Do all the same initialization as before

```
    // ...

    String[] dataColumns = { "_id", "data" };

    int[] viewIDs = { R.id.idString, R.id.data };

    mAdapter = new SimpleCursorAdapter(this,
        R.layout.list_layout, null, dataColumns, viewIDs, 0);
```

Map columns from an **initially null** cursor to TextViews or ImageViews defined in an XML file

developer.android.com/reference/android/widget/SimpleCursorAdapter.html

# ContactProviderActivityAsync Example

```
public class ContactProviderActivityAsync
             extends ListActivity
             implements LoaderManager.LoaderCallbacks<Cursor> {
  public void onCreate(Bundle savedInstanceState) {
    ...
```

Associate adapter w/ListView

```
    setListAdapter(mAdapter);
```

Activity is the callback object for LoaderManager

```
    mCallbacks = this;
```

Initialize Loader with id & mCallbacks

```
    getLoaderManager().initLoader(LOADER_ID, null, mCallbacks);
  }
```

developer.android.com/reference/android/app/LoaderManager.html#initLoader(int, android.os.Bundle, LoaderManager.LoaderCallbacks<D>)

# ContactProviderActivityAsync Example

```
public class ContactProviderActivityAsync
             extends ListActivity
             implements LoaderManager.LoaderCallbacks<Cursor> {
```

Create a new CursorLoader with query parameter

```
  public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    return new CursorLoader(ContentProviderActivityAsync.this,
                            MyCP.CONTENT_URI,
                            null, null, null, null);
  }

  ...
```

# ContactProviderActivityAsync Example

```
public class ContactProviderActivityAsync
             extends ListActivity
             implements LoaderManager.LoaderCallbacks<Cursor> {
  ...

  public void onLoadFinished(Loader<Cursor> loader,
                             Cursor cursor) {
    switch (loader.getId()) {
    case LOADER_ID:

      mAdapter.swapCursor(cursor);

    }

  }
```

Async load is complete & data is available for SimpleCursorAdapter

The listview now displays the queried data

# Summary

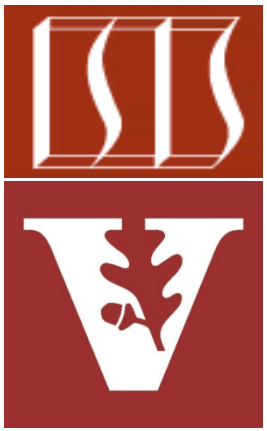
- The LoaderManager framework helps an App manage longer-running operations in conjunction with the Activity or Fragment lifecycle

# Summary

- The LoaderManager framework helps an App manage longer-running operations in conjunction with the Activity or Fragment lifecycle
- The most common use of LoaderManager is with a CursorLoader
  - Apps can write their own loaders for loading other types of data

# Android Content Providers: Programming with AsyncQueryHandler

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

# Learning Objectives in this Part of the Module

- Understand how to access Content Providers asynchronously via the AsyncQueryHandler framework

# Overview of AsyncQueryHandler

- LoaderManager & CursorLoader only provide a way to access results of async invoked query() operations on ContentResolvers
  - Other ContentResolver operations are still synchronous

public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 1**

## AsyncQueryHandler

extends Handler

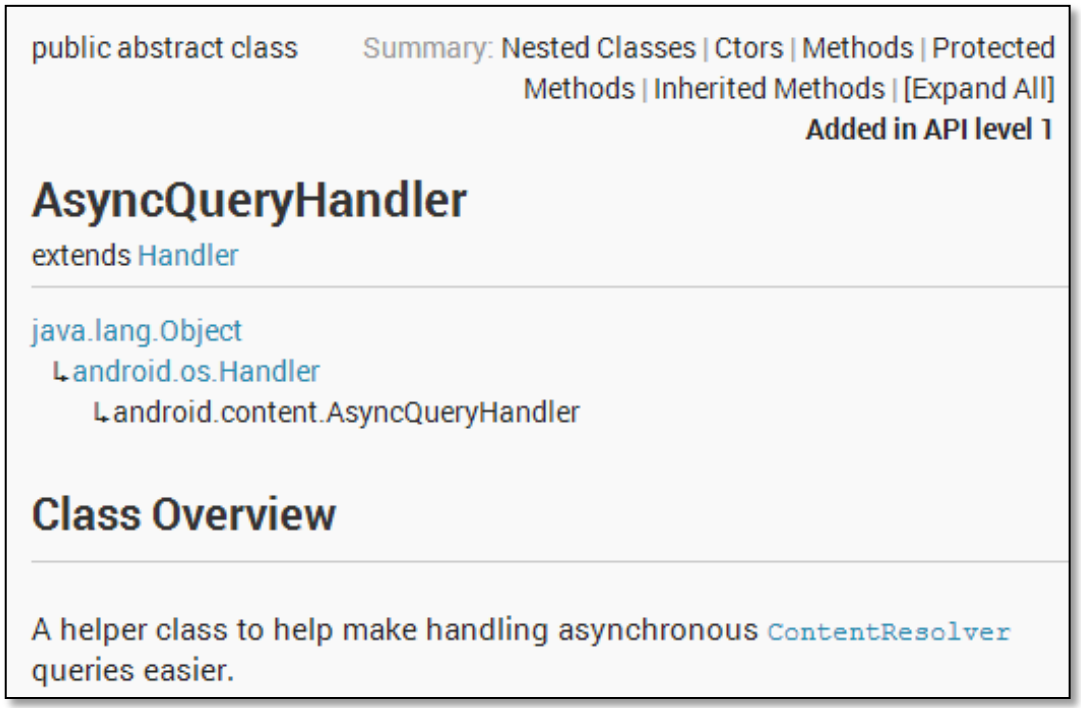
java.lang.Object
↳android.os.Handler
↳android.content.AsyncQueryHandler

## Class Overview

A helper class to help make handling asynchronous `ContentResolver` queries easier.

# Overview of AsyncQueryHandler

- LoaderManager & CursorLoader only provide a way to access results of async invoked query() operations on ContentResolvers
- AsyncQueryHandler invokes all ContentResolver calls asynchronously



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 1**

## AsyncQueryHandler

extends Handler

java.lang.Object
↳android.os.Handler
↳android.content.AsyncQueryHandler

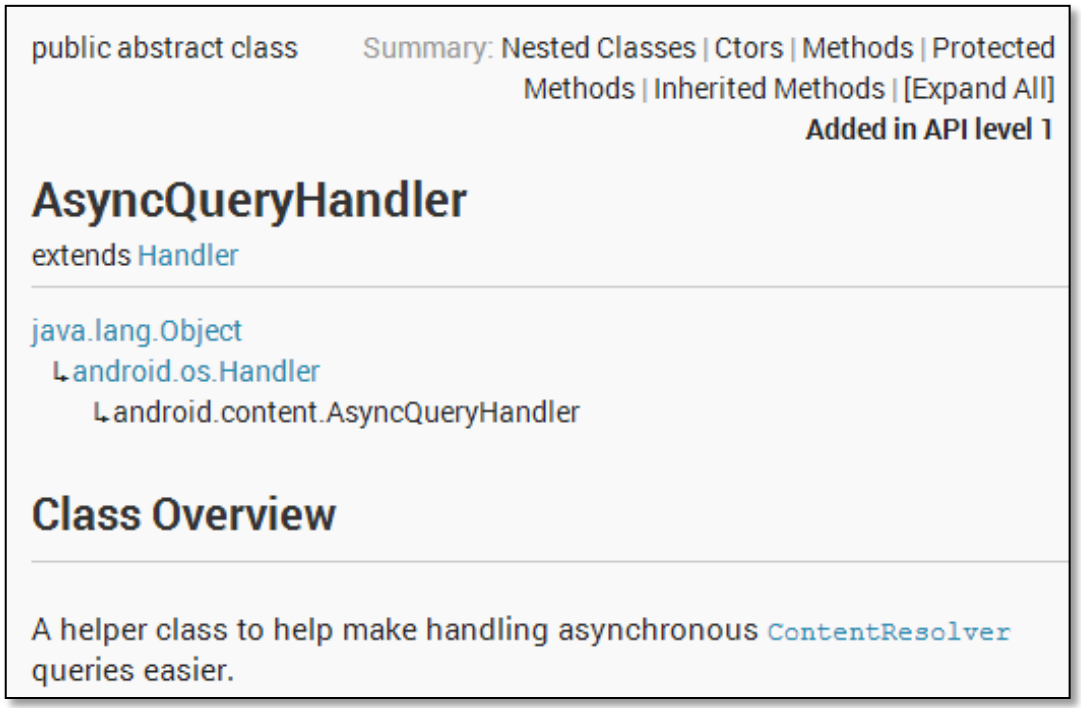
## Class Overview

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html

# Overview of AsyncQueryHandler

- LoaderManager & CursorLoader only provide a way to access results of async invoked query() operations on ContentResolvers
- AsyncQueryHandler invokes all ContentResolver calls asynchronously
  - startDelete () – Begins an asynchronous delete



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

Added in API level 1

**AsyncQueryHandler**

extends Handler

java.lang.Object
↳ android.os.Handler
↳ android.content.AsyncQueryHandler

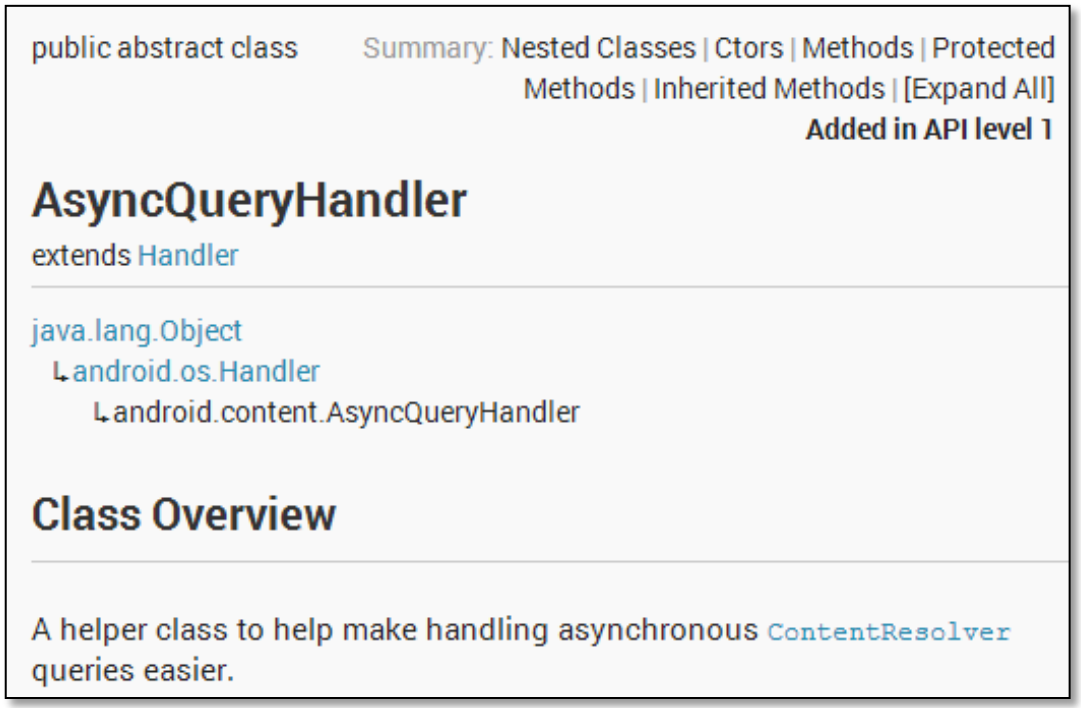
**Class Overview**

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html#startDelete(int, java.lang.Object, Uri, java.lang.String, java.lang.String[])

# Overview of AsyncQueryHandler

- LoaderManager & CursorLoader only provide a way to access results of async invoked query() operations on ContentResolvers
- AsyncQueryHandler invokes all ContentResolver calls asynchronously
  - startDelete () – Begins an asynchronous delete
  - startInsert () – Begins an asynchronous insert



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

Added in API level 1

**AsyncQueryHandler**

extends Handler

java.lang.Object
↳android.os.Handler
↳android.content.AsyncQueryHandler

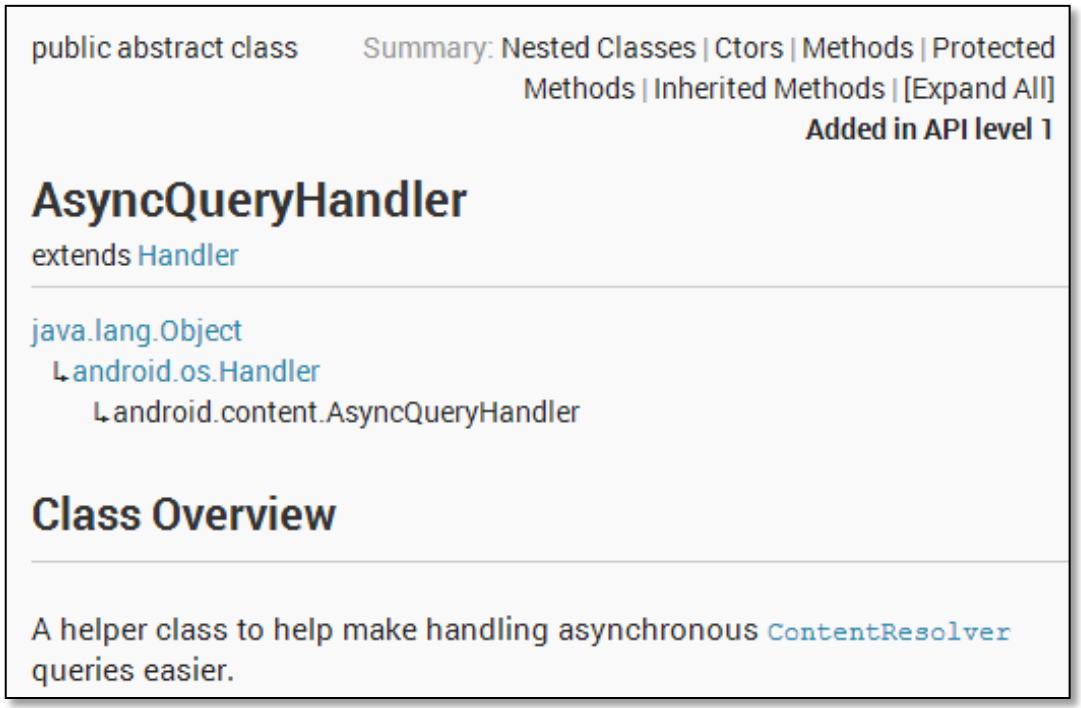
**Class Overview**

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html#startInsert(int, java.lang.Object, Uri, android.content.ContentValues)

# Overview of AsyncQueryHandler

- LoaderManager & CursorLoader only provide a way to access results of async invoked query() operations on ContentResolvers
- AsyncQueryHandler invokes all ContentResolver calls asynchronously
  - startDelete () – Begins an asynchronous delete
  - startInsert () – Begins an asynchronous insert
  - startQuery () – Begins an asynchronous query



public abstract class

Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 1**

## AsyncQueryHandler

extends Handler

java.lang.Object
↳ android.os.Handler
↳ android.content.AsyncQueryHandler

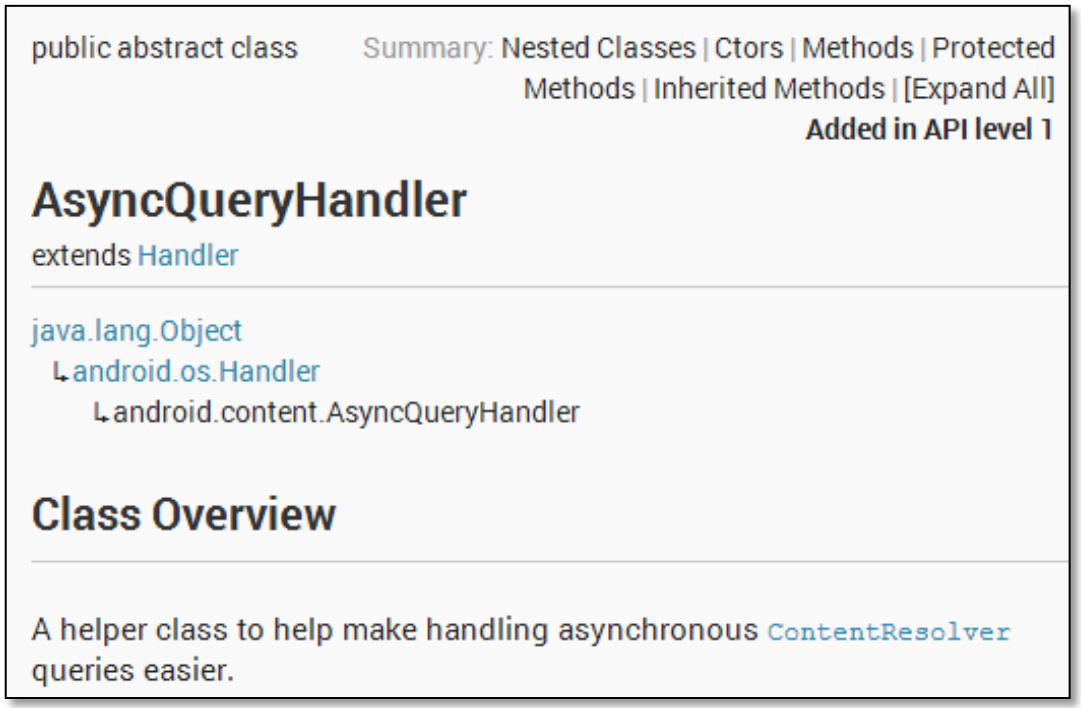
## Class Overview

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html#startQuery(int, Object, Uri, String[], String, String[], String)

# Overview of AsyncQueryHandler

- LoaderManager & CursorLoader only provide a way to access results of async invoked query() operations on ContentResolvers
- AsyncQueryHandler invokes all ContentResolver calls asynchronously
  - startDelete () – Begins an asynchronous delete
  - startInsert () – Begins an asynchronous insert
  - startQuery () – Begins an asynchronous query
  - startUpdate () – Begins an asynchronous update



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

Added in API level 1

**AsyncQueryHandler**

extends Handler

java.lang.Object
↳ android.os.Handler
↳ android.content.AsyncQueryHandler

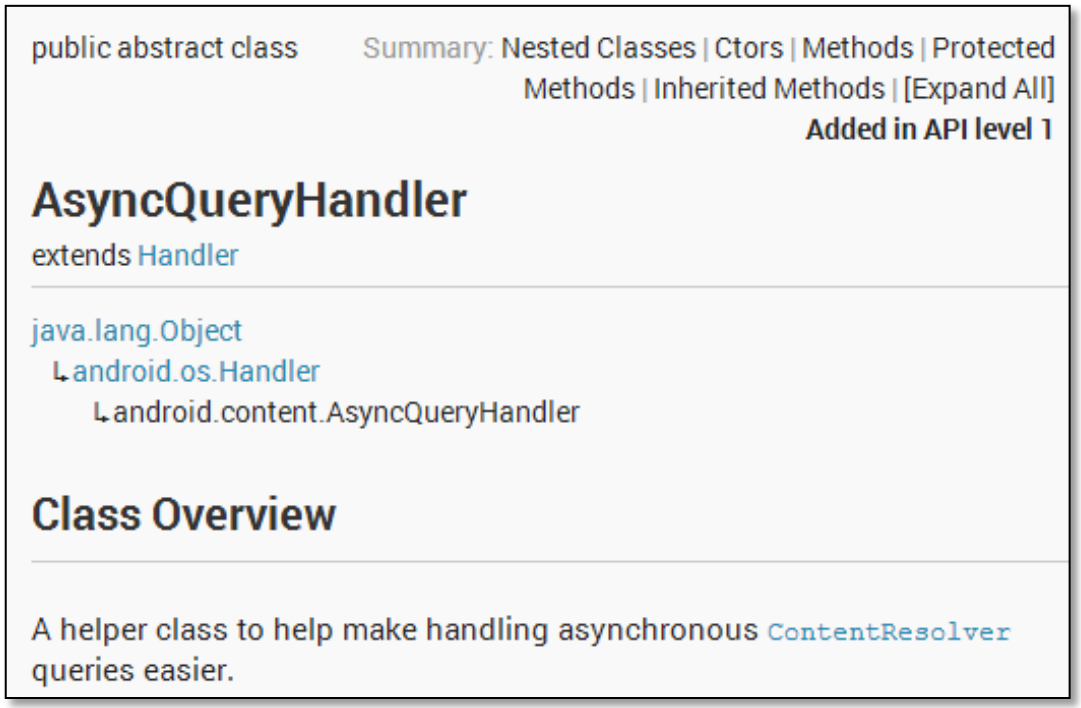
**Class Overview**

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html#startUpdate(int, Object, Uri, android.content.ContentValues, String, String[])

# Overview of AsyncQueryHandler

- LoaderManager & CursorLoader only provide a way to access results of async invoked query() operations on ContentResolvers
- AsyncQueryHandler invokes all ContentResolver calls asynchronously
- Async operations can be cancelled via cancelOperation()



public abstract class

Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 1**

## AsyncQueryHandler

extends Handler

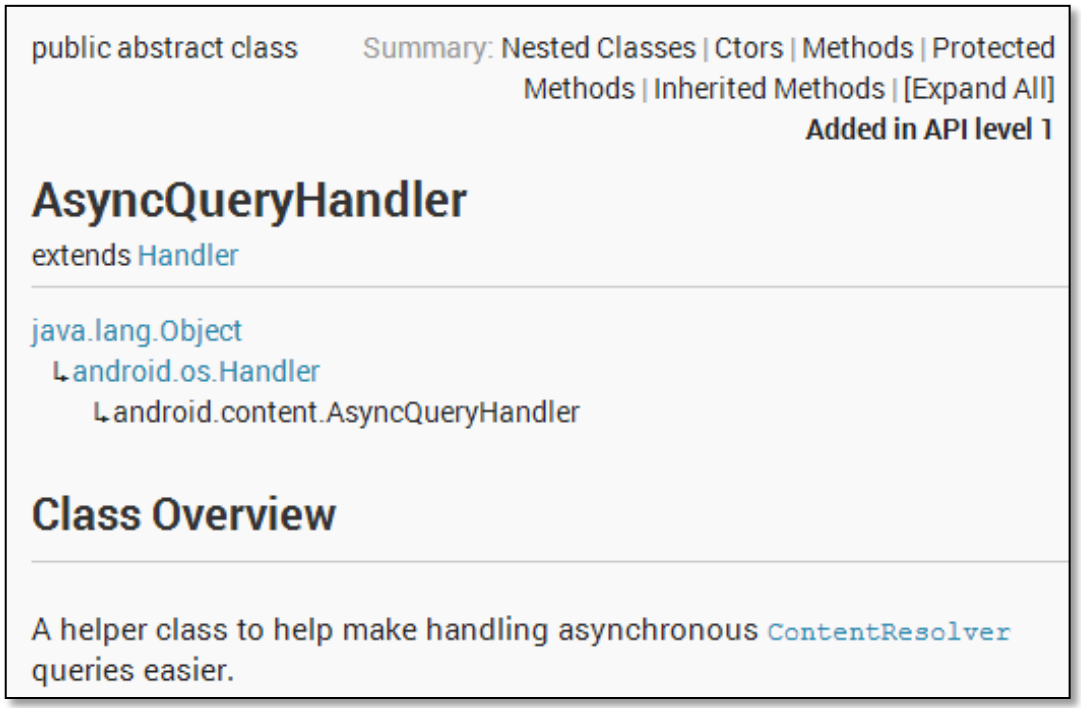
java.lang.Object
↳ android.os.Handler
↳ android.content.AsyncQueryHandler

## Class Overview

A helper class to help make handling asynchronous `ContentResolver` queries easier.

# Overview of AsyncQueryHandler

- Defines callback hook methods that are invoked in a handler thread when async operations invoked on a ContentResolver have completed
  - onDeleteComplete () – Called when async delete completes



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 1**

## AsyncQueryHandler

extends Handler

java.lang.Object
↳android.os.Handler
↳android.content.AsyncQueryHandler

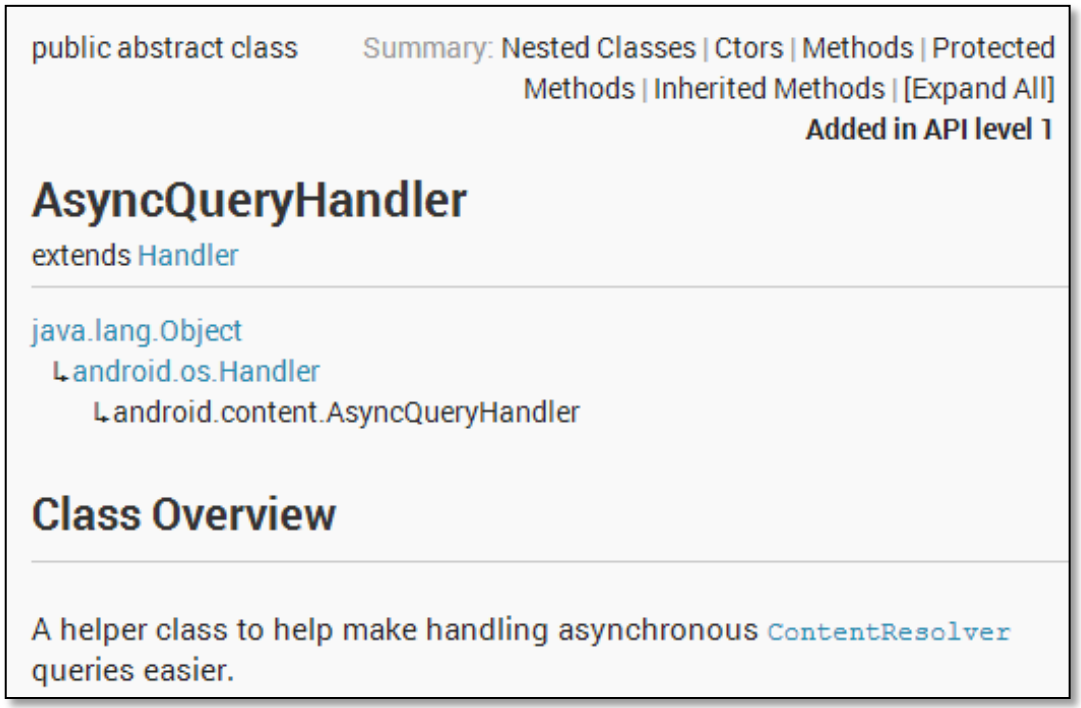
## Class Overview

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html#onDeleteComplete(int, java.lang.Object, int)

# Overview of AsyncQueryHandler

- Defines callback hook methods that are invoked in a handler thread when async operations invoked on a ContentResolver have completed
  - onDeleteComplete () – Called when async delete completes
  - onInsertComplete () – Called when async insert completes



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 1**

## AsyncQueryHandler

extends Handler

java.lang.Object
↳android.os.Handler
↳android.content.AsyncQueryHandler

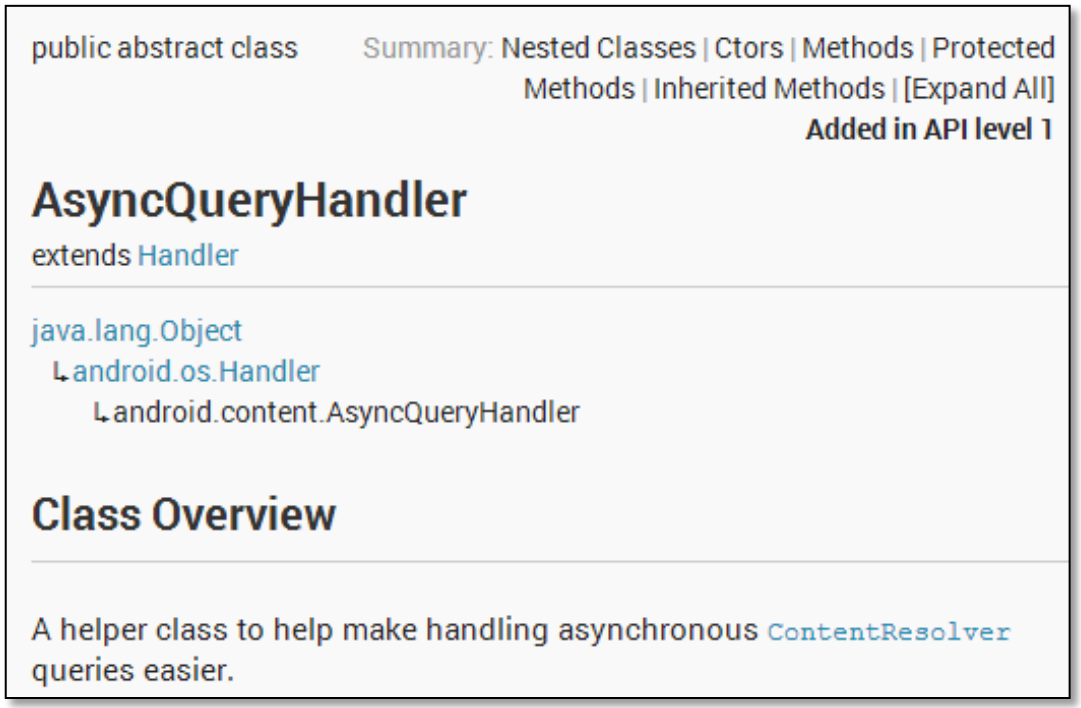
## Class Overview

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html#onInsertComplete(int, java.lang.Object, android.net.Uri)

# Overview of AsyncQueryHandler

- Defines callback hook methods that are invoked in a handler thread when async operations invoked on a ContentResolver have completed
  - onDeleteComplete () – Called when async delete completes
  - onInsertComplete () – Called when async insert completes
  - onQueryComplete () – Called when async query completes



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 1**

## AsyncQueryHandler

extends Handler

java.lang.Object
↳android.os.Handler
↳android.content.AsyncQueryHandler

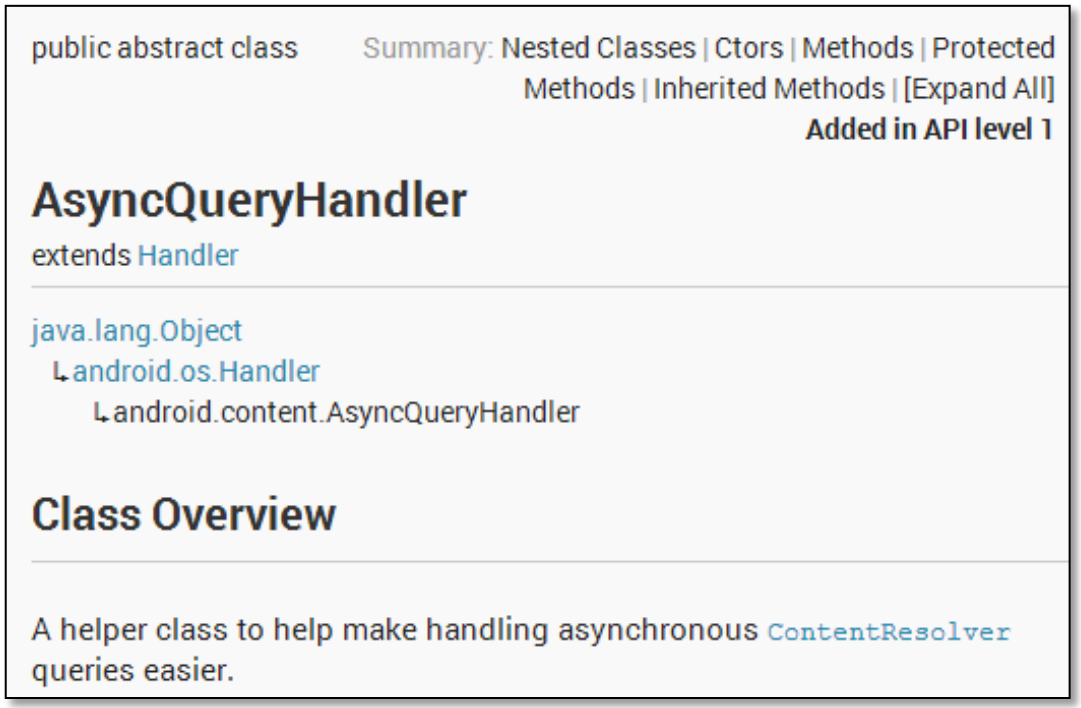
## Class Overview

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html#onQueryComplete(int, java.lang.Object, android.database.Cursor)

# Overview of AsyncQueryHandler

- Defines callback hook methods that are invoked in a handler thread when async operations invoked on a ContentResolver have completed
  - onDeleteComplete () – Called when async delete completes
  - onInsertComplete () – Called when async insert completes
  - onQueryComplete () – Called when async query completes
  - onUpdateComplete () – Called when async update completes



public abstract class Summary: Nested Classes | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

**Added in API level 1**

## AsyncQueryHandler

extends Handler

java.lang.Object
↳android.os.Handler
↳android.content.AsyncQueryHandler

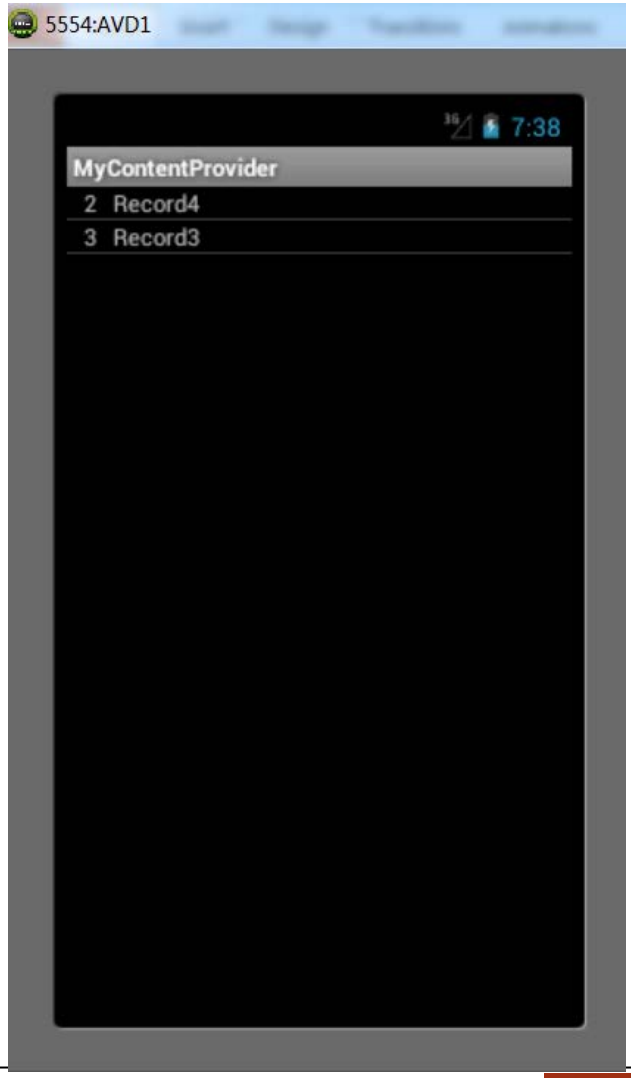
## Class Overview

A helper class to help make handling asynchronous `ContentResolver` queries easier.

developer.android.com/reference/android/content/AsyncQueryHandler.html#onUpdateComplete(int, java.lang.Object, int)

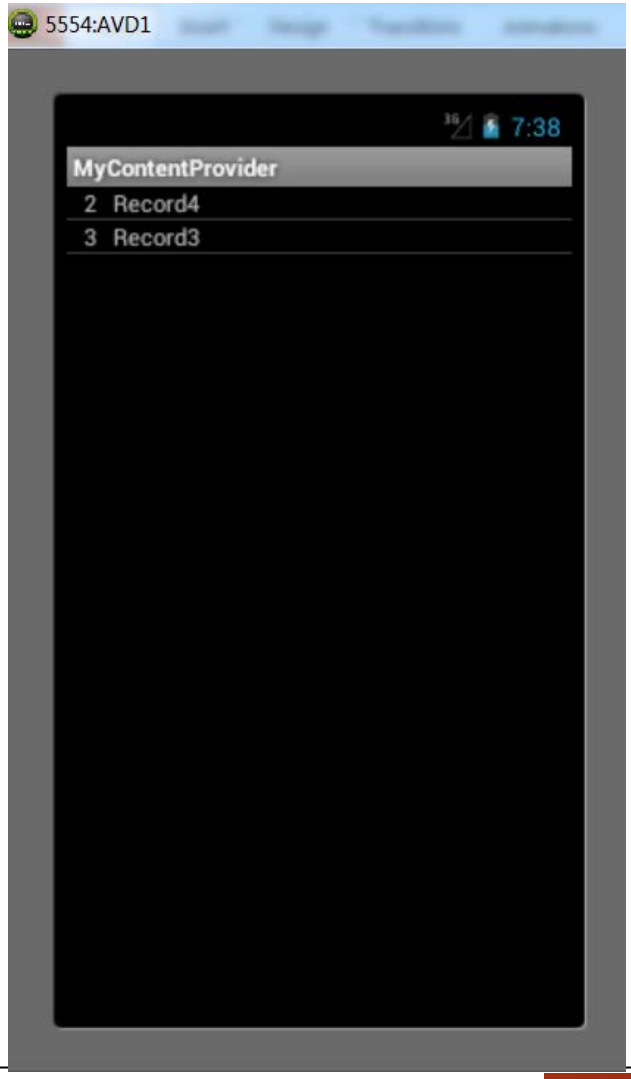
# Example AsyncQueryHandler ContentProvider

- Shows how to implement a ContentProvider that is accessed asynchronously



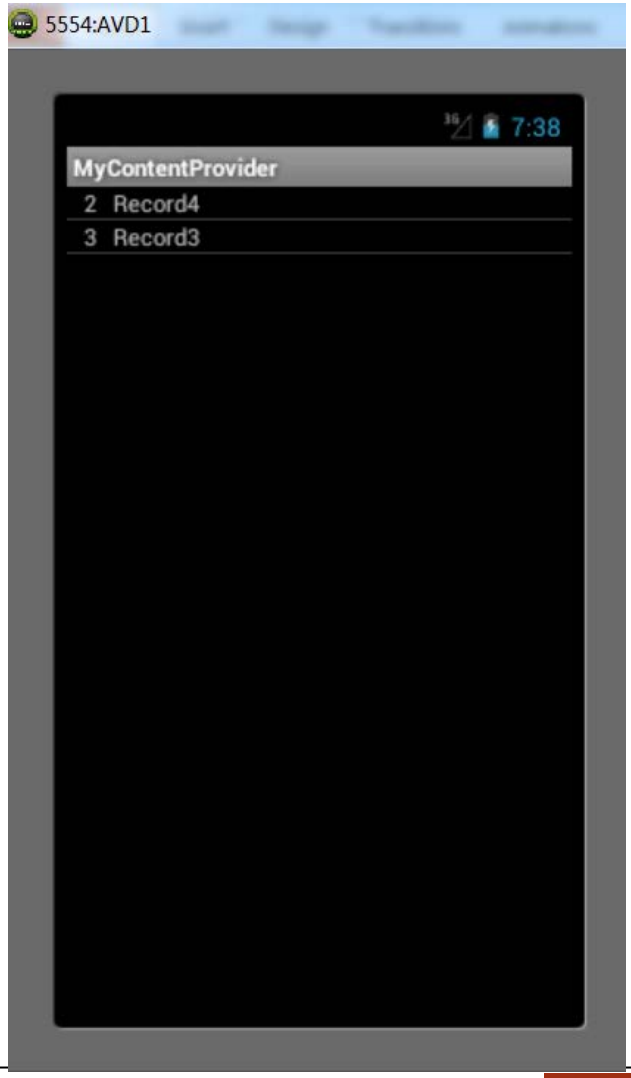
# Example AsyncQueryHandler ContentProvider

- Shows how to implement a ContentProvider that is accessed asynchronously
- Stores the DataRecord objects in a HashMap



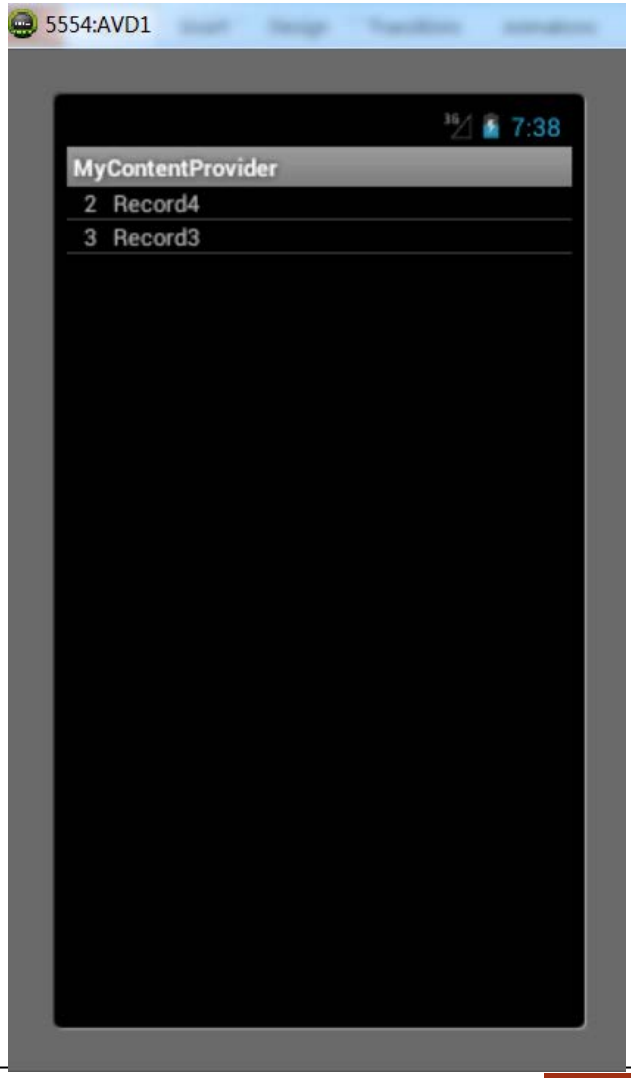
# Example AsyncQueryHandler ContentProvider

- Shows how to implement a ContentProvider that is accessed asynchronously
- Stores the DataRecord objects in a HashMap
- Supports all the ContentProvider "CRUD" operations
  - All of which are implemented as synchronized Java methods



# Example AsyncQueryHandler ContentProvider

- Shows how to implement a ContentProvider that is accessed asynchronously
- Stores the DataRecord objects in a HashMap
- Supports all the ContentProvider "CRUD" operations
  - All of which are implemented as synchronized Java methods
- Client Activity accesses the ContentProvider using asynchronous two-way calls made via an AsyncQueryHandler
  - Note the use of the *Command*, *Asynchronous Completion Token*, & *Proactor* patterns in this example



# Example AsyncQueryHandler ContentProvider

```
public class ContactProviderActivityAsync extends ListActivity {
```

The adapter that binds our data to the Listview

```
  private SimpleCursorAdapter mAdapter;
```

This class implements the *Command* pattern (the execute() method) & the *Asynchronous Completion Token* pattern (by virtue of inheriting from AsyncQueryHandler)

```
  abstract class CompletionHandler extends AsyncQueryHandler {
    public CompletionHandler() {
      super (getContentResolver());
    }

    abstract public void execute();
}
```

Command hook method that must be overridden by subclasses

# Example AsyncQueryHandler ContentProvider

```
class InsertQueryHandler extends CompletionHandler {
  private Object mNextCommand;
  private String mValue = null;

  public InsertQueryHandler (String value, Object nextCommand) {
    mValue = value; mNextCommand = nextCommand;
  }
```

Store value to insert & next command to execute when the async operation is done

```
  public void execute() {
    ContentValues v = new ContentValues();
    v.put("data", mValue);
    startInsert(0, mNextCommand, MyCP.CONTENT_URI, v);
  }
```

Invoke the async insert operation on the CONTENT_URI

```
  public void onInsertComplete(int t, Object nextCommand, Uri u) {
    ((CompletionHandler) nextCommand).execute();
  }
}
```

Execute the next command when async insert completes

# Example AsyncQueryHandler ContentProvider

```
class DeleteUpdateQueryHandler extends CompletionHandler {
  private String mDeleteItem, mUpdateItem, mUpdateValue;

  DeleteUpdateQueryHandler(String dI, String uI, String uV) {
    mDeleteItem = dI; mUpdateItem = uI; mUpdateValue = uV;
  }

  public void execute() {

    startDelete(0, (Object) new UpdateQueryHandler(mUpdateItem,
                                                   mUpdateValue),
                Uri.parse (MyCP.CONTENT_URI + mDeleteItem),
                           (String) null, (String[]) null);
  }

  public void onDeleteComplete(int t, Object nextCommand, int r) {
    ((CompletionHandler) nextCommand).execute();
  }
}
```

Store items to delete & update, as well as the value to update

Invoke the async delete operation, passing in the next command

Add mDeleteItem to the CONTENT_URI

Execute the next command when async delete completes

# Example AsyncQueryHandler ContentProvider

```
class UpdateQueryHandler extends CompletionHandler {
  private String mUpdateItem, mUpdateValue;

  UpdateQueryHandler(String updateItem, String updateValue) {
    mUpdateItem = updateItem; mUpdateValue = updateValue;
  }

  public void execute() {
    ContentValues v = new ContentValues();
    v.put("data", mUpdateValue);
    startUpdate(0, (Object) new QueryQueryHandler(),
                Uri.parse (MyCP.CONTENT_URI + mUpdateItem), v,
                           (String) null, (String[]) null);
  }

  public void onUpdateComplete(int t, Object nextCommand, int r){
    ((CompletionHandler) nextCommand).execute();
  }
}
```

Store item & value to update

Invoke the async update operation

Add mUpdateItem to the CONTENT_URI

Execute the next command when async update completes

# Example AsyncQueryHandler ContentProvider

```
class QueryQueryHandler extends CompletionHandler {

   public void execute() {
     startQuery(0, null, MyCP.CONTENT_URI,
                (String []) null, (String) null, (String[]) null,
                (String) null);
   }

   public void onQueryComplete(int t, Object command, Cursor c) {
     String[] cols = {"_id","data"};
     int[] ids = {R.id.idString, R.id.data};
     mAdapter =
       new SimpleCursorAdapter(ContactProviderActivityAsync.this,
                               R.layout.list_layout, c,
                               cols, ids);
     setListAdapter(mAdapter);
   }
}
```

Invoke the async query operation

Display the results when the query completes

# Example AsyncQueryHandler ContentProvider

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);



  new InsertQueryHandler("Value1",
    new InsertQueryHandler("Value2",
      new InsertQueryHandler("Value3",
        new DeleteUpdateQueryHandler("/1",
                                     "/2",
                                     "Value4")))).execute();
}
```

Insert "Value1", "Value2", & "Value3" into ContentProvider, then delete item 1 & change the value of item 2 to "Value4"

# Summary

- AsyncQueryHandler is a helper class that helps make handling async ContentResolver queries easier
  - It's not widely used in Android, though the MMS app uses it extensively

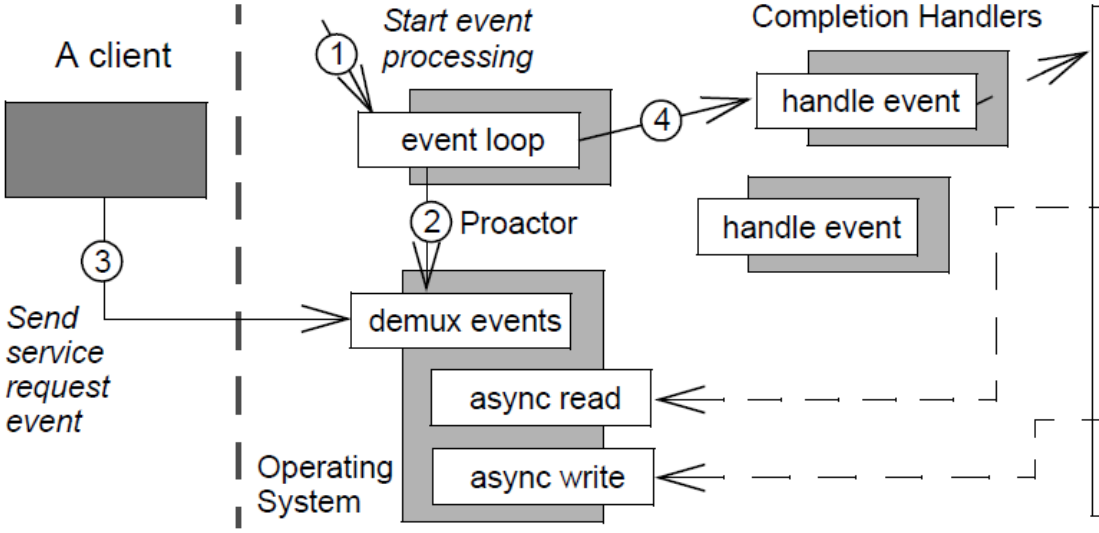
```
AsyncQueryHandler mQueryHandler;
...

mQueryHandler = new AsyncQueryHandler(cr) {
  protected void onQueryComplete
    (int token, Object cookie, Cursor c) {
      ...

    }
  };

mQueryHandler.startQuery(0, null, searchUri,
                         null, null, null, null);
```

When query completes cons up a new CursorAdapter to display the results

Initiate a query for MMS threads that match the search string

# Summary

- AsyncQueryHandler is a helper class that helps make handling async ContentResolver queries easier
- AsyncQueryHandler implements several patterns:
  - *Proactor* – Split an App's functionality into async operations that perform activities on event sources & completion handlers that use the results of async operations to implement App business logic




```
handle_event (Event event)
begin ## Process the received event
   if (event.type == REQUEST)
      ## Read request asynchronously
      ## and return control.
      async_read ();

   elsif (event.type == READ_COMPLETE)
      ## Process event, deliver results
      ## asynchronously, and return control.
      process_data ();
      async_write ();
   fi
end
```

# Summary

- AsyncQueryHandler is a helper class that helps make handling async ContentResolver queries easier
- AsyncQueryHandler implements several patterns:
  - *Proactor* –
  - *Asynchronous Completion Token* – allows an App to efficiently demultiplex & process the responses of async operations it invokes on services