System.Threading.Parallel.ParallelLoop<T> Class

```
[ILAsm]
.class public abstract serializable ParallelLoop<T> implements
System.IDisposable

[C#]
public abstract class ParallelLoop<T>: IDisposable
```

Assembly Info:

- *Name:* System.Threading.Parallel
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 2.0.x.x
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- **System.IDisposable**

Summary

A parallel loop over iteration values of type T.

Inherits From: System.Object

Library: Parallel

Thread Safety: All public static members of this type are safe for multithreaded operations. No instance members, unless specifically stated, are guaranteed to be thread safe.

Description

Abstract generic class `System.Threading.Parallel.ParallelLoop<T>` abstracts common behavior of the loop classes that iterate over values of type T. Its derived classes differ in how the iteration space is defined.

Iteration commences once method `System.Threading.Parallel.ParallelLoop<T>.BeginRun` is called. The callback is applied to each iteration value. A conforming implementation can use the thread calling `System.Threading.Parallel.ParallelLoop<T>.BeginRun` to execute all iterations, regardless of the value of `System.Threading.Parallel.ParallelLoop<T>.MaxThreads`. The thread that calls `System.Threading.Parallel.ParallelLoop<T>.BeginRun` shall call method `System.Threading.Parallel.ParallelLoop<T>.EndRun` to block until all iterations complete or are cancelled. When `System.Threading.Parallel.ParallelLoop<T>.EndRun` is called, the calling thread can be employed as a worker thread.

Calling method `System.Threading.Parallel.ParallelLoop<T>.Run` is equivalent to calling `System.Threading.Parallel.ParallelLoop<T>.BeginRun` followed by calling “`System.Threading.Parallel.ParallelLoop<T>.EndRun`.

A parallel loop can be cancelled at any time (even before it starts running) by calling method `System.Threading.Parallel.ParallelLoop<T>.Cancel`. Cancellation is asynchronous in the sense that method `System.Threading.Parallel.ParallelLoop<T>.Cancel` can return while portions of the loop are still running. Any number of threads can call `System.Threading.Parallel.ParallelLoop<T>.Cancel` on the same object. Cancellation affects only iterations that have not yet been issued to worker threads. Outstanding iterations are completed normally.

If one or more invocations of a callback throws an unhandled exception, the rest of the loop is cancelled. One of the exceptions is saved inside the `System.Threading.Parallel.ParallelLoop<T>` until the loop has stopped running, and then the saved exception is rethrown when method `System.Threading.Parallel.ParallelLoop<T>.EndRun` is invoked. In the case of multiple exceptions, the implementation can choose any one of the exceptions to save and rethrow.

ParallelLoop<T>.BeginRun(System.Action<T>) Method

```
[ILAsm]
.method public hidebysig abstract void BeginRun(class System.Action<!0>
action)

[C#]
public abstract void BeginRun(Action<T> action)
```

Summary

Begin executing iterations, applying the action delegate to each iteration value.

Parameters

Parameter	Description
action	The `System.Delegate`to apply to each iteration value.

Description

This method is not thread safe. It should be called only once for a given instance of a `System.Threading.Parallel.ParallelLoop<T>`.

If one or more invocations of a callback throws an unhandled exception, the rest of the loop is cancelled. One of the exceptions is saved inside the `System.Threading.Parallel.ParallelLoop<T>`until the loop has stopped running, and then the saved exception is rethrown when method EndRun is invoked. In the case of multiple exceptions, the implementation can choose any one of the exceptions to save and rethrow.

[*Note:* Implementations, particularly on single-threaded hardware, are free to employ the calling thread to execute all loop iterations.]

[*Note:* The return value is void, not `System.IAsyncResult`, and there is no callBack or stateObject arguments. This departure from the usual asynchronous call pattern (e.g. FileStreamBeginRead) is deliberate, because in typical scenarios the extra complexity would just add pointless burden on the implementation.]

Exceptions

Exception	Condition
System.ArgumentNullException	*action* is `null`.

ParallelLoop<T>.Cancel() Method

[ILAsm]
```
.method public hidebysig abstract virtual void Cancel()
```

[C#]
```
public abstract void Cancel()
```

Summary

Eventually cancel issuance of any further iterations

Description

A `System.Threading.Parallel.ParallelLoop<T>` can be cancelled at any time (even before it starts running) by calling method Cancel. Cancellation is asynchronous in the sense that method Cancel can return while portions of the loop are still running. Any number of threads can concurrently call Cancel on the same object. Cancellation affects only iterations that have not yet been issued to worker threads. Outstanding iterations are completed normally.

ParallelLoop<T>.EndRun() Method

[ILAsm]
```
.method public hidebysig virtual void EndRun()
```

[C#]
```
public void EndRun()
```

Summary

Wait until all iterations are finished (or cancelled).

Description

This method is not thread safe. It should be called exactly once by the thread that called `System.Threading.Parallel.ParallelLoop<T>.BeginRun`.

ParallelLoop<T>.Run(System.Action<T>) Method

```
[ILAsm]
.method public hidebysig virtual abstract void Run(class System.Action<!0>
action)

[C#]
public void Run(Action<T> action)
```

Summary

Start processing of loop iterations and wait until done.

Parameters

Parameter	**Description**
action	The `System.Delegate` applied to each iteration value

Description

This method is equivalent to calling `System.Threading.Parallel.ParallelLoop<T>.BeginRun` followed by calling `System.Threading.Parallel.ParallelLoop<T>.EndRun`.

Exceptions

Exception	**Condition**
System.ArgumentNullException	*action* is `null`.