Overview of the AsyncTask Framework (Part 1)

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

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Recognize the capabilities provided by the Android AsyncTask framework

Allows apps to perform background operations & publish results on UI thread *without* manipulating threads, handlers, messages, or runnables
Overview of the AsyncTask Framework
Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected

- e.g., it’s not clear that the classes in the HaMeR framework are related
Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected
- This flexibility works well for simple concurrency use cases
Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected
- This flexibility works well for simple concurrency use cases

e.g., where a background thread posts a runnable to the UI thread...
Classes in HaMeR framework are loosely connected. This flexibility works well for simple concurrency use cases.

... & the UI thread dispatches the run() hook method of the runnable
Overview of the AsyncTask Framework

- However, there are drawbacks to the HaMeR concurrency framework
Overview of the AsyncTask Framework

- However, there are drawbacks to the HaMeR concurrency framework
- Must understand patterns to use this framework effectively

See en.wikipedia.org/wiki/Active_object & www.dre.vanderbilt.edu/~schmidt/CommandProcessor.pdf
Overview of the AsyncTask Framework

- However, there are drawbacks to the HaMeR concurrency framework
  - Must understand patterns to use this framework effectively
  - Tedious & error-prone to use

E.g., apps must understand how to manage the lifecycle of messages
Overview of the AsyncTask Framework

- However, there are drawbacks to the HaMeR concurrency framework
  - Must understand patterns to use this framework effectively
  - Tedious & error-prone to use
- All communication between threads must be explicitly programmed

```java
mHandler.sendMessage(message)
```

```java
mHandler.post(runnable)
```
Overview of the AsyncTask Framework

• However, there are drawbacks to the HaMeR concurrency framework
  • Must understand patterns to use this framework effectively
  • Tedious & error-prone to use
  • All communication between threads must be explicitly programmed
  • Likewise, any “pre” and/or “post” processing must be explicitly programmed
  • e.g., starting & stopping a progress dialog box
• However, there are drawbacks to the HaMeR concurrency framework
  • Must understand patterns to use this framework effectively
  • Tedious & error-prone to use
  • All communication between threads must be explicitly programmed
  • Likewise, any “pre” and/or “post” processing must be explicitly programmed
• Performance can’t be scaled up transparently

Overview of the AsyncTask Framework

- Looper
  - Message Queue
  - Executor
  - UI Thread (main thread)
  - Message
  - Message
  - Message
  - Message
  - Message

Diagram: Diagram showing the AsyncTask framework components such as Looper, Message Queue, Executor, UI Thread, and Messages.
Overview of the AsyncTask Framework

- In contrast, AsyncTask framework classes are more strongly connected
Overview of the AsyncTask Framework

• In contrast, AsyncTask framework classes are more strongly connected
• Complex framework details hidden via Façade pattern

AsyncTask

1. execute(url)

See [en.wikipedia.org/wiki/Facade_pattern](en.wikipedia.org/wiki/Facade_pattern)
Overview of the AsyncTask Framework

- In contrast, AsyncTask framework classes are more strongly connected.
- Complex framework details hidden via Façade pattern.
- Encapsulates a complicated subsystem or framework with a simpler interface.

1. **execute(url)**
2. **onPreExecute()**
3. **execute(future)**
4. **doInBackground()**
5. **onProgressUpdate()**
6. **onPostExecute()**
Overview of the AsyncTask Framework

- In contrast, AsyncTask framework classes are more strongly connected.
- Complex framework details hidden via Façade pattern.
- Yields a smaller “surface area”

i.e., programmers can focus on the “what” not the “how”
Overview of the AsyncTask Framework

- In contrast, AsyncTask framework classes are more strongly connected
  - Complex framework details hidden via *Facade* pattern
  - Yields a smaller “surface area”
  - Run concurrently, *without* directly manipulating threads, handlers, messages, or runnables

See en.wikipedia.org/wiki/Template_Method_pattern
Overview of the AsyncTask Framework

• In contrast, AsyncTask framework classes are more strongly connected
  • Complex framework details hidden via Façade pattern
  • Yields a smaller “surface area”
  • Run concurrently, without directly manipulating threads, handlers, messages, or runnables
• Hook methods can perform pre-processing, post-processing, & communication between the background & UI threads

1. execute(url)
2. onPreExecute()
3. execute(future)
4. doInBackground()
5. onProgressUpdate()
6. onPostExecute()
Overview of the AsyncTask Framework

- Likewise, AsyncTask performance can be scaled up transparently.
Overview of the AsyncTask Framework

• Likewise, AsyncTask performance can be scaled up transparently, e.g.
  • Contains a thread pool that can be specified by programmers

See en.wikipedia.org/wiki/Thread_pool
Overview of the AsyncTask Framework

- Likewise, AsyncTask performance can be scaled up transparently, e.g.
  - Contains a thread pool that can be specified by programmers
  - This thread pool can be implemented via the Java Executor framework

See docs.oracle.com/javase/tutorial/essential/concurrency/executors.html
End of Overview of the AsyncTask Framework (Part 1)
Overview of the AsyncTask Framework (Part 2)

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

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Recognize the capabilities provided by the Android AsyncTask framework

• Know which methods are provided by AsyncTask class

1. `execute(url)`
2. `onPreExecute()`
3. `execute(future)`
4. `doInBackground()`
5. `onProgressUpdate()`
6. `onPostExecute()`
Categories of Methods in AsyncTask
Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods

AsyncTask
extends Object

java.lang.Object

android.os.AsyncTask<Params, Progress, Result>

Class Overview

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is designed to be a helper class around Thread and Handler and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the java.util.concurrent package such as Executor, ThreadPoolExecutor and FutureTask.

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

See developer.android.com/reference/android/os/AsyncTask.html
The AsyncTask class has two types of methods

- Public methods
  - Invoked by apps

AsyncTask<Params, Progress, Result>

execute(Params... params)

- Executes the task with the specified parameters

AsyncTask<Params, Progress, Result>

executeOnExecutor(Executor exec, Params... params)

- Executes the task with the specified parameters on the specified Executor

static void execute(Runnable runnable)

- Convenience version of execute(Object) for use with a simple Runnable object

boolean cancel (boolean mayInterruptIfRunning)

- Attempts to cancel execution of this task

...
Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
    - Invoked by apps

**AsyncTask<Params, Progress, Result>**

- `execute(Params... params)`
  - Executes the task with the specified parameters

**AsyncTask<Params, Progress, Result>**

- `executeOnExecutor(Executor exec, Params... params)`
  - Executes the task with the specified parameters on the specified Executor

**static void execute(Runnable runnable)**

- Convenience version of `execute(Object)` for use with a simple Runnable object

**boolean cancel**

- `cancel(boolean mayInterruptIfRunning)`
  - Attempts to cancel execution of this task

...
Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
    - Invoked by apps

  ```java
  public class AsyncTask<Params, Progress, Result> {
      public static void execute(Runnable runnable)
      // Convenience version of execute(Object)
      // for use with a simple Runnable object

      public boolean cancel(boolean mayInterruptIfRunning)
      // Attempts to cancel execution of this task
      ...
  }
  ```

  ```java
  public class AsyncTask<Params, Progress, Result> {
      public static void executeOnExecutor(Executor exec, Params... params)
      // Executes the task with the specified parameters on the specified Executor

      public void execute(Params... params)
      // Executes the task with the specified parameters

      executeOnExecutor(Executor exec, Params... params)
      // Executes the task with the specified parameters on the specified Executor
  }
  ```

executeOnExecutor() can run multiple AsyncTasks concurrently in a pool of threads within a process
Categories of Methods in the AsyncTask Class

• The AsyncTask class has two types of methods
  • Public methods
    • Invoked by apps

  **AsyncTask<Params, Progress, Result>**
  ```java
  execute(Params... params)
  ```
  • Executes the task with the specified parameters

  **AsyncTask<Params, Progress, Result>**
  ```java
  executeOnExecutor(Executor exec, Params... params)
  ```
  • Executes the task with the specified parameters on the specified Executor

  **static void execute(Runnable runnable)**
  • Convenience version of execute(Object) for use with a simple Runnable object

  **boolean cancel**
  ```java
  (boolean mayInterruptIfRunning)
  ```
  • Attempts to cancel execution of this task

  This method is simply a front-end to the underlying Executor
The AsyncTask class has two types of methods

- Public methods
  - Invoked by apps

AsyncTask<Params, Progress, Result>
execute(Params... params)

- Executes the task with the specified parameters

AsyncTask<Params, Progress, Result>
executeOnExecutor(Executor exec,
Params... params)

- Executes the task with the specified parameters on the specified Executor

static void execute(Runnable runnable)

- Convenience version of execute(Object) for use with a simple Runnable object

boolean cancel

- Attempts to cancel execution of this task

...
Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
  - Protected hook methods
    - Overridden by apps & invoked by the Async Task framework

void onPreExecute()
  - Runs on UI thread before doInBackground()

abstract Result doInBackground(Params... params)
  - Override this method to perform a computation in a background thread

void onProgressUpdate(Progress... values)
  - Runs on UI thread after publishProgress() called

void onPostExecute(Result result)
  - Runs on UI thread after doInBackground()

void onCancelled(Result result)
  - Runs on UI thread after cancel() is invoked & doInBackground() has finished

...
Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
  - Protected hook methods
    - Overridden by apps & invoked by the AsyncTask framework

The AsyncTask framework invokes these methods at different points of time & in different thread contexts
The AsyncTask class has two types of methods:

**Public methods**
- void onPreExecute()
  - Runs on UI thread before doInBackground()

```java
abstract Result doInBackground(Params... params)
```
  - Override this method to perform a computation in a background thread

- void onProgressUpdate(Progress... values)
  - Runs on UI thread after publishProgress() called

- void onPostExecute(Result result)
  - Runs on UI thread after doInBackground() called

- void onCancelled(Result result)
  - Runs on UI thread after cancel() is invoked & doInBackground() has finished

**Protected hook methods**
- Overridden by apps & invoked by the AsyncTask framework

Called in UI thread after execute() called, i.e., prior to any other processing
The AsyncTask class has two types of methods:

- Public methods
- Protected hook methods
  - Overridden by apps and invoked by the AsyncTask framework

**Categories of Methods in the AsyncTask Class**

**Public methods**

- `void onPreExecute()`
  - Runs on UI thread before `doInBackground()`

- `abstract Result doInBackground(Params... params)`
  - Override this method to perform a computation in a background thread

- `void onProgressUpdate(Progress... values)`
  - Runs on UI thread after `publishProgress()` called

- `void onPostExecute(Result result)`
  - Runs on UI thread after `doInBackground()`

- `void onCancelled(Result result)`
  - Runs on UI thread after `cancel()` is invoked and `doInBackground()` has finished

**Protected hook methods**

- `void onPreExecute()`
  - Runs on UI thread before `doInBackground()`

- `abstract Result doInBackground(Params... params)`
  - Override this method to perform a computation in a background thread

- `void onProgressUpdate(Progress... values)`
  - Runs on UI thread after `publishProgress()` called

- `void onPostExecute(Result result)`
  - Runs on UI thread after `doInBackground()`

- `void onCancelled(Result result)`
  - Runs on UI thread after `cancel()` is invoked and `doInBackground()` has finished

---

Runs in the background thread to perform the computation
Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
  - Protected hook methods
    - Overridden by apps & invoked by the AsyncTask framework

void onPreExecute()
  - Runs on UI thread before doInBackground()

abstract Result doInBackground(Params... params)
  - Override this method to perform a computation in a background thread

void onProgressUpdate(Progress... values)
  - Runs on UI thread after publishProgress() called

void onPostExecute(Result result)
  - Runs on UI thread after doInBackground()

void onCancelled(Result result)
  - Runs on UI thread after cancel() is invoked & doInBackground() has finished

Called in UI thread to convey incremental results sent from a background thread
Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
  - Protected hook methods
    - Overridden by apps & invoked by the Async Task framework

```java
void onPreExecute()
  - Runs on UI thread before doInBackground()

abstract Result doInBackground(Params... params)
  - Override this method to perform a computation in a background thread

void onProgressUpdate(Progress... values)
  - Runs on UI thread after publishProgress() called

void onPostExecute(Result result)
  - Runs on UI thread after doInBackground() called

void onCancelled(Result result)
  - Runs on UI thread after cancel() is invoked & doInBackground() has finished
```

Called in UI thread after all background processing is finished successfully
The AsyncTask class has two types of methods:

- Public methods
- Protected hook methods

Protected hook methods are overridden by apps and invoked by the AsyncTask framework.

```java
void onPreExecute()
    // Runs on UI thread before doInBackground()

abstract Result doInBackground(Params... params)
    // Override this method to perform a computation in a background thread

void onProgressUpdate(Progress... values)
    // Runs on UI thread after publishProgress() called

void onPostExecute(Result result)
    // Runs on UI thread after doInBackground() completed

void onCancelled(Result result)
    // Runs on UI thread after cancel() is invoked & doInBackground() has finished
```

Called in UI thread after background processing has been cancelled.
Overriding Hook Methods in the AsyncTask Class
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

**AsyncTask**
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

**ImageDownloadTask**
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

AsyncTask

- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

ImageDownloadTask

- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

```java
public class ImageDownloadTask extends AsyncTask<String, Integer, String> {
    protected void onPreExecute() {
        // Code for pre-execute
    }

    @Override
    protected String doInBackground(String... params) {
        // Code for background task
        return null;
    }

    protected void onProgressUpdate(Integer... values) {
        // Code for progress update
    }

    protected void onPostExecute(String result) {
        // Code for post-execution
    }

    public void onCancel() {
        // Code for cancellation
    }
}
```

Can only be called once per instance of AsyncTask by code in the UI thread

- `execute()`, `executeOnExecutor()`, `cancel()`
- `onPreExecute()`, `doInBackground()`, `onProgressUpdate()`
- `onPostExecute()`, `onCancelled()`
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

AsyncTask
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

Implemented as a variant of the Template Method pattern

ImageDownloadTask
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

See en.wikipedia.org/wiki/Template_method_pattern
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

**AsycTask**

- `executeOnExecutor()`
- `execute()`
- `cancel()`
- `onPreExecute()`
- `doInBackground()`
- `onProgressUpdate()`
- `onPostExecute()`
- `onCancelled()`

Is passed an Executor used to run multiple AsyncTask objects concurrently

**ImageDownloadTask**

- `onPreExecute()`
- `doInBackground()`
- `onProgressUpdate()`
- `onPostExecute()`
- `onCancelled()`
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

**AsyncTask**
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

**ImageDownloadTask**
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

*Invoked by framework in the UI thread to perform initialization actions*
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

**AsyncTask**
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

**ImageDownloadTask**
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

*Invoked by framework in UI thread when background thread calls publishProgress()*
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

**AsyncTask**
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

**ImageDownloadTask**
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

*Invoked by framework in UI thread when doInBackground() returns its result*
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

**AsyncTask**
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

**ImageDownloadTask**
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

Called by application to attempt to stop the execution of the task
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

**AsyncTask**
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

**ImageDownloadTask**
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

---

*Invoked by framework in UI Thread after cancel() is called & doInBackground() is finished*

If onCancelled() is called then onPostExecute() is not called & vice versa
Overriding Hook Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden

**AsyncTask**
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

**ImageDownloadTask**
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

Can periodically call isCancelled() to check if it’s been cancelled

Similar to using the Java interrupt() method to voluntarily shutdown threads
AsyncTask is also parameterized with three types used by its hook methods:

- **Params** – Type used in background work
- **Progress** – Type used when indicating progress
- **Result** – Type of result

**AsyncTask**
- `executeOnExecutor()`
- `execute()`
- `cancel()`
- `onPreExecute()`
- `doInBackground()`
- `onProgressUpdate()`
- `onPostExecute()`
- `onCancelled()`

**ImageDownloadTask**
- `onPreExecute()`
- `doInBackground()`
- `onProgressUpdate()`
- `onPostExecute()`
- `onCancelled()`
AsyncTask is also parameterized with three types used by its hook methods:

- **Params** – Type used in background work
- **Progress** – Type used when indicating progress
- **Result** – Type of result

### ImageDownloadTask
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()
AsyncTask is also parameterized with three types used by its hook methods:

- **Params** – Type used in background work
- **Progress** – Type used when indicating progress
- **Result** – Type of result

### AsyncTask
- `executeOnExecutor()`
- `execute()`
- `cancel()`
- `onPreExecute()`
- `doInBackground()`
- `onProgressUpdate()`
- `onPostExecute()`
- `onCancelled()`

### ImageDownloadTask
- `onPreExecute()`
- `doInBackground()`
- `onProgressUpdate()`
- `onPostExecute()`
- `onCancelled()`
Overriding Hook Methods in the AsyncTask Class

- AsyncTask is also parameterized with three types used by its hook methods:
  - **Params** – Type used in background work
  - **Progress** – Type used when indicating progress
  - **Result** – Type of result

---

**AsyncTask**
- executeOnExecutor()
- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()
Overriding Hook Methods in the AsyncTask Class

- Apps must customize the AsyncTask class to meet their concurrency needs

```java
class DownloadTaskTask extends
    AsyncTask<Uri, Integer, Long> {
    protected Long doInBackground
        (Uri... urls)
    { /* Download files */ }

    protected void onProgressUpdate
        (Integer... progress)
    { setProgressPercent(progress[0]); }

    protected void onPostExecute
        (Long result)
    { showDialog("Downloaded 
        + result
        + " bytes"); }

    new DownloadTask().execute(downloadURL);
}
See developer.android.com/reference/android/os/AsyncTask.html
```
Overriding Hook Methods in the AsyncTask Class

- Apps must customize the AsyncTask class to meet their concurrency needs.

```java
class DownloadTask extends 
    AsyncTask<Uri, Integer, Long> {
    protected Long doInBackground
        (Uri... urls)
    { /* Download files */ }

    protected void onProgressUpdate
        (Integer... progress)
    { setProgressPercent(progress[0]); }

    protected void onPostExecute
        (Long result)
    { showDialog("Downloaded 
        + result
        + " bytes"); }

    new DownloadTask().execute(downloadURL);
}
```

Extend AsyncTask & fill in generic parameters
Apps must customize the AsyncTask class to meet their concurrency needs.

```java
class DownloadTask extends AsyncTask<Uri, Integer, Long> {
    protected Long doInBackground(Uri... urls) {
        /* Download files */
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}

new DownloadTask().execute(downloadURL);
```

Overriding Hook Methods in the AsyncTask Class

*This template method initiates async processing*
Overriding Hook Methods in the AsyncTask Class

Apps must customize the AsyncTask class to meet their concurrency needs.

```java
class DownloadTask extends
    AsyncTask<Uri, Integer, Long> {
    protected Long doInBackground
        (Uri... urls)
    { /* Download files */ } }

protected void onProgressUpdate
    (Integer... progress)
    { setProgressPercent(progress[0]); } }

protected void onPostExecute
    (Long result)
    { showDialog("Downloaded "+ result
                  + " bytes"); } }
}

new DownloadTask().execute(downloadURL);
```

*These hook methods are dispatched by the AsyncTask framework in several different thread contexts.*
End of Overview of the AsyncTask Framework (Part 2)