System.Threading.Parallel.ParallelWhile<T> Class

```
[ILAsm]
.class public sealed serializable ParallelWhile<T> extends
System.Threading.Parallel.ParallelLoop<!0>

[C#]
public sealed class ParallelWhile<T>: ParallelLoop<T>
```

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)

Summary

A parallel while loop over iteration values of type T.

Inherits From: System.Threading.Parallel.ParallelLoop<T>

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

Class `System.Threading.Parallel.ParallelWhile<T>` provides a simple way to establish a pool of work to be distributed among multiple threads, and to wait for the work to complete before proceeding.

A freshly constructed `System.Threading.Parallel.ParallelWhile<T>` has an empty pool of work items. Method `System.Threading.Parallel.ParallelWhile<T>.Add` adds a work item to the pool. Method `System.Threading.Parallel.ParallelWhile<T>.BeginRun` activates processing of the pool. Inherited method `System.Threading.Parallel.ParallelLoop<T>.EndRun` waits until all work in the pool completes. Inherited method `System.Threading.Parallel.ParallelLoop<T>.Run` is a shorthand that combines `System.Threading.Parallel.ParallelWhile<T>.BeginRun` and `System.Threading.Parallel.ParallelLoop<T>.EndRun`. New work can be added to the pool while it is active, hence the class corresponds roughly to a parallel while loop that continually chops away at a (possibly growing) collection until the collection becomes empty. Once the loop is running, implementations are free to make method Add process the work item instead of putting it in the pool, for sake of limiting the size of the work pool. (The pool is typically a small multiple of the number of threads.) Once the pool is activated, one or more worker threads pull work items from the pool and apply the callback to each. The implementation is free to process work items in any order. Inherited method `System.Threading.Parallel.ParallelLoop<T>.EndRun` blocks until the pool is empty and all pending invocations of the callback have returned. An iteration should not cause method `System.Threading.Parallel.ParallelWhile<T>.Add` to be called after the iteration finishes (e.g. by use of yet another thread), otherwise a race condition ensues in which `System.Threading.Parallel.ParallelLoop<T>.EndRun` might return prematurely even though there is more work to be done.

A conforming implementation is allowed to execute serially, by using the thread that calls `System.Threading.Parallel.ParallelWhile<T>.BeginRun`to process all pending work items that are added before BeginRun returns, and using the thread that calls `System.Threading.Parallel.ParallelLoop<T>.EndRun` to process all pending work items that are added after `System.Threading.Parallel.ParallelWhile<T>.BeginRun` returned and before `System.Threading.Parallel.ParallelLoop<T>.EndRun`returns.

ParallelWhile<T>() Constructor

```
[ILAsm]
public rtspecialname specialname instance void .ctor()

[C#]
public ParallelWhile()
```

Summary

Constructs a `System.Threading.Parallel.ParallelWhile<T>` with an initially empty collection of work items.

Description

The loop does not start executing until at least method `System.Threading.Parallel.ParallelWhile<T>.BeginRun` is called and possibly not until method `System.Threading.Parallel.ParallelLoop<T>.EndRun` is called.

ParallelWhile<T>(System.Int32) Constructor

```
[ILAsm]
public rtspecialname specialname instance void .ctor(int32 numThreads)

[C#]
public ParallelWhile(int numThreads)
```

Summary

Constructs a `System.Threading.Parallel.ParallelWhile<T>` with an initially empty collection of work items.

Parameters

Parameter	Description
numThreads	maximum number of threads to use

Description

The loop does not start executing until at least method `System.Threading.Parallel.ParallelWhile<T>.BeginRun` is called and possibly not until method `System.Threading.Parallel.ParallelLoop<T>.EndRun` is called.

If numThreads is 0, then up to `System.Threading.Parallel.ParallelEnvironment.MaxThreads` threads are used instead. The value includes the thread that created the `System.Threading.Parallel.ParallelFor<T>`, hence using numThreads=1 causes sequential execution.

ParallelWhile<T>.Add(T) Method

```
[ILAsm]
.method public hidebysig instance void Add(!0 item)

[C#]
public void Add(T item)
```

Summary

Add a work item.

Parameters

Parameter	Description
item	value for an iteration.

Description

This method can be called before or after method `System.Threading.Parallel.ParallelWhile<T>.BeginRun` is called.

This method is always thread safe.

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

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

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

Summary

Begin processing work items.

Parameters

Parameter	Description
action	The `System.Delegate` that processes each work item.

Description

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

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

Exceptions

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

ParallelWhile<T>.Cancel() Method

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

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

Summary

Cancel any iterations that have not yet started

Description

This method is safe to call concurrently on the same instance.

It does not cancel any future iterations that can be added.

ParallelWhile<T>.EndRun() Method

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

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

Summary

Waits until all iterations are finished (or cancelled). If any of the iterations threw an exception, then one of these exceptions is rethrown.

Description

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