A task that **does not return a value** is represented by the **System.Threading.Tasks.Task** class.

 A task that **returns a value** is represented by the **System.Threading.Tasks.Task<TResult>**class, which inherits from [Task](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx). The task object handles the infrastructure details and provides methods and properties that are accessible from the **calling thread** throughout the lifetime of the task.

For example, you can access the **Status**property of a task at any time to determine whether it has started running, ran to completion, was canceled, or has thrown an exception. The status is represented by a **TaskStatus enumeration.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace \_01\_console

{

class Program

{ static void Main(string[] args)

{

Thread.CurrentThread.Name = "Main";

// Create a task and supply a user delegate by using a lambda expression.

Task taskA = new Task(() => Console.WriteLine("Hello from taskA."));

// Start the task.

taskA.Start();

// Output a message from the calling thread.

Console.WriteLine("Hello from thread '{0}'.", Thread.CurrentThread.Name);

//Note call to the Task.Wait method to ensure that the task completes execution before the console mode application ends.

taskA.Wait();

Console.WriteLine("done");

Console.ReadLine();

}

}

}

You can also use the **[Task.Run](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run(v=vs.110).aspx) methods** to **create and start a task in one operation**.

To manage the task, the [Run](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run(v=vs.110).aspx) methods use the **default task scheduler**, regardless of which task scheduler is associated with the current thread. The [Run](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run(v=vs.110).aspx) methods are the preferred way to create and start tasks when **more control** over the creation and scheduling of the task **is not needed**.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace \_01\_console

{

class Program

{ static void Main(string[] args)

{

Thread.CurrentThread.Name = "Main";

// Define and run the task.

Task taskA = Task.Run( () => Console.WriteLine("Hello from taskA."));

// Output a message from the calling thread.

Console.WriteLine("Hello from thread '{0}'.", Thread.CurrentThread.Name);

taskA.Wait();

Console.WriteLine("done");

Console.ReadLine();

}

}

}

You can also use the **[TaskFactory.StartNew](https://msdn.microsoft.com/en-us/library/system.threading.tasks.taskfactory.startnew(v=vs.110).aspx)** method to **create and start** a task in one operation. Use this method when creation and scheduling do not have to be separated and you require additional **task creation options** or the use of a **specific scheduler**, or when you need to pass **additional state** into the task through its [AsyncState](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.asyncstate(v=vs.110).aspx) property,

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace \_01\_console

{

class Program

{

static void Main(string[] args)

{

Thread.CurrentThread.Name = "Main";

// Better: Create and start the task in one operation.

Task taskA = Task.Factory.StartNew(() => Console.WriteLine("Hello from taskA."));

// Output a message from the calling thread.

Console.WriteLine("Hello from thread '{0}'.",

Thread.CurrentThread.Name);

taskA.Wait();

Console.WriteLine("done");

Console.ReadLine();

}

}

}

Starting with the .NET Framework 4.5, the [Task.Run](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run(v=vs.110).aspx) method is the recommended way to launch a compute-bound task. Use the StartNew method only when you require fine-grained control for a long-running, compute-bound task. This includes scenarios in which you want to control the following:

* Task creation options. Tasks created by the [Task.Run](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run(v=vs.110).aspx) method by default are created with the **[TaskCreationOptions.DenyChildAttach](https://msdn.microsoft.com/en-us/library/system.threading.tasks.taskcreationoptions(v=vs.110).aspx)** option. To **override this behavior**, or to provide other **[TaskCreationOptions](https://msdn.microsoft.com/en-us/library/system.threading.tasks.taskcreationoptions(v=vs.110).aspx)** options, call a StartNew overload.
* Parameter passing. The overloads of the [Task.Run](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run(v=vs.110).aspx) method do not allow you to pass a **parameter to the task delegate**. Overloads of theStartNew method do.
* The task scheduler. The overloads of the [Task.Run](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run(v=vs.110).aspx) method use the **default task scheduler**. To control the task scheduler, call a **StartNewoverload with a scheduler parameter**.