Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

AsyncTask

Kotlin (/reference/kotlin/android/os/AsyncTask) | **Java**

```
public abstract class AsyncTask
extends Object (/reference/java/lang/Object)
```

java.lang.Object (/reference/java/lang/Object)
  ↳ android.os.AsyncTask<Params, Progress, Result>

This class was deprecated in API level 30.
Use the standard `java.util.concurrent` or Kotlin concurrency utilities (https://developer.android.com/topic/libraries/architecture/coroutines) instead.

AsyncTask was intended to enable proper and easy use of the UI thread. However, the most common use case was for integrating into UI, and that would cause Context leaks, missed callbacks, or crashes on configuration changes. It also has inconsistent behavior on different versions of the platform, swallows exceptions from `doInBackground`, and does not provide much utility over using `Executor` (/reference/java/util/concurrent/Executor)s directly.

AsyncTask is designed to be a helper class around `Thread` (/reference/java/lang/Thread) and `Handler` (/reference/android/os/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` (/reference/java/util/concurrent/Executor), `ThreadPoolExecutor` (/reference/java/util/concurrent/ThreadPoolExecutor) and `FutureTask` (/reference/java/util/concurrent/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`.

Developer Guides

For more information about using tasks and threads, read the Processes and Threads (/guide/components/processes-and-threads) developer guide.

Usage

AsyncTask must be subclassed to be used. The subclass will override at least one method (`doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[]))), and most often will override a second one (`onPostExecute(Result)` (/reference/android/os/AsyncTask#onPostExecute(Result)).)

Here is an example of subclassing:

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

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

    protected void onPostExecute(Long result) {
```

```
        showDialog("Downloaded " + result + " bytes");
    }
}
```

Once created, a task is executed very simply:

```
new DownloadFilesTask().execute(url1, url2, url3);
```

AsyncTask's generic types

The three types used by an asynchronous task are the following:

1. `Params`, the type of the parameters sent to the task upon execution.
2. `Progress`, the type of the progress units published during the background computation.
3. `Result`, the type of the result of the background computation.

Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type `Void` (/reference/java/lang/Void):

```
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

The 4 steps

When an asynchronous task is executed, the task goes through 4 steps:

1. `onPreExecute()` (/reference/android/os/AsyncTask#onPreExecute()), invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by

showing a progress bar in the user interface.

2. `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])), invoked on the background thread immediately after `onPreExecute()` (/reference/android/os/AsyncTask#onPreExecute()) finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use `publishProgress(Progress)` (/reference/android/os/AsyncTask#publishProgress(Progress[])) to publish one or more units of progress. These values are published on the UI thread, in the `onProgressUpdate(Progress)` (/reference/android/os/AsyncTask#onProgressUpdate(Progress[])) step.
3. `onProgressUpdate(Progress)` (/reference/android/os/AsyncTask#onProgressUpdate(Progress[])), invoked on the UI thread after a call to `publishProgress(Progress)` (/reference/android/os/AsyncTask#publishProgress(Progress[])). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
4. `onPostExecute(Result)` (/reference/android/os/AsyncTask#onPostExecute(Result)), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

Cancelling a task

A task can be cancelled at any time by invoking `cancel(boolean)` (/reference/android/os/AsyncTask#cancel(boolean)). Invoking this method will cause subsequent calls to `isCancelled()` (/reference/android/os/AsyncTask#isCancelled()) to return true. After invoking this method, `onCancelled(java.lang.Object)` (/reference/android/os/AsyncTask#onCancelled(Result)), instead of `onPostExecute(java.lang.Object)` (/reference/android/os/AsyncTask#onPostExecute(Result)) will be invoked after `doInBackground(java.lang.Object[])` (/reference/android/os/AsyncTask#doInBackground(Params[])) returns. To ensure that a task is cancelled as quickly as possible, you should always check the return value of `isCancelled()` (/reference/android/os/AsyncTask#isCancelled()) periodically from `doInBackground(java.lang.Object[])`

(/reference/android/os/AsyncTask#doInBackground(Params[])), if possible (inside a loop for instance.)

Threading rules

There are a few threading rules that must be followed for this class to work properly:

- The AsyncTask class must be loaded on the UI thread. This is done automatically as of `Build.VERSION_CODES.JELLY_BEAN` (/reference/android/os/Build.VERSION_CODES#JELLY_BEAN).
- The task instance must be created on the UI thread.
- `execute(Params)` (/reference/android/os/AsyncTask#execute(Params[])) must be invoked on the UI thread.
- Do not call `onPreExecute()` (/reference/android/os/AsyncTask#onPreExecute()), `onPostExecute(Result)` (/reference/android/os/AsyncTask#onPostExecute(Result)), `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])), `onProgressUpdate(Progress)` (/reference/android/os/AsyncTask#onProgressUpdate(Progress[])) manually.
- The task can be executed only once (an exception will be thrown if a second execution is attempted.)

Memory observability

AsyncTask guarantees that all callback calls are synchronized to ensure the following without explicit synchronizations.

- The memory effects of `onPreExecute()` (/reference/android/os/AsyncTask#onPreExecute()), and anything else executed before the call to `execute(Params)` (/reference/android/os/AsyncTask#execute(Params[])), including the construction of the AsyncTask object, are visible to `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])).
- The memory effects of `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])) are visible to

`onPostExecute(Result)` (/reference/android/os/AsyncTask#onPostExecute(Result)).

- Any memory effects of `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])) preceding a call to `publishProgress(Progress)` (/reference/android/os/AsyncTask#publishProgress(Progress[])) are visible to the corresponding `onProgressUpdate(Progress)` (/reference/android/os/AsyncTask#onProgressUpdate(Progress[])) call. (But `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])) continues to run, and care needs to be taken that later updates in `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])) do not interfere with an in-progress `onProgressUpdate(Progress)` (/reference/android/os/AsyncTask#onProgressUpdate(Progress[])) call.)
- Any memory effects preceding a call to `cancel(boolean)` (/reference/android/os/AsyncTask#cancel(boolean)) are visible after a call to `isCancelled()` (/reference/android/os/AsyncTask#isCancelled()) that returns true as a result, or during and after a resulting call to `onCancelled()` (/reference/android/os/AsyncTask#onCancelled()).

Order of execution

When first introduced, AsyncTasks were executed serially on a single background thread. Starting with `Build.VERSION_CODES.DONUT` (/reference/android/os/Build.VERSION_CODES#DONUT), this was changed to a pool of threads allowing multiple tasks to operate in parallel. Starting with `Build.VERSION_CODES.HONEYCOMB` (/reference/android/os/Build.VERSION_CODES#HONEYCOMB), tasks are executed on a single thread to avoid common application errors caused by parallel execution.

If you truly want parallel execution, you can invoke `executeOnExecutor(java.util.concurrent.Executor, java.lang.Object[])` (/reference/android/os/AsyncTask#executeOnExecutor(java.util.concurrent.Executor,%20Params[])) with `THREAD_POOL_EXECUTOR` (/reference/android/os/AsyncTask#THREAD_POOL_EXECUTOR).

Summary

Fields

`public static final` Executor (/reference/java/util/concurrent/Executor)	SERIAL_EXECUTOR (/reference/android/os/AsyncTask#SERIAL_EXECUTOR) *This field was deprecated in API level 30. Globally serializing tasks results in excessive queuing for unrelated operations.*
`public static final` Executor (/reference/java/util/concurrent/Executor)	THREAD_POOL_EXECUTOR (/reference/android/os/AsyncTask#THREAD_POOL_EXECUTOR) *This field was deprecated in API level 30. Using a single thread pool for a general purpose results in suboptimal behavior for different tasks. Small, CPU-bound tasks benefit from a bounded pool and queueing, and long-running blocking tasks, such as network operations, benefit from many threads. Use or create an* Executor (/reference/java/util/concurrent/Executor) *configured for your use case.*

Public constructors

AsyncTask (/reference/android/os/AsyncTask#AsyncTask()) ( )

Creates a new asynchronous task.

Public methods

`final boolean`	cancel (/reference/android/os/AsyncTask#cancel(boolean)) (b `mayInterruptIfRunning`) Attempts to cancel execution of this task.
`final` AsyncTask (/reference/android/os/AsyncTask) `<Params, Progress, Result>`	execute (/reference/android/os/AsyncTask#execute(Params[])) Executes the task with the specified parameters.
`static void`	execute (/reference/android/os/AsyncTask#execute(java.lang.R (/reference/java/lang/Runnable) `runnable`)

	Convenience version of `execute(java.lang.Object)` (/reference/android/os/AsyncTask#execute(Params[])) for use w
`final AsyncTask (/reference/android/os/AsyncTask) <Params, Progress, Result>`	`executeOnExecutor` (/reference/android/os/AsyncTask#executeOnExecutor(java.util. `(Executor (/reference/java/util/concurrent/Executor) exec, I` Executes the task with the specified parameters.
`final Result`	`get` (/reference/android/os/AsyncTask#get(long,%20java.util.cor `timeout, TimeUnit` (/reference/java/util/concurrent/TimeUnit Waits if necessary for at most the given time for the computation result.
`final Result`	`get` (/reference/android/os/AsyncTask#get())`()` Waits if necessary for the computation to complete, and then retr
`final AsyncTask.Status` (/reference/android/os/AsyncTask.Status)	`getStatus` (/reference/android/os/AsyncTask#getStatus())`()` Returns the current status of this task.
`final boolean`	`isCancelled` (/reference/android/os/AsyncTask#isCancelled() Returns `true` if this task was cancelled before it completed norm

Protected methods

`abstract Result`	`doInBackground` (/reference/android/os/AsyncTask#doInBackground(Params[]))`(Params... params)` Override this method to perform a computation on a background thread.
`void`	`onCancelled` (/reference/android/os/AsyncTask#onCancelled())`()` Applications should preferably override `onCancelled(java.lang.Object)` (/reference/android/os/AsyncTask#onCancelled(Result)).

void	onCancelled (/reference/android/os/AsyncTask#onCancelled(Result))(Result result) Runs on the UI thread after cancel(boolean) (/reference/android/os/AsyncTask#cancel(boolean)) is invoked and doInBackground(java.lang.Object[]) (/reference/android/os/AsyncTask#doInBackground(Params[])) has finished.
void	onPostExecute (/reference/android/os/AsyncTask#onPostExecute(Result))(Result result) Runs on the UI thread after doInBackground(Params) (/reference/android/os/AsyncTask#doInBackground(Params[])).
void	onPreExecute (/reference/android/os/AsyncTask#onPreExecute())() Runs on the UI thread before doInBackground(Params) (/reference/android/os/AsyncTask#doInBackground(Params[])).
void	onProgressUpdate (/reference/android/os/AsyncTask#onProgressUpdate(Progress[]))(Progress... values) Runs on the UI thread after publishProgress(Progress) (/reference/android/os/AsyncTask#publishProgress(Progress[])) is invoked.
final void	publishProgress (/reference/android/os/AsyncTask#publishProgress(Progress[]))(Progress... values) This method can be invoked from doInBackground(Params) (/reference/android/os/AsyncTask#doInBackground(Params[])) to publish updates on the UI thread while the background computation is still running.

Inherited methods

From class java.lang.Object (/reference/java/lang/Object)

Object (/reference/java/lang/Object)	clone (/reference/java/lang/Object#clone())() Creates and returns a copy of this object.

`boolean`	`equals` (/reference/java/lang/Object#equals(java.lang.Object)) `(Object` (/reference/java/lang/Object) `obj)` Indicates whether some other object is "equal to" this one.
`void`	`finalize` (/reference/java/lang/Object#finalize()) `( )` Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`final Class` (/reference/java/lang/Class)`<?>`	`getClass` (/reference/java/lang/Object#getClass()) `( )` Returns the runtime class of this `Object`.
`int`	`hashCode` (/reference/java/lang/Object#hashCode()) `( )` Returns a hash code value for the object.
`final void`	`notify` (/reference/java/lang/Object#notify()) `( )` Wakes up a single thread that is waiting on this object's monitor.
`final void`	`notifyAll` (/reference/java/lang/Object#notifyAll()) `( )` Wakes up all threads that are waiting on this object's monitor.
`String` (/reference/java/lang/String)	`toString` (/reference/java/lang/Object#toString()) `( )` Returns a string representation of the object.
`final void`	`wait` (/reference/java/lang/Object#wait(long,%20int)) `(long timeoutMillis, int nanos)` Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.

`final void`	`wait` (/reference/java/lang/Object#wait(long))`(long timeoutMillis)` Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.
`final void`	`wait` (/reference/java/lang/Object#wait())`()` Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*.

Fields

SERIAL_EXECUTOR

Added in API level 11 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public static final Executor (/reference/java/util/concurrent/Executor) SERIAL_EXECUTOR
```

This field was deprecated in API level 30.
Globally serializing tasks results in excessive queuing for unrelated operations.

An `Executor` (/reference/java/util/concurrent/Executor) that executes tasks one at a time in serial order. This serialization is global to a particular process.

THREAD_POOL_EXECUTOR

Added in API level 11 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public static final Executor (/reference/java/util/concurrent/Executor) THREAD_POOL_EXECUTO
```

This field was deprecated in API level 30.
Using a single thread pool for a general purpose results in suboptimal behavior for different tasks. Small, CPU-bound tasks benefit from a bounded pool and queueing, and long-running blocking tasks, such as network operations, benefit from many threads. Use or create an **`Executor`** (/reference/java/util/concurrent/Executor) configured for your use case.

An `Executor` (/reference/java/util/concurrent/Executor) that can be used to execute tasks in parallel.

Public constructors

AsyncTask

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public AsyncTask ()
```

Creates a new asynchronous task. This constructor must be invoked on the UI thread.

Public methods

cancel

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public final boolean cancel (boolean mayInterruptIfRunning)
```

Attempts to cancel execution of this task. This attempt will fail if the task has already completed, already been cancelled, or could not be cancelled for some other reason. If successful, and this task has not started when `cancel` is called, this task should never run. If the task has already started, then the `mayInterruptIfRunning` parameter determines whether the thread executing this task should be interrupted in an attempt to stop the task.

Calling this method will result in `onCancelled(java.lang.Object)` (/reference/android/os/AsyncTask#onCancelled(Result)) being invoked on the UI thread after `doInBackground(java.lang.Object[])` (/reference/android/os/AsyncTask#doInBackground(Params[])) returns. Calling this method guarantees that onPostExecute(Object) is never subsequently invoked, even if `cancel` returns false, but `onPostExecute(Result)` (/reference/android/os/AsyncTask#onPostExecute(Result)) has not yet run. To finish the task as early as possible, check `isCancelled()` (/reference/android/os/AsyncTask#isCancelled()) periodically from `doInBackground(java.lang.Object[])` (/reference/android/os/AsyncTask#doInBackground(Params[])).

This only requests cancellation. It never waits for a running background task to terminate, even if `mayInterruptIfRunning` is true.

Parameters	
`mayInterruptIfRunning`	**`boolean`**: `true` if the thread executing this task should be interrupted; otherwise, in-progress tasks are allowed to complete.

Returns	
`boolean`	`false` if the task could not be cancelled, typically because it has already completed normally; `true` otherwise

See also:

`isCancelled()` (/reference/android/os/AsyncTask#isCancelled())

`onCancelled(Object)` (/reference/android/os/AsyncTask#onCancelled(Result))

execute

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public final AsyncTask (/reference/android/os/AsyncTask)<Params, Progress, Result> exec
```

Executes the task with the specified parameters. The task returns itself (this) so that the caller can keep a reference to it.

Note: this function schedules the task on a queue for a single background thread or pool of threads depending on the platform version. When first introduced, AsyncTasks were executed serially on a single background thread. Starting with `Build.VERSION_CODES.DONUT` (/reference/android/os/Build.VERSION_CODES#DONUT), this was changed to a pool of threads allowing multiple tasks to operate in parallel. Starting `Build.VERSION_CODES.HONEYCOMB` (/reference/android/os/Build.VERSION_CODES#HONEYCOMB), tasks are back to being executed on a single thread to avoid common application errors caused by parallel execution. If you truly want parallel execution, you can use the `executeOnExecutor(Executor, Params)` (/reference/android/os/AsyncTask#executeOnExecutor(java.util.concurrent.Executor,%20Params[])) version of this method with `THREAD_POOL_EXECUTOR` (/reference/android/os/AsyncTask#THREAD_POOL_EXECUTOR); however, see commentary there for warnings on its use.

This method must be invoked on the UI thread.
This method must be called from the main thread of your app.

Parameters	
`params`	`Params`: The parameters of the task.

Returns

AsyncTask (/reference/android/os/AsyncTask)<Params, Progress, Result>	This instance of AsyncTask.

Throws	
IllegalStateException (/reference/java/lang/IllegalStateException)	If getStatus() (/reference/android/os/AsyncTask#getStatus()) returns either AsyncTask.Status#RUNNING (/reference/android/os/AsyncTask.Status#RUNNING) or AsyncTask.Status#FINISHED (/reference/android/os/AsyncTask.Status#FINISHED).

See also:

executeOnExecutor(java.util.concurrent.Executor, Object[]) (/reference/android/os/AsyncTask#executeOnExecutor(java.util.concurrent.Executor,%20Params[]))

execute(Runnable) (/reference/android/os/AsyncTask#execute(java.lang.Runnable))

execute

Added in API level 11 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public static void execute (Runnable (/reference/java/lang/Runnable) runnable)
```

Convenience version of execute(java.lang.Object) (/reference/android/os/AsyncTask#execute(Params[])) for use with a simple Runnable object. See execute(java.lang.Object[]) (/reference/android/os/AsyncTask#execute(Params[])) for more information on the order of execution.
This method must be called from the main thread of your app.

Parameters	
`runnable`	`Runnable`

See also:

`execute(Object[])` (/reference/android/os/AsyncTask#execute(Params[]))

`executeOnExecutor(java.util.concurrent.Executor, Object[])` (/reference/android/os/AsyncTask#executeOnExecutor(java.util.concurrent.Executor,%20Params[]))

executeOnExecutor

Added in API level 11 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public final AsyncTask (/reference/android/os/AsyncTask)<Params, Progress, Result> exec
                Params... params)
```

Executes the task with the specified parameters. The task returns itself (this) so that the caller can keep a reference to it.

This method is typically used with `THREAD_POOL_EXECUTOR` (/reference/android/os/AsyncTask#THREAD_POOL_EXECUTOR) to allow multiple tasks to run in parallel on a pool of threads managed by AsyncTask, however you can also use your own `Executor` (/reference/java/util/concurrent/Executor) for custom behavior.

Warning: Allowing multiple tasks to run in parallel from a thread pool is generally *not* what one wants, because the order of their operation is not defined. For example, if these tasks are used to modify any state in common (such as writing a file due to a button click), there are no guarantees on the order of the modifications. Without careful work it is possible in rare cases for the newer version of the data to be over-written by an older one, leading to obscure data loss and stability issues. Such changes are best executed in serial; to guarantee such work is serialized regardless of platform version you can use this function with `SERIAL_EXECUTOR` (/reference/android/os/AsyncTask#SERIAL_EXECUTOR).

This method must be invoked on the UI thread.
This method must be called from the main thread of your app.

Parameters	
`exec`	`Executor`: The executor to use. `THREAD_POOL_EXECUTOR` (/reference/android/os/AsyncTask#THREAD_POOL_EXECUTOR) is available as a convenient process-wide thread pool for tasks that are loosely coupled.
`params`	`Params`: The parameters of the task.

Returns	
`AsyncTask` (/reference/android/os/AsyncTask) `<Params, Progress, Result>`	This instance of AsyncTask.

Throws	
`IllegalStateException` (/reference/java/lang/IllegalStateException)	If `getStatus()` (/reference/android/os/AsyncTask#getStatus()) returns either `AsyncTask.Status#RUNNING` (/reference/android/os/AsyncTask.Status#RUNNING) or `AsyncTask.Status#FINISHED` (/reference/android/os/AsyncTask.Status#FINISHED).

See also:

`execute(Object[])` (/reference/android/os/AsyncTask#execute(Params[]))

get

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public final Result get (long timeout,
                TimeUnit (/reference/java/util/concurrent/TimeUnit) unit)
```

Waits if necessary for at most the given time for the computation to complete, and then retrieves its result.

Parameters	
`timeout`	`long`: Time to wait before cancelling the operation.
`unit`	`TimeUnit`: The time unit for the timeout.

Returns	
`Result`	The computed result.

Throws	
`CancellationException` (/reference/java/util/concurrent/CancellationException)	If the computation was cancelled.
`ExecutionException` (/reference/java/util/concurrent/ExecutionException)	If the computation threw an exception.

InterruptedException (/reference/java/lang/InterruptedException)	If the current thread was interrupted while waiting.
TimeoutException (/reference/java/util/concurrent/TimeoutException)	If the wait timed out.

get

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public final Result get ()
```

Waits if necessary for the computation to complete, and then retrieves its result.

Returns

Result	The computed result.

Throws

CancellationException (/reference/java/util/concurrent/CancellationException)	If the computation was cancelled.
ExecutionException (/reference/java/util/concurrent/ExecutionException)	If the computation threw an exception.
InterruptedException (/reference/java/lang/InterruptedException)	If the current thread was interrupted while waiting.

getStatus

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public final AsyncTask.Status (/reference/android/os/AsyncTask.Status) getStatus ()
```

Returns the current status of this task.

Returns	
AsyncTask.Status (/reference/android/os/AsyncTask.Status)	The current status.

isCancelled

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
public final boolean isCancelled ()
```

Returns `true` if this task was cancelled before it completed normally. If you are calling `cancel(boolean)` (/reference/android/os/AsyncTask#cancel(boolean)) on the task, the value returned by this method should be checked periodically from `doInBackground(java.lang.Object[])` (/reference/android/os/AsyncTask#doInBackground(Params[])) to end the task as soon as possible.

Returns	
boolean	`true` if task was cancelled before it completed

See also:

`cancel(boolean)` (/reference/android/os/AsyncTask#cancel(boolean))

Protected methods

doInBackground

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
protected abstract Result doInBackground (Params... params)
```

Override this method to perform a computation on a background thread. The specified parameters are the parameters passed to `execute(Params)` (/reference/android/os/AsyncTask#execute(Params[])) by the caller of this task. This will normally run on a background thread. But to better support testing frameworks, it is recommended that this also tolerates direct execution on the foreground thread, as part of the `execute(Params)` (/reference/android/os/AsyncTask#execute(Params[])) call. This method can call `publishProgress(Progress)` (/reference/android/os/AsyncTask#publishProgress(Progress[])) to publish updates on the UI thread.
This method may take several seconds to complete, so it should only be called from a worker thread.

Parameters	
`params`	`Params`: The parameters of the task.

Returns	
`Result`	A result, defined by the subclass of this task.

See also:

`onPreExecute()` (/reference/android/os/AsyncTask#onPreExecute())

`onPostExecute(Result)` (/reference/android/os/AsyncTask#onPostExecute(Result))

`publishProgress(Progress)` (/reference/android/os/AsyncTask#publishProgress(Progress[]))

onCancelled

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
protected void onCancelled ()
```

Applications should preferably override `onCancelled(java.lang.Object)` (/reference/android/os/AsyncTask#onCancelled(Result)). This method is invoked by the default implementation of `onCancelled(java.lang.Object)` (/reference/android/os/AsyncTask#onCancelled(Result)). The default version does nothing.

Runs on the UI thread after `cancel(boolean)` (/reference/android/os/AsyncTask#cancel(boolean)) is invoked and `doInBackground(java.lang.Object[])` (/reference/android/os/AsyncTask#doInBackground(Params[])) has finished.

This method must be called from the main thread of your app.

See also:

`onCancelled(Object)` (/reference/android/os/AsyncTask#onCancelled(Result))

`cancel(boolean)` (/reference/android/os/AsyncTask#cancel(boolean))

`isCancelled()` (/reference/android/os/AsyncTask#isCancelled())

onCancelled

Added in API level 11 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

Deprecated in API 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
protected void onCancelled (Result result)
```

Runs on the UI thread after `cancel(boolean)` (/reference/android/os/AsyncTask#cancel(boolean)) is invoked and `doInBackground(java.lang.Object[])` (/reference/android/os/AsyncTask#doInBackground(Params[])) has finished.

The default implementation simply invokes `onCancelled()` (/reference/android/os/AsyncTask#onCancelled()) and ignores the result. If you write your own implementation, do not call `super.onCancelled(result)`.

This method must be called from the main thread of your app.

Parameters	
`result`	`Result`: The result, if any, computed in `doInBackground(java.lang.Object[])` (/reference/android/os/AsyncTask#doInBackground(Params[])), can be null

See also:

`cancel(boolean)` (/reference/android/os/AsyncTask#cancel(boolean))

`isCancelled()` (/reference/android/os/AsyncTask#isCancelled())

onPostExecute

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
protected void onPostExecute (Result result)
```

Runs on the UI thread after `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])). The specified result is the value returned by `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])). To better support testing frameworks, it is recommended that this be written to tolerate direct execution as part of the execute() call. The default version does nothing.

This method won't be invoked if the task was cancelled.

This method must be called from the main thread of your app.

Parameters	
`result`	`Result`: The result of the operation computed by `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])).

See also:

`onPreExecute()` (/reference/android/os/AsyncTask#onPreExecute())

`doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[]))

`onCancelled(Object)` (/reference/android/os/AsyncTask#onCancelled(Result))

onPreExecute

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
protected void onPreExecute ()
```

Runs on the UI thread before `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])). Invoked directly by `execute(Params)` (/reference/android/os/AsyncTask#execute(Params[])) or

`executeOnExecutor(Executor, Params)` (/reference/android/os/AsyncTask#executeOnExecutor(java.util.concurrent.Executor,%20Params[])). The default version does nothing.
This method must be called from the main thread of your app.

See also:

`onPostExecute(Result)` (/reference/android/os/AsyncTask#onPostExecute(Result))

`doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[]))

onProgressUpdate

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
protected void onProgressUpdate (Progress... values)
```

Runs on the UI thread after `publishProgress(Progress)` (/reference/android/os/AsyncTask#publishProgress(Progress[])) is invoked. The specified values are the values passed to `publishProgress(Progress)` (/reference/android/os/AsyncTask#publishProgress(Progress[])). The default version does nothing.
This method must be called from the main thread of your app.

Parameters	
`values`	`Progress`: The values indicating progress.

See also:

`publishProgress(Progress)` (/reference/android/os/AsyncTask#publishProgress(Progress[]))

`doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[]))

publishProgress

Added in API level 3 (/guide/topics/manifest/uses-sdk-element#ApiLevels)
Deprecated in API level 30 (/guide/topics/manifest/uses-sdk-element#ApiLevels)

```
protected final void publishProgress (Progress... values)
```

This method can be invoked from `doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[])) to publish updates on the UI thread while the background computation is still running. Each call to this method will trigger the execution of `onProgressUpdate(Progress)` (/reference/android/os/AsyncTask#onProgressUpdate(Progress[])) on the UI thread.
`onProgressUpdate(Progress)` (/reference/android/os/AsyncTask#onProgressUpdate(Progress[])) will not be called if the task has been canceled.
This method may take several seconds to complete, so it should only be called from a worker thread.

Parameters	
`values`	`Progress`: The progress values to update the UI with.

See also:

`onProgressUpdate(Progress)` (/reference/android/os/AsyncTask#onProgressUpdate(Progress[]))

`doInBackground(Params)` (/reference/android/os/AsyncTask#doInBackground(Params[]))

Last updated 2023-04-12 UTC.