## Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="" alt="AggregateException" /></td>
<td>Represents one or more errors that occur during application execution.</td>
</tr>
<tr>
<td><img src="" alt="Lazy(Of (T)&gt;)" /></td>
<td>Provides support for lazy initialization.</td>
</tr>
</tbody>
</table>
**Delegates**

<table>
<thead>
<tr>
<th>Delegate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Func&lt;(Of&lt;(T1, T2, T3, T4, T5, TResult)&gt;)&gt;</code></td>
<td>Encapsulates a method that has five parameters and returns a value of the type specified by the TResult parameter.</td>
</tr>
</tbody>
</table>

Send [feedback](#) on this topic to Microsoft.
Represents one or more errors that occur during application execution.

**Namespace:** System

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute> _
Public Class AggregateException _
     Inherits Exception

C#

[SerializableAttribute]
public class AggregateException : Exception
Remarks

AggregateException is used to consolidate multiple failures into a single, throwable exception object.
Inheritance Hierarchy

System...:::Object
  System...:::Exception
    System...:::AggregateException
See Also

System Namespace

Send feedback on this topic to Microsoft.
 Visual Basic  □  C#

 □ Include Protected Members
 □ Include Inherited Members

.NET Framework Class Library

AggregateException Constructor

[Links: AggregateException Class  See Also  Send Feedback]
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregateException()()</td>
<td>Initializes a new instance of the AggregateException class.</td>
</tr>
<tr>
<td>AggregateException(IEnumerable&lt;Exception&gt;)[]</td>
<td>Initializes a new instance of the AggregateException class with references to the inner exceptions that are the cause of this exception.</td>
</tr>
<tr>
<td>AggregateException(array&lt;Exception&gt;[][])</td>
<td>Initializes a new instance of the AggregateException class with references to the inner exceptions that are the cause of this exception.</td>
</tr>
<tr>
<td>AggregateException(String)</td>
<td>Initializes a new instance of the AggregateException class with a specified error message.</td>
</tr>
<tr>
<td>AggregateException(SerializationInfo, StreamingContext)</td>
<td>Initializes a new instance of the AggregateException class with serialized data.</td>
</tr>
<tr>
<td>AggregateException(String, IEnumerable&lt;Exception&gt;[])</td>
<td>Initializes a new instance of the AggregateException class with a specified error message and references to</td>
</tr>
</tbody>
</table>
the inner exceptions that are the cause of this exception.

Initializes a new instance of the `AggregateException` class with a specified error message and a reference to the inner exception that is the cause of this exception.

Initializes a new instance of the `AggregateException` class with a specified error message and references to the inner exceptions that are the cause of this exception.
See Also

AggregateException Class
System Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `AggregateException` class.

**Namespace:** `System`  
**Assembly:** `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public AggregateException()
See Also

AggregateException Class
AggregateException Overload
System Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `AggregateException` class with references to the inner exceptions that are the cause of this exception.

**Namespace:** System  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    innerExceptions As IEnumerable(Of Exception) _
)

C#

public AggregateException(
    IEnumerable<Exception> innerExceptions
)

Parameters

innerExceptions
    Type: System.Collections.Generic.IEnumerable(Of Exception)
    The exceptions that are the cause of the current exception.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The <code>innerExceptions</code> argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentException</td>
<td>An element of <code>innerExceptions</code> is null.</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
AggregateException Overload
System Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `AggregateException` class with references to the inner exceptions that are the cause of this exception.

**Namespace:** System

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    ParamArray innerExceptions As Exception() _
)

C#

public AggregateException(
    params Exception[] innerExceptions
)

Parameters

innerExceptions
    Type: array<System.Exception>[]
    The exceptions that are the cause of the current exception.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>The innerExceptions argument is null.</td>
</tr>
<tr>
<td>System...::.ArgumentException</td>
<td>An element of innerExceptions is null.</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
AggregateException Overload
System Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `AggregateException` class with a specified error message.

**Namespace:**  System
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    message As String _
)

C#

public AggregateException(
    string message
)

Parameters

message
    Type: System.String
    The error message that explains the reason for the exception.
See Also

AggregateException Class
AggregateException Overload
System Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `AggregateException` class with serialized data.

**Namespace:** System

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Sub New (_
    info As SerializationInfo, _
    context As StreamingContext _
)

C#

protected AggregateException(
    SerializationInfo info,
    StreamingContext context
)

Parameters

info
    Type: System.Runtime.Serialization...SerializationInfo
    The SerializationInfo that holds the serialized object data about the exception being thrown.

context
    Type: System.Runtime.Serialization...StreamingContext
    The StreamingContext that contains contextual information about the source or destination.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The info argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception could not be deserialized correctly.</td>
</tr>
<tr>
<td>System.Runtime.Serialization..::.SerializationException</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
AggregateException Overload
System Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `AggregateException` class with a specified error message and references to the inner exceptions that are the cause of this exception.

**Namespace:** System  
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Sub New ( _
    message As String, _
    innerExceptions As IEnumerable(Of Exception) _
)
```

### C#

```csharp
public AggregateException(
    string message,
    IEnumerable<Exception> innerExceptions
)
```

## Parameters

**message**

- **Type:** System:::String
- The error message that explains the reason for the exception.

**innerExceptions**

- **Type:** System.Collections.Generic:::IEnumerable<Of <(Exception)>>
- The exceptions that are the cause of the current exception.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>The innerExceptions argument is null.</td>
</tr>
<tr>
<td>System...::.ArgumentException</td>
<td>An element of innerExceptions is null.</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
AggregateException Overload
System Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `AggregateException` class with a specified error message and a reference to the inner exception that is the cause of this exception.

**Namespace:**  System

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New (_
    message As String, _
    innerException As Exception _
)

C#

public AggregateException(
    string message,
    Exception innerException
)

Parameters

message
    Type: System:::String
    The error message that explains the reason for the exception.

innerException
    Type: System:::Exception
    The exception that is the cause of the current exception.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The innerException argument is null.</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
AggregateException Overload
System Namespace

Send feedback on this topic to Microsoft.
Initializing a new instance of the `AggregateException` class with a specified error message and references to the inner exceptions that are the cause of this exception.

**Namespace:** System  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    message As String, _
    ParamArray innerExceptions As Exception() _
)

C#

public AggregateException(
    string message,
    params Exception[] innerExceptions
)

Parameters

message
    Type: System:::.String
    The error message that explains the reason for the exception.

innerExceptions
    Type: array< System:::.Exception >[]()
    The exceptions that are the cause of the current exception.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The innerExceptions argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentException</td>
<td>An element of innerExceptions is null.</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
AggregateException Overload
System Namespace

Send feedback on this topic to Microsoft.
The **AggregateException** type exposes the following members.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Flatten</td>
<td>Flattens an <a href="https://docs.microsoft.com/en-us/dotnet/api/system.aggregateexception">AggregateException</a> instances into a single, new instance.</td>
</tr>
<tr>
<td>GetBaseException</td>
<td>Returns the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.aggregateexception">AggregateException</a> that is the root cause of this exception. (Overrides Exception:::GetBaseException(()().)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetObjectData</td>
<td>Sets the SerializationInfo with information about the exception. (Overrides Exception:::GetObjectData(SerializationInfo, StreamingContext).)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Handle</td>
<td>Invokes a handler on each Exception contained by this <a href="https://docs.microsoft.com/en-us/dotnet/api/system.aggregateexception">AggregateException</a>.</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>Creates and returns a string representation of the current <a href="https://docs.microsoft.com/en-us/dotnet/api/system.aggregateexception">AggregateException</a>. (Overrides Exception:::ToString(()().)</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
System Namespace

Send feedback on this topic to Microsoft.
Flattens an `AggregateException` instances into a single, new instance.

**Namespace:** System
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Flatten As AggregateException

C#

public AggregateException Flatten()

Return Value

A new, flattened AggregateException.
Remarks

If any inner exceptions are themselves instances of `AggregateException`, this method will recursively flatten all of them. The inner exceptions returned in the new `AggregateException` will be the union of all of the inner exceptions from exception tree rooted at the provided `AggregateException` instance.
See Also

AggregateException Class
System Namespace

Send feedback on this topic to Microsoft.
Returns the `AggregateException` that is the root cause of this exception.

**Namespace:** System  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overrides Function GetBaseException As Exception

C#

public override Exception GetBaseException()

Implements

_Exception...:.GetBaseException()()
See Also

AggregateException Class
System Namespace

Send feedback on this topic to Microsoft.
Sets the SerializationInfo with information about the exception.

**Namespace:** System  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overrides Sub GetObjectData ( _
    info As SerializationInfo, _
    context As StreamingContext _
)

C#

public override void GetObjectData(  
    SerializationInfo info,  
    StreamingContext context
)

Parameters

info
    Type: System.Runtime.Serialization.SerializationInfo
    The SerializationInfo that holds the serialized object data about the exception being thrown.

context
    Type: System.Runtime.Serialization.StreamingContext
    The StreamingContext that contains contextual information about the source or destination.

Implements

ISerializable.GetObjectData(SerializationInfo, StreamingContext)
_Exception.Exception.GetObjectData(SerializationInfo, StreamingContext)
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The info argument is null.</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
System Namespace

Send feedback on this topic to Microsoft.
Invokes a handler on each Exception contained by this AggregateException.

**Namespace:** System  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Handle (
    predicate As Func(Of Exception, Boolean) 
)

C#

public void Handle(
    Func<Exception, bool> predicate
)

Parameters

predicate
    Type: System..::.Func<(Of <(Exception, Boolean)>))
The predicate to execute for each exception. The predicate accepts as an argument the Exception to be processed and returns a Boolean to indicate whether the exception was handled.
Remarks

Each invocation of the predicate returns true or false to indicate whether the Exception was handled. After all invocations, if any exceptions went unhandled, all unhandled exceptions will be put into a new AggregateException which will be thrown. Otherwise, the Handle(Func<((Exception, Boolean)>) method simply returns. If any invocations of the predicate throws an exception, it will halt the processing of any more exceptions and immediately propagate the thrown exception as-is.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::AggregateException</code></td>
<td>An exception contained by this AggregateException was not handled.</td>
</tr>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>The predicate argument is null.</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
System Namespace

Send feedback on this topic to Microsoft.
Creates and returns a string representation of the current `AggregateException`.

**Namespace:**  System

**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Overrides Function ToString As String

**C#**

public override string ToString()

**Return Value**

A string representation of the current exception.

**Implements**

_Exception...:::ToString()()
See Also

[AggregateException Class](#)
[System Namespace](#)

Send [feedback](#) on this topic to Microsoft.
The **AggregateException** type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>HelpLink</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>HResult</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>InnerException</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td><strong>InnerExceptions</strong></td>
<td>Gets a read-only collection of the Exception instances that caused the current exception.</td>
</tr>
<tr>
<td>Message</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Source</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>StackTrace</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>TargetSite</td>
<td>(Inherited from Exception.)</td>
</tr>
</tbody>
</table>
See Also

AggregateException Class
System Namespace

Send feedback on this topic to Microsoft.
Gets a read-only collection of the Exception instances that caused the current exception.

**Namespace:** System  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public ReadOnly Property InnerExceptions As ReadOnlyCollection(Of E)
```

**C#**

```csharp
public ReadOnlyCollection<Exception> InnerExceptions { get; }
```
See Also

AggregateException Class
System Namespace

Send feedback on this topic to Microsoft.
Encapsulates a method that has five parameters and returns a value of the type specified by the TResult parameter.

**Namespace:** System

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Delegate Function Func(Of T1, T2, T3, T4, T5, TResult) ( _
    arg1 As T1, _
    arg2 As T2, _
    arg3 As T3, _
    arg4 As T4, _
    arg5 As T5 _
) As TResult

C#

public delegate TResult Func_<T1, T2, T3, T4, T5, TResult>(
    T1 arg1,
    T2 arg2,
    T3 arg3,
    T4 arg4,
    T5 arg5
)

Parameters

arg1
    Type: T1

arg2
    Type: T2

arg3
    Type: T3

arg4
    Type: T4

arg5
    Type: T5
Type Parameters

T1
T2
T3
T4
T5
TResult
See Also

System Namespace

Send feedback on this topic to Microsoft.
Provides support for lazy initialization.

**Namespace:** System

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute> _
<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading := True)> _
Public Class Lazy(Of T)

C#

[SerializableAttribute]
[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading = true)]
public class Lazy<T>
Type Parameters

T

Specifies the type of element being lazily initialized.
Remarks

By default, all public and protected members of Lazy<Of<(T)>) are thread-safe and may be used concurrently from multiple threads. These thread-safety guarantees may be removed optionally and per instance using parameters to the type's constructors.
Inheritance Hierarchy

System...:::Object
System...:::Lazy<(Of <(T)>)>
See Also

System Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

Lazy<Of <(T)>>) Constructor

Lazy<Of <(T)>>) Class

See Also

Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lazy&lt;Of&lt;(T)&gt;&gt;()()</td>
<td>Initializes a new instance of the Lazy&lt;Of&lt;(T)&gt;&gt;() class that uses T's default constructor for lazy initialization.</td>
</tr>
<tr>
<td>Lazy&lt;Of&lt;(T)&gt;&gt;(Boolean)</td>
<td>Initializes a new instance of the Lazy&lt;Of&lt;(T)&gt;&gt;() class that uses T's default constructor and a specified thread-safety mode.</td>
</tr>
<tr>
<td>Lazy&lt;Of&lt;(T)&gt;&gt;(Func&lt;Of&lt;(T)&gt;))</td>
<td>Initializes a new instance of the Lazy&lt;Of&lt;(T)&gt;&gt;() class that uses a specified initialization function.</td>
</tr>
<tr>
<td>Lazy&lt;Of&lt;(T)&gt;&gt;()(LazyThreadSafetyMode)</td>
<td>Initializes a new instance of the Lazy&lt;Of&lt;(T)&gt;&gt;() class that uses T's default constructor and a specified thread-safety mode.</td>
</tr>
<tr>
<td>Lazy&lt;Of&lt;(T)&gt;&gt;(Func&lt;Of&lt;(T)&gt;), Boolean)</td>
<td>Initializes a new instance of the Lazy&lt;Of&lt;(T)&gt;&gt;() class that uses a specified initialization function and a specified thread-safety mode.</td>
</tr>
<tr>
<td>Lazy&lt;Of&lt;(T)&gt;&gt;(Func&lt;Of&lt;(T)&gt;), LazyThreadSafetyMode)</td>
<td>Initializes a new instance of the Lazy&lt;Of&lt;(T)&gt;&gt;() class that uses a specified initialization function and a specified thread-safety mode.</td>
</tr>
</tbody>
</table>
See Also

Lazy(Of (T)> ) Class
System Namespace

Send feedback on this topic to Microsoft.
Lazy<Of<(T)>> Constructor

Initializes a new instance of the Lazy<Of<(T)>> class that uses T's default constructor for lazy initialization.

**Namespace:** System

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public Lazy()
Remarks

An instance created with this constructor may be used concurrently from multiple threads.
See Also

Lazy(Of(T)>) Class
Lazy(Of(T)>) Overload
System Namespace

Send feedback on this topic to Microsoft.
Lazy(Of(Of(T)>)>) Constructor (Boolean)

Initializes a new instance of the Lazy(Of(Of(T)>)>) class that uses T's default constructor and a specified thread-safety mode.

Namespace: System
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Sub New (_
    isThreadSafe As Boolean _
)

**C#**

public Lazy(
    bool isThreadSafe
)

**Parameters**

isThreadSafe
  Type: System..::.Boolean
  true if this instance should be usable by multiple threads concurrently; false if the instance will only be used by one thread at a time.
See Also

Lazy(Of (T)> ) Class
Lazy(Of (T)> ) Overload
System Namespace

Send feedback on this topic to Microsoft.
Lazy<(Of <T>)> Constructor (Func<(Of <T>)>)

Lazy<(Of <T>)> Class  See Also  Send Feedback

Initializes a new instance of the Lazy<(Of <T>)> class that uses a specified initialization function.

Namespace:  System
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    valueFactory As Func(Of T) _
)

C#

public Lazy(
    Func<T> valueFactory
)

Parameters

valueFactory
    Type: System:::Func(Of <T>)(
    The Func(Of <T>()) invoked to produce the lazily-initialized value when it is needed.
Remarks

An instance created with this constructor may be used concurrently from multiple threads.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>valueFactory is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

Lazy(Of (T)> ) Class
Lazy(Of (T)> ) Overload
System Namespace

Send feedback on this topic to Microsoft.
Lazy<(Of (T)>) Constructor (LazyThreadSafetyMode)

Lazy<(Of (T)>) Class  See Also  Send Feedback

Initializes a new instance of the Lazy<(Of (T)>) class that uses T's default constructor and a specified thread-safety mode.

Namespace:  System
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New (_
    mode As LazyThreadSafetyMode _
)

C#

public Lazy(
    LazyThreadSafetyMode mode
)

Parameters

mode
    Type: System.Threading::$_::LazyThreadSafetyMode
    The lazy thread-safety mode mode
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.Diagnostics.ArgumentOutOfRangeException</code></td>
<td>mode mode contains an invalid valuee</td>
</tr>
</tbody>
</table>
See Also

Lazy(Of (T)>) Class
Lazy(Of (T)>) Overload
System Namespace

Send feedback on this topic to Microsoft.
Lazy<Of <(T)>>(Of <(T)>) Constructor (Func<Of <(T)>>, Boolean)

Lazy<Of <(T)>>(Of <(T)>) Class  See Also  Send Feedback

Initializes a new instance of the Lazy<Of <(T)>> class that uses a specified initialization function and a specified thread-safety mode.

Namespace:  System
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    valueFactory As Func(Of T), _
    isThreadSafe As Boolean _
)

C#

public Lazy(
    Func<T> valueFactory,
    bool isThreadSafe
)

Parameters

valueFactory
    Type: System...:Func(Of (Of (T)>))
    The Func(Of (Of (T)>)) invoked to produce the lazily-initialized value when it is needed.

isThreadSafe
    Type: System...:Boolean
    true if this instance should be usable by multiple threads concurrently; false if the instance will only be used by one thread at a time.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>valueFactory is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

Lazy(Of (T)> ) Class
Lazy(Of (T)> ) Overload
System Namespace

Send feedback on this topic to Microsoft.
Lazy(Of T>) Constructor (Func(Of T>), LazyThreadSafetyMode)

Lazy(Of T>) Class  See Also  Send Feedback

Initializes a new instance of the Lazy(Of T>) class that uses a specified initialization function and a specified thread-safety mode.

Namespace:  System
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    valueFactory As Func(Of T), _
    mode As LazyThreadSafetyMode _
)

C#

public Lazy(
    Func<T> valueFactory,
    LazyThreadSafetyMode mode
)

Parameters

valueFactory
Type: System::Func(Of <(T)>)
The Func(Of <(T)>) invoked to produce the lazily-initialized value when it is needed.

mode
Type: System.Threading::LazyThreadSafetyMode
The lazy thread-safety mode.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>valueFactory is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>mode mode contains an invalid value.</td>
</tr>
</tbody>
</table>
See Also

Lazy(Of T>) Class
Lazy(Of T>) Overload
System Namespace

Send feedback on this topic to Microsoft.
The **Lazy(Of <T>)** type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>ToString</strong></td>
<td>Creates and returns a string representation of this instance.</td>
</tr>
<tr>
<td></td>
<td>(Overrides Object..::.ToString()().)</td>
</tr>
</tbody>
</table>
See Also

Lazy<Of<T>> Class
System Namespace

Send feedback on this topic to Microsoft.
Lazy(Of Of(T)>)::.ToString Method

Lazy(Of Of(T)>) Class
See Also
Send Feedback

Creates and returns a string representation of this instance.

Namespace: System
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overrides Function ToString As String

C#

public override string ToString() 

Return Value

The result of calling ToString() on the Value.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::NullReferenceException</td>
<td>The <strong>Value</strong> is null.</td>
</tr>
</tbody>
</table>
See Also

Lazy<Of <(T)> Class
System Namespace

Send feedback on this topic to Microsoft.
Visual Basic ⊕  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
Lazy<Of (T>)> Properties
Lazy<Of (T>)> Class  See Also  Send Feedback

The Lazy<Of (T>)> type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsValueCreated</td>
<td>Gets a value indicating whether the Lazy&lt;Of &lt;(T)?&gt;&gt; has been initialized.</td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>
See Also

Lazy<Of<(T)> Class
System Namespace

Send feedback on this topic to Microsoft.
Lazy<(Of <(T)>)>::IsValueCreated Property

Gets a value indicating whether the Lazy<(Of <(T)>)> has been initialized.

**Namespace:** System

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsValueCreated As Boolean

C#

public bool IsValueCreated { get; }

Field Value

true if the Lazy<(Of <(T)>)> instance has been initialized; otherwise, false.
Remarks

The initialization of a Lazy<Of <(T)>> instance may result in either a value being produced or an exception being thrown. If an exception goes unhandled during initialization, IsValueCreated will return false.
See Also

Lazy(Of (T)->) Class
System Namespace

Send feedback on this topic to Microsoft.
Lazy(Of (Of T)>)::.Value Property

Namespace: System
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Value As T

C#

public T Value { get; }
See Also

Lazy(Of (T)> Class
System Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
.NET Framework Class Library
System.Collections.Concurrent Namespace
Send Feedback
# Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BlockingCollection</strong>&lt;Of <em>(T)&gt;</em></td>
<td>Provides blocking and bounding capabilities for thread-safe collections that implement IProducerConsumerCollection&lt;Of <em>(T)&gt;.</em></td>
</tr>
<tr>
<td><strong>ConcurrentBag</strong>&lt;Of <em>(T)&gt;</em></td>
<td>Represents an thread-safe, unordered collection of objects.</td>
</tr>
<tr>
<td><strong>ConcurrentDictionary</strong>&lt;Of <em>(TKey, TValue)&gt;</em></td>
<td>Represents a thread-safe collection of keys and values.</td>
</tr>
<tr>
<td><strong>ConcurrentQueue</strong>&lt;Of <em>(T)&gt;</em></td>
<td>Represents a thread-safe first-in, first-out collection of objects.</td>
</tr>
<tr>
<td><strong>ConcurrentStack</strong>&lt;Of <em>(T)&gt;</em></td>
<td>Represents a thread-safe last-in, first-out collection of objects.</td>
</tr>
<tr>
<td><strong>OrderablePartitioner</strong>&lt;Of <em>(TSource)&gt;</em></td>
<td>Represents a particular manner of splitting an orderable data source into multiple partitions.</td>
</tr>
<tr>
<td><strong>Partitioner</strong></td>
<td>Provides common partitioning strategies for arrays, lists, and enumerables.</td>
</tr>
<tr>
<td><strong>Partitioner</strong>&lt;Of <em>(TSource)&gt;</em></td>
<td>Represents a particular manner of splitting a data source into multiple partitions.</td>
</tr>
<tr>
<td>Interface</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>IProducerConsumerCollection*(Of T)*</td>
<td>Defines methods to manipulate thread-safe collections intended for producer/consumer usage.</td>
</tr>
</tbody>
</table>

Send feedback on this topic to Microsoft.
Provides blocking and bounding capabilities for thread-safe collections that implement IProducerConsumerCollection\<(Of \<(T)\)>\).

**Namespace:**  [System.Collections.Concurrent](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

```vbnet
<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
    ExternalThreading := True)> _
Public Class BlockingCollection(Of T) _
    Implements IEnumerable(Of T), ICollection, IEnumerator, _
    IDisposable
```

C#

```csharp
[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
    ExternalThreading = true)]
public class BlockingCollection<T> : IEnumerable<T>,
    ICollection, IEnumerator, IDisposable
```
Type Parameters

T

Specifies the type of elements in the collection.
Remarks

IProducerConsumerCollection<Of <(T)>>) represents a collection that allows for thread-safe adding and removing of data. BlockingCollection<Of <(T)>>) is used as a wrapper for an IProducerConsumerCollection<Of <(T)>>) instance, allowing removal attempts from the collection to block until data is available to be removed. Similarly, a BlockingCollection<Of <(T)>>) can be created to enforce an upper-bound on the number of data elements allowed in the IProducerConsumerCollection<Of <(T)>>); addition attempts to the collection may then block until space is available to store the added items. In this manner, BlockingCollection<Of <(T)>>) is similar to a traditional blocking queue data structure, except that the underlying data storage mechanism is abstracted away as an IProducerConsumerCollection<Of <(T)>>).
Inheritance Hierarchy

System..::.Object
System.Collections.Concurrent..::.BlockingCollection(Of T)
See Also

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#  Include Protected Members  Include Inherited Members  .NET Framework Class Library  BlockingCollection<Of <(T)>> Constructor  BlockingCollection<Of <(T)>> Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;()()</code></td>
<td>Initializes a new instance of the <code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;()</code> class without an upper-bound.</td>
</tr>
<tr>
<td><code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;(IProducerConsumerCollection&lt;Of &lt;(T)&gt;&gt;)</code></td>
<td>Initializes a new instance of the <code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;()</code> class without an upper-bound and using the provided <code>IProducerConsumerCollection&lt;Of &lt;(T)&gt;&gt;()</code> as its underlying data store.</td>
</tr>
<tr>
<td><code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;(Int32)</code></td>
<td>Initializes a new instance of the <code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;()</code> class with the specified upper-bound.</td>
</tr>
<tr>
<td><code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;(IProducerConsumerCollection&lt;Of &lt;(T)&gt;&gt;, Int32)</code></td>
<td>Initializes a new instance of the <code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;()</code> class with the specified upper-bound and using the provided <code>IProducerConsumerCollection&lt;Of &lt;(T)&gt;&gt;()</code> as its underlying data store.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `BlockingCollection(Of <T>)` class without an upper-bound.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public BlockingCollection()
Remarks

The default underlying collection is a ConcurrentQueue<T>.
See Also

BlockingCollection(Of (T)) Class
BlockingCollection(Of (T)) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the BlockingCollection(Of (T)) class without an upper-bound and using the provided IProducerConsumerCollection(Of (T)) as its underlying data store.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New (_
    collection As IProducerConsumerCollection(Of T) _
)

C#

public BlockingCollection(
    IProducerConsumerCollection<T> collection
)

Parameters

collection
    Type: System.Collections.Concurrent..::.IProducerConsumerCollection(Of T)
    The collection to use as the underlying data store.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The collection argument is null.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
BlockingCollection(Of T) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the BlockingCollection(Of T) class with the specified upper-bound.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    boundedCapacity As Integer _
)

C#

public BlockingCollection(
    int boundedCapacity
)

Parameters

boundedCapacity
    Type: System::Int32
    The bounded size of the collection.
Remarks

The default underlying collection is a `ConcurrentQueue<T>`.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The boundedCapacity is not a positive value.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
BlockingCollection(Of (T)) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the BlockingCollection(Of <(T)>>) class with the specified upper-bound and using the provided IProducerConsumerCollection(Of <(T)>>) as its underlying data store.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( 
    collection As IProducerConsumerCollection(Of T), _
    boundedCapacity As Integer _
)

C#

public BlockingCollection( 
    IProducerConsumerCollection<T> collection, 
    int boundedCapacity 
)

Parameters

collection
    Type: System.Collections.Concurrent.IProducerConsumerCollection<T>
    The collection to use as the underlying data store.

boundedCapacity
    Type: System.Int32
    The bounded size of the collection.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The collection argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The boundedCapacity is not a positive value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The supplied collection contains more values than is permitted by boundedCapacity.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
BlockingCollection(Of T) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The **BlockingCollection(Of T)<>** type exposes the following members.
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>AddToAny</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>CompleteAdding</strong></td>
<td>Marks the BlockingCollection&lt;Of &lt;(T)▷) instances as not accepting any more additions.</td>
</tr>
<tr>
<td><strong>CopyTo</strong></td>
<td>Copies all of the items in the BlockingCollection&lt;Of &lt;(T)▷) instance to a compatible one-dimensional array, starting at the specified index of the target array.</td>
</tr>
<tr>
<td><strong>Dispose</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Equals</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>Finalize</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetConsumingEnumerable</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>GetHashCode</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetType</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>MemberwiseClone</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>Take</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>TakeFromAny</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>ToArray</strong></td>
<td>Copies the items from the BlockingCollection&lt;Of &lt;(T)▷) instance into a new array.</td>
</tr>
<tr>
<td><strong>ToString</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>TryAdd</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>TryAddToAny</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>TryTake</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>TryTakeFromAny</strong></td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICollection::CopyTo</td>
<td>Copies all of the items in the BlockingCollection&lt;Of &lt;(T)&gt;&gt;() instance to a compatible one-dimensional array, starting at the specified index of the target array.</td>
</tr>
<tr>
<td>IEnumerable&lt;Of &lt;(T)&gt;&gt;::GetEnumerator</td>
<td>Provides an IEnumerator&lt;Of &lt;(T)&gt;&gt;() for items in the collection.</td>
</tr>
<tr>
<td>IEnumerable::GetEnumerator</td>
<td>Provides an IEnumerator for items in the collection.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Copies all of the items in the `BlockingCollection(Of T)` instance to a compatible one-dimensional array, starting at the specified index of the target array.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Private Sub CopyTo ( _
    array As Array, _
    index As Integer _
) Implements ICollection.CopyTo

C#

void ICollection.CopyTo(
    Array array,
    int index
)

**Parameters**

array
  Type: System..::.Array
  The one-dimensional array that is the destination of the elements copied from the BlockingCollection<Of <T>> instance. The array must have zero-based indexing.

index
  Type: System..::.Int32
  The zero-based index in array at which copying begins.

**Implements**

ICollection..::.CopyTo(Array, Int32)
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The array argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The index argument is less than zero.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The index argument is equal to or greater than the length of the array,</td>
</tr>
<tr>
<td></td>
<td>the array is multidimensional, or the type parameter for the collection</td>
</tr>
<tr>
<td></td>
<td>cannot be cast automatically to the type of the destination array.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of (T)&gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Provides an IEnumerator<(Of <(T)>)> for items in the collection.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function GetEnumerator As IEnumerator(Of T)
    Implements IEnumerable(Of T).GetEnumerator

C#

IEnumerator<T> IEnumerable<T>.GetEnumerator()

Return Value

An IEnumerator(Of <T>) for the items in the collection.

Implements

IEnumerable(Of <T>).GetEnumerator()
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The BlockingCollection(Of (T)&gt;)) has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Provides an IEnumerator for items in the collection.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function GetEnumerator As IEnumerator
Implements IEnumerable.GetEnumerator

C#

IEnumerator IEnumerable.GetEnumerator()

Return Value

An IEnumerator for the items in the collection.

Implements

IEnumerable...::GetEnumera1tor00()
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
BlockingCollection(Of (Of T>)>::.::Add Method
BlockingCollection(Of (Of T>)> Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Add(T)</code></td>
<td>Adds the item to the <code>BlockingCollection&lt;Of (T)&gt;</code></td>
</tr>
<tr>
<td><code>Add(T, CancellationToken)</code></td>
<td>Adds the item to the <code>BlockingCollection&lt;Of (T)&gt;</code>). A <code>OperationCanceledException</code> is thrown if the <code>CancellationToken</code> is canceled.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
.NET Framework Class Library
BlockingCollection<(Of <(T)>)>..::..Add Method (T)
BlockingCollection<(Of <(T)>)> Class  See Also  Send Feedback

Adds the item to the BlockingCollection<(Of <(T)>)>.

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Add ( _
    item As T _
)  

C#

public void Add(
    T item
)

Parameters

item
    Type: T
    The item to be added to the collection. The value can be a null reference.
Remarks

If a bounded capacity was specified when this instance of BlockingCollection<(Of <(T)>)> was initialized, a call to Add may block until space is available to store the provided item.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The BlockingCollection&lt;(Of &lt;T&gt;)&gt; has been marked as complete with regards to additions.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;T&gt;)&gt; has been disposed.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The underlying collection didn't accept the item.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
Add Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library

BlockingCollection(Of T)::.Add Method (T, CancellationToken)

BlockingCollection(Of T) Class  See Also  Send Feedback

Adds the item to the BlockingCollection(Of T). A OperationCanceledException is thrown if the CancellationToken is canceled.

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Add ( _
    item As T, _
    cancellationToken As CancellationToken _
)  

C#

public void Add(
    T item,
    CancellationToken cancellationToken
)  

Parameters

item
    Type: T
    The item to be added to the collection. The value can be a null reference.

cancellationToken
    Type: System.Threading.CancellationToken
    A cancellation token to observe.
Remarks

If a bounded capacity was specified when this instance of BlockingCollection(Of (Of (T)>) was initialized, a call to Add(T, CancellationToken) may block until space is available to store the provided item.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.OperationCanceledException</td>
<td>If the <a href="#">CancellationToken</a> is canceled.</td>
</tr>
<tr>
<td>System...::.InvalidOperationException</td>
<td>The BlockingCollection&lt;Of &lt;(T)&gt;&gt;) has been marked as complete with regards to additions.</td>
</tr>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of &lt;(T)&gt;&gt;) has been disposed.</td>
</tr>
<tr>
<td>System...::.InvalidOperationException</td>
<td>The underlying collection didn't accept the item.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
Add Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
BlockingCollection<(Of <(T)>)>..AddToAny Method
BlockingCollection<(Of <(T)>)> Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddToAny(array&lt;BlockingCollection&lt;Of (T)&gt;, []], T)</td>
<td>Adds the specified item to any one of the specified BlockingCollection&lt;Of (T)&gt; instances.</td>
</tr>
<tr>
<td>AddToAny(array&lt;BlockingCollection&lt;Of (T)&gt;, []], T, CancellationToken)</td>
<td>Adds the specified item to any one of the specified BlockingCollection&lt;Of (T)&gt; instances. A OperationCanceledException is thrown if the CancellationToken is canceled.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library
BlockingCollection(Of T)::.AddToAny Method
(array<BlockingCollection(Of T)>[][], T)

BlockingCollection(Of T) Class  See Also  Send Feedback

Adds the specified item to any one of the specified BlockingCollection(Of T) instances.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function AddToAny ( _
  collections As BlockingCollection(Of T)(), _
  item As T _
) As Integer

C#

public static int AddToAny(
    BlockingCollection<T>[] collections,
    T item
)

Parameters

collections
  Type: array< System.Collections.Concurrent..; BlockingCollection<T>[]>
  The array of collections.

type
  Type: T
  The item to be added to one of the collections.

Return Value

The index of the collection in the collections array to which the item was added.
Remarks

If a bounded capacity was specified when all of the BlockingCollection<Of <(T)>>) instances were initialized, a call to AddToAny may block until space is available in one of the collections to store the provided item.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The collections argument is a 0-length array or contains a null element, or at least one of collections has been marked as complete for adding.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>At least one of the BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; instances has been disposed.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>At least one underlying collection didn't accept the item.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection<(Of <(T)>)> Class
AddToAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
BlocksCollection<(Of <(T)>)[...]AddToAny Method
(array<BlocksCollection<(Of <(T)>)[...]>[], T, CancellationToken)
BlocksCollection<(Of <(T)>)[] Class  See Also  Send Feedback

Adds the specified item to any one of the specified BlocksCollection<(Of<br>(T)>)[...] instances. A OperationCanceledException is thrown if the
CancellationToken is canceled.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function AddToAny (_
collections As BlockingCollection(Of T)(), _
item As T, _
cancellationToken As CancellationToken _
) As Integer

C#

public static int AddToAny(
    BlockingCollection<T>[] collections,
    T item,
    CancellationToken cancellationToken
)

Parameters

collections
  Type: array< System.Collections.Concurrent..::.BlockingCollection<Of
  <(T)>) >[]
  The array of collections.

item
  Type: T
  The item to be added to one of the collections.

cancellationToken
  Type: System.Threading..::.CancellationToken
  A cancellation token to observe.

Return Value

The index of the collection in the collections array to which the item was added.
Remarks

If a bounded capacity was specified when all of the BlockingCollection<Of <(T)>>) instances were initialized, a call to AddToAny may block until space is available in one of the collections to store the provided item.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>If the <code>CancellationToken</code> is canceled.</td>
</tr>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td></td>
<td>The collections argument is a 0-length array or contains a null element,</td>
</tr>
<tr>
<td></td>
<td>or at least one of the collections has been marked as complete for adding.</td>
</tr>
<tr>
<td><code>System:::ArgumentException</code></td>
<td>At least one of the <code>BlockingCollection&lt;Of &lt;(T)&gt;&gt;</code> instances has been</td>
</tr>
<tr>
<td></td>
<td>disposed.</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>At least one underlying collection didn't accept the item.</td>
</tr>
<tr>
<td><code>System:::ArgumentOutOfRangeException</code></td>
<td>The count of collections is greater than the maximum size of 62 for STA</td>
</tr>
<tr>
<td></td>
<td>and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
AddToAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Marks the `BlockingCollection<Of <(T)>)` instances as not accepting any more additions.

**Namespace:**  [System.Collections.Concurrent](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub CompleteAdding

C#

public void CompleteAdding()
Remarks

After a collection has been marked as complete for adding, adding to the collection is not permitted and attempts to remove from the collection will not wait when the collection is empty.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of &lt;(T)&gt;) has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection<Of <(T)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Copies all of the items in the `BlockingCollection(Of T)`) instance to a compatible one-dimensional array, starting at the specified index of the target array.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub CopyTo (_
array As T(), _
index As Integer _
)

C#

public void CopyTo(  
    T[] array,  
    int index
)

Parameters

array
Type: array<T>[]()[]  
The one-dimensional array that is the destination of the elements copied from the BlockingCollection(Of<T>()) instance. The array must have zero-based indexing.

index
Type: System:::Int32  
The zero-based index in array at which copying begins.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The array argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The index argument is less than zero.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The index argument is equal to or greater than the length of the array.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#

Include Protected Members
Include Inherited Members

.NET Framework Class Library

BlockingCollection(Of (T))::.Dispose Method

BlockingCollection(Of (T)) Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose()</td>
<td>Releases resources used by the BlockingCollection&lt;Of &lt;(T)&gt;&gt;) instance.</td>
</tr>
<tr>
<td>Dispose(Boolean)</td>
<td>Releases resources used by the BlockingCollection&lt;Of &lt;(T)&gt;&gt;) instance.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Releases resources used by the BlockingCollection<(Of <(T)>)> instance.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Dispose

C#

public void Dispose()

Implements

IDisposable:::Dispose(void)
See Also

BlockingCollection(Of T) Class
Dispose Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Releases resources used by the BlockingCollection<Of <T>> instance.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Protected Overridable Sub Dispose (_
       disposing As Boolean _
)
```

**C#**

```csharp
protected virtual void Dispose(
       bool disposing
)
```

**Parameters**

disposing
   Type: System..::.Boolean
   Whether being disposed explicitly (true) or due to a finalizer (false).
See Also

BlockingCollection(Of (T)) Class
Dispose Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
BlockingCollection(Of T)::.GetConsumingEnumerable Method
BlockingCollection(Of T) Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GetConsumingEnumerable()</code></td>
<td>Provides a consuming <code>IEnumerable&lt;Of (Of T)&gt;</code> for items in the collection.</td>
</tr>
<tr>
<td><code>GetConsumingEnumerable(CancellationToken)</code></td>
<td>Provides a consuming <code>IEnumerable&lt;Of (Of T)&gt;</code> for items in the collection. Calling <code>MoveNext</code> on the returned enumerable will block if there is no data available, or will throw an <code>OperationCanceledException</code> if the <code>CancellationToken</code> is canceled.</td>
</tr>
</tbody>
</table>
See Also

[BlockingCollection(Of T) Class](System.Collections.Concurrent Namespace)

Send [feedback](#) on this topic to Microsoft.
Provides a consuming IEnumerable<(Of <(T)>)> for items in the collection.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function GetConsumingEnumerable As IEnumerable(Of T)

C#

public IEnumerable<T> GetConsumingEnumerable()

Return Value

An IEnumerable<(Of <(T)>)> that removes and returns items from the collection.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of &lt;(T)&gt;) has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
GetConsumingEnumerable Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Provides a consuming IEnumerable<(Of <(T)>)> for items in the collection. Calling MoveNext on the returned enumerable will block if there is no data available, or will throw an OperationCanceledException if the CancellationToken is canceled.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function GetConsumingEnumerable ( _
    cancellationToken As CancellationToken _
) As IEnumerable(Of T)
```

### C#

```csharp
public IEnumerable<T> GetConsumingEnumerable(
    CancellationToken cancellationToken
)
```

### Parameters

`cancellationToken`  
Type: System.Threading.CancellationToken  
A cancellation token to observe.

### Return Value

An IEnumerable<(T)> that removes and returns items from the collection.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; has been disposed.</td>
</tr>
<tr>
<td>System...::.OperationCanceledException</td>
<td>If the <a href="#">CancellationToken</a> is canceled.</td>
</tr>
</tbody>
</table>
See Also

[BlockingCollection(Of (T))](System.Collections.Concurrent) Class
[GetConsumingEnumerable Overload](System.Collections.Concurrent) Namespace

Send [feedback](#) on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
BlockingCollection(Of (T))::.Take Method
BlockingCollection(Of (T)) Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take()()</td>
<td>Takes an item from the BlockingCollection&lt;Of &lt;(T)&gt;).</td>
</tr>
<tr>
<td>Take(CancellationToken)</td>
<td>Takes an item from the BlockingCollection&lt;Of &lt;(T)&gt;).</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection<Of (T)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Takes an item from the BlockingCollection<Of <T>>.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Take As T

C#

public T Take()

Return Value

The item removed from the collection.
Remarks

A call to Take()() may block until an item is available to be removed.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The <code>BlockingCollection&lt;Of (Of &lt;T&gt;)&gt;</code> is empty and has been marked as complete with regards to additions.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The <code>BlockingCollection&lt;Of (Of &lt;T&gt;)&gt;</code> has been disposed.</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>The underlying collection was modified outside of this <code>BlockingCollection&lt;Of (Of &lt;T&gt;)&gt;</code> instance.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
Take Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Takes an item from the `BlockingCollection(Of T)`.  

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Public Function Take ( _
cancellationToken As CancellationToken _
) As T

C#

public T Take(
    CancellationToken cancellationToken
)

Parameters

cancellationToken
    Type: System.Threading:::CancellationToken

Return Value

The item removed from the collection.
Remarks

A call to Take(CancellationToken) may block until an item is available to be removed.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::OperationCanceledException</td>
<td>If the <a href="#">CancellationToken</a> is canceled or the BlockingCollection&lt;Of &lt;(T)&gt;&gt;) is empty and has been marked as complete with regards to additions.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of &lt;(T)&gt;&gt;) has been disposed.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The underlying collection was modified outside of this BlockingCollection&lt;Of &lt;(T)&gt;&gt;) instance.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
Take Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
BlockingCollection(Of T)::.TakeFromAny Method

BlockingCollection(Of T) Class
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TakeFromAny(array&lt;BlockingCollection&lt;T&gt;&gt;[], T%)</code></td>
<td>Takes an item from any one of the specified <code>BlockingCollection&lt;T&gt;</code> instances.</td>
</tr>
<tr>
<td><code>TakeFromAny(array&lt;BlockingCollection&lt;T&gt;&gt;[], T%, CancellationToken)</code></td>
<td>Takes an item from any one of the specified <code>BlockingCollection&lt;T&gt;</code> instances. A <code>OperationCanceledException</code> is thrown if the <code>CancellationToken</code> is canceled.</td>
</tr>
</tbody>
</table>
See Also

**BlockingCollection**< Of *(T)* > Class
System.Collections.Concurrent Namespace

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Takes an item from any one of the specified `BlockingCollection<(Of <T>)>`) instances.

**Namespace:** [System.Collections.Concurrent](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TakeFromAny ( _
collections As BlockingCollection(Of T)(), _
<OutAttribute> ByRef item As T _
) As Integer

C#

public static int TakeFromAny(
    BlockingCollection<T>[] collections,
    out T item
)

Parameters

collections
    Type: array< System.Collections.Concurrent..:::BlockingCollection<Of
    <(T)>[])>[]]
    The array of collections.

item
    Type: T %
    The item removed from one of the collections.

Return Value

The index of the collection in the collections array from which the item was removed, or -1 if an item could not be removed.
Remarks

A call to TakeFromAny may block until an item is available to be removed.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The collections argument is a 0-length array or contains a null element.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>At least one of the BlockingCollection&lt;(Of &lt;T&gt;)&gt; instances has been disposed.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>At least one of the underlying collections was modified outside of its BlockingCollection&lt;(Of &lt;T&gt;)&gt; instance.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
TakeFromAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

BlockingCollection(Of T)::.TakeFromAny Method
(array<BlockingCollection(Of T)>[][], T%, CancellationToken)

BlockingCollection(Of T) Class  See Also  Send Feedback

Takes an item from any one of the specified BlockingCollection(Of T) instances. A OperationCanceledException is thrown if the CancellationToken is canceled.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TakeFromAny ( _
collections As BlockingCollection(Of T)(), _
<OutAttribute> ByRef item As T, _
cancellationToken As CancellationToken _
) As Integer

C#

public static int TakeFromAny(
    BlockingCollection<T>[] collections,
    out T item,
    CancellationToken cancellationToken
)

Parameters

collections
    Type: array< System.Collections.Concurrent..::..BlockingCollection<Of
    <(T)>) >[]
    The array of collections.

type
    Type: T %
    The item removed from one of the collections.

cancellationToken
    Type: System.Threading...::..CancellationToken
    A cancellation token to observe.

Return Value

The index of the collection in the collections array from which the item was
removed, or -1 if an item could not be removed.
Remarks

A call to TakeFromAny may block until an item is available to be removed.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>OperationCanceledException</td>
<td>If the <a href="#">CancellationToken</a> is canceled.</td>
</tr>
<tr>
<td>ArgumentException</td>
<td>The collections argument is a 0-length array or contains a null element.</td>
</tr>
<tr>
<td>ObjectDisposedException</td>
<td>At least one of the BlockingCollection&lt;Of (T)&gt; instances has been disposed.</td>
</tr>
<tr>
<td>InvalidOperationException</td>
<td>At least one of the underlying collections was modified outside of its BlockingCollection&lt;Of (T)&gt; instance.</td>
</tr>
<tr>
<td>ArgumentOutOfRangeException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (Of T>)>) Class
TakeFromAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Copies the items from the `BlockingCollection<Of <(T)>>` instance into a new array.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ToArray As T()

C#

public T[] ToArray()

Return Value

An array containing copies of the elements of the collection.
Remarks

The copied elements are not removed from the collection.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of &lt;(T)&gt;) has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection\<(Of \<(T)\)>\) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
BlockingCollection(Of T) TryAdd Method
BlockingCollection(Of T) Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TryAdd(T)</strong></td>
<td>Attempts to add the specified item to the BlockingCollection&lt;Of &lt;(T)&gt;&gt;.</td>
</tr>
<tr>
<td><strong>TryAdd(T, Int32)</strong></td>
<td>Attempts to add the specified item to the BlockingCollection&lt;Of &lt;(T)&gt;&gt;.</td>
</tr>
<tr>
<td><strong>TryAdd(T, TimeSpan)</strong></td>
<td>Attempts to add the specified item to the BlockingCollection&lt;Of &lt;(T)&gt;&gt;.</td>
</tr>
<tr>
<td><strong>TryAdd(T, Int32, CancellationToken)</strong></td>
<td>OperationCanceledException is thrown if the CancellationToken is canceled.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add the specified item to the `BlockingCollection<Of <(T)>(T)>()`.

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
### Syntax

**Visual Basic (Declaration)**

Public Function TryAdd ( _  
            item As T _  
        ) As Boolean

**C#**

```csharp
public bool TryAdd(
        T item
    )
```

### Parameters

- **item**
  - Type: `T`
  - The item to be added to the collection.

### Return Value

true if the item could be added; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The BlockingCollection&lt;(Of &lt;T&gt;)&gt; has been marked as complete with regards to additions.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;T&gt;)&gt; has been disposed.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The underlying collection didn't accept the item.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection<(Of (T)> ) Class
TryAdd Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add the specified item to the BlockingCollection<(Of <(T)>)>.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryAdd ( _
    item As T, _
    millisecondsTimeout As Integer _
) As Boolean

C#

public bool TryAdd(
    T item,
    int millisecondsTimeout
)

Parameters

item
    Type: T
    The item to be added to the collection.

millisecondsTimeout
    Type: System..::.Int32
    The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

Return Value

true if the item could be added to the collection within the alloted time;
otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; has been marked as complete with regards to additions.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; has been disposed.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The underlying collection didn't accept the item.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection<(Of <(T)>)> Class
TryAdd Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add the specified item to the BlockingCollection<Of <(T)>).

**Namespace:** System.Collections.Concurrent
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function TryAdd (  
    item As T,  
    timeout As TimeSpan  
) As Boolean
```

### C#

```csharp
public bool TryAdd(  
    T item,  
    TimeSpan timeout
)
```

## Parameters

**item**

Type: `T`

The item to be added to the collection.

**timeout**

Type: `System:::TimeSpan`

A `TimeSpan` that represents the number of milliseconds to wait, or a `TimeSpan` that represents -1 milliseconds to wait indefinitely.

## Return Value

true if the item could be added to the collection within the alloted time; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::::InvalidOperationException</td>
<td>The BlockingCollection&lt;(Of &lt;T&gt;)&gt; has been marked as complete with regards to additions.</td>
</tr>
<tr>
<td>System::::ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;T&gt;)&gt; has been disposed.</td>
</tr>
<tr>
<td>System::::ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out -or- timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System::::InvalidOperationException</td>
<td>The underlying collection didn't accept the item.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (Of T)>) Class
TryAdd Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add the specified item to the BlockingCollection<Of <(T)>). A OperationCanceledException is thrown if the CancellationToken is canceled.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryAdd ( _
    item As T, _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public bool TryAdd(
    T item,
    int millisecondsTimeout,
    CancellationToken cancellationToken
)

Parameters

item
Type: T
The item to be added to the collection.

millisecondsTimeout
Type: System..::.Int32
The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

cancellationToken
Type: System.Threading..::.CancellationToken
A cancellation token to observe.

Return Value

true if the item could be added to the collection within the alloted time; otherwise, false.
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::OperationCanceledException</td>
<td>If the CancellationToken is canceled.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The BlockingCollection&lt;Of (T)&gt; has been marked as complete with regards to additions.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of (T)&gt; has been disposed.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The underlying collection didn't accept the item.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
TryAdd Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
BlockingCollection(Of T).TryAddToAny Method
BlockingCollection(Of T). Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TryAddToAny(array&lt;BlockingCollection&lt;Of&lt;br&gt;(T)&gt;,[][], T)</td>
<td>Attempts to add the specified item to any one of the specified BlockingCollection&lt;Of&lt;br&gt;(T)&gt; instances.</td>
</tr>
<tr>
<td>TryAddToAny(array&lt;BlockingCollection&lt;Of&lt;br&gt;(T)&gt;,[][], T, Int32)</td>
<td>Attempts to add the specified item to any one of the specified BlockingCollection&lt;Of&lt;br&gt;(T)&gt; instances.</td>
</tr>
<tr>
<td>TryAddToAny(array&lt;BlockingCollection&lt;Of&lt;br&gt;(T)&gt;,[][], T, TimeSpan)</td>
<td>Attempts to add the specified item to any one of the specified BlockingCollection&lt;Of&lt;br&gt;(T)&gt; instances.</td>
</tr>
<tr>
<td>TryAddToAny(array&lt;BlockingCollection&lt;Of&lt;br&gt;(T)&gt;,[][], T, Int32, CancellationToken)</td>
<td>Attempts to add the specified item to any one of the specified BlockingCollection&lt;Of&lt;br&gt;(T)&gt; instances. A OperationCanceledException is thrown if the CancellationToken is canceled.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add the specified item to any one of the specified `BlockingCollection<T>` instances.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TryAddToAny ( _
collections As BlockingCollection(Of T)(), _
item As T _
) As Integer

C#

public static int TryAddToAny(
    BlockingCollection<T>[] collections,
    T item
)

Parameters

collections
    Type: array< System.Collections.Concurrent,.;.,BlockingCollection<Of 
    <(T)>)>[]]
    The array of collections.

item
    Type: T
    The item to be added to one of the collections.

Return Value

The index of the collection in the collections array to which the item was added, or -1 if the item could not be added.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::.ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System::.ArgumentNullException</td>
<td>The collections argument is a 0-length array or contains a null element, or at least one of collections has been marked as complete for adding.</td>
</tr>
<tr>
<td>System::.ArgumentException</td>
<td>At least one of the BlockingCollection&lt;Of &lt;(Of&lt;T&gt;)&gt; instances has been disposed.</td>
</tr>
<tr>
<td>System::.InvalidOperationException</td>
<td>At least one underlying collection didn't accept the item.</td>
</tr>
<tr>
<td>System::.ArgumentOutOfRangeException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

[BlockingCollection(Of(Of(T))>)] Class
TryAddToAny Overload
System.Collections.Concurrent Namespace

Send [feedback] on this topic to Microsoft.
Attempts to add the specified item to any one of the specified BlockingCollection<(Of <(T)>)> instances.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TryAddToAny ( _
collections As BlockingCollection(Of T)(), _
item As T, _
millisecondsTimeout As Integer _
) As Integer

C#

public static int TryAddToAny(
    BlockingCollection<T>[] collections,
    T item,
    int millisecondsTimeout
)

Parameters

collections
    Type: array< System.Collections.Concurrent..::.BlockingCollection<Of
<T>>>[]
The array of collections.

item
    Type: T
    The item to be added to one of the collections.

millisecondsTimeout
    Type: System..::..Int32
    The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

Return Value

The index of the collection in the collections array to which the item was added, or -1 if the item could not be added.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is a 0-length array or contains a null element, or at least one of the collections has been marked as complete for adding.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>At least one of the BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; instances has been disposed. millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>At least one of the BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; instances has been disposed. millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>At least one underlying collection didn't accept the item.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
TryAddToAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add the specified item to any one of the specified BlockingCollection<(Of <(T)>)> instances.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function TryAddToAny ( _
collections As BlockingCollection(Of T)(), _
item As T, _
timeout As TimeSpan _
) As Integer
```

#### C#

```csharp
public static int TryAddToAny(
    BlockingCollection<T>[] collections,
    T item,
    TimeSpan timeout
)
```

### Parameters

**collections**
- **Type:** array of System.Collections.Concurrent::BlockingCollection<T>[]
- The array of collections.

**item**
- **Type:** T
- The item to be added to one of the collections.

**timeout**
- **Type:** System::TimeSpan
- A TimeSpan that represents the number of milliseconds to wait, or a TimeSpan that represents -1 milliseconds to wait indefinitely.

### Return Value

The index of the collection in the collections array to which the item was added,
or -1 if the item could not be added.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is null. The collections argument is a 0-length array or contains a null element, or at least one of collections has been marked as complete for adding.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>At least one of the</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>BlockingCollection(Of (Of (T)&gt;) instances has been disposed. timeout is a negative number other than -1 milliseconds, which represents an infinite time-out -or- timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>At least one underlying collection didn't accept the item. The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td></td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
TryAddToAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add the specified item to any one of the specified
BlockingCollection<(Of <(T)>)> instances. A OperationCanceledException is
thrown if the CancellationToken is canceled.

**Namespace:** System.Collections.Concurrent
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TryAddToAny ( _
    collections As BlockingCollection(Of T)(), _
    item As T, _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Integer

C#

public static int TryAddToAny(
    BlockingCollection<T>[] collections,
    T item,
    int millisecondsTimeout,
    CancellationToken cancellationToken
)

Parameters

collections
    Type: array< System.Collections.Concurrent..:::BlockingCollection<(Of
        (Of T))),>[]]
The array of collections.

item
    Type: T
    The item to be added to one of the collections.

millisecondsTimeout
    Type: System..:::.Int32
    The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

cancellationToken
    Type: System.Threading..:::.CancellationToken
    A cancellation token to observe.
**Return Value**

The index of the collection in the collections array to which the item was added, or -1 if the item could not be added.
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::&lt;i&gt;.OperationCanceledException</td>
<td>If the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.cancellationtoken">CancellationToken</a> is canceled.</td>
</tr>
<tr>
<td>System::&lt;i&gt;.ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System::&lt;i&gt;.ArgumentNullException</td>
<td>The collections argument is a 0-length array or contains a null element, or at least one of the collections has been marked as complete for adding.</td>
</tr>
<tr>
<td>System::&lt;i&gt;.ArgumentException</td>
<td>At least one of the BlockingCollection&lt;Of &lt;(T)&gt;&gt;) instances has been disposed.</td>
</tr>
<tr>
<td>System::&lt;i&gt;.ObjectDisposedException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System::&lt;i&gt;.ArgumentOutOfRangeException</td>
<td>At least one underlying collection didn't accept the item.</td>
</tr>
<tr>
<td>System::&lt;i&gt;.InvalidOperationException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
TryAddToAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
BlockingCollection<(Of <(T)>))...TryTake Method
BlockingCollection<(Of <(T)>)) Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TryTake(T%)</code></td>
<td>Attempts to remove an item from the <code>BlockingCollection&lt;(Of &lt;(T)&gt;)&gt;</code>.</td>
</tr>
<tr>
<td><code>TryTake(T%, Int32)</code></td>
<td>Attempts to remove an item from the <code>BlockingCollection&lt;(Of &lt;(T)&gt;)&gt;</code>.</td>
</tr>
<tr>
<td><code>TryTake(T%, TimeSpan)</code></td>
<td>Attempts to remove an item from the <code>BlockingCollection&lt;(Of &lt;(T)&gt;)&gt;</code>.</td>
</tr>
<tr>
<td><code>TryTake(T%, Int32, CancellationToken)</code></td>
<td>Attempts to remove an item from the <code>BlockingCollection&lt;(Of &lt;(T)&gt;)&gt;</code>. A <code>OperationCanceledException</code> is thrown if the <code>CancellationToken</code> is canceled.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove an item from the `BlockingCollection<T>`.

**Namespace:** `System.Collections.Concurrent`  
**Assembly:** `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Public Function TryTake ( _
    <OutAttribute> ByVal item As T _
) As Boolean

C#

public bool TryTake(
    out T item
)

Parameters

item
    Type: T
    The item removed from the collection.

Return Value

true if an item could be removed; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::&lt;ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of &lt;(T)&gt;&gt;) has been disposed.</td>
</tr>
<tr>
<td>System::&lt;InvalidOperationException</td>
<td>The underlying collection was modified outside of this BlockingCollection&lt;Of &lt;(T)&gt;&gt;) instance.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (Of (T)>) Class
TryTake Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove an item from the BlockingCollection<(Of <(T)>).}

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Public Function TryTake ( _  
  <OutAttribute> ByRef item As T, _  
  millisecondsTimeout As Integer _  
) As Boolean

**C#**

public bool TryTake(  
  out T item,  
  int millisecondsTimeout  
)

**Parameters**

*item*  
Type: **T** %  
The item removed from the collection.

*millisecondsTimeout*  
Type: System...:::Int32  
The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

**Return Value**

true if an item could be removed from the collection within the alloted time;  
otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The <code>BlockingCollection&lt;Of &lt;(T)▻&gt;</code> has been disposed.</td>
</tr>
<tr>
<td><code>System:::ArgumentOutOfRangeException</code></td>
<td><code>millisecondsTimeout</code> is a negative number other than -1, which represents an infinite time-out. The underlying collection was modified outside of this <code>BlockingCollection&lt;Of &lt;(T)▻&gt;</code> instance.</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td></td>
</tr>
</tbody>
</table>


See Also

BlockingCollection(Of(Of T)>) Class
TryTake Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove an item from the `BlockingCollection<Of <(T)▻>`.

**Namespace:**  [System.Collections.Concurrent](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryTake ( _
    <OutAttribute> ByRef item As T, _
    timeout As TimeSpan _
) As Boolean

C#

public bool TryTake(
    out T item,
    TimeSpan timeout
)

Parameters

item
    Type: T
    The item removed from the collection.

timeout
    Type: System::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

true if an item could be removed from the collection within the alloted time;
otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; has been disposed.</td>
</tr>
<tr>
<td></td>
<td>timeout is a negative number other than -1 milliseconds, which</td>
</tr>
<tr>
<td></td>
<td>represents an infinite time-out - or - timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The underlying collection was</td>
</tr>
<tr>
<td></td>
<td>modified outside of this</td>
</tr>
<tr>
<td></td>
<td>BlockingCollection&lt;(Of &lt;(T)&gt;)&gt;</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>instance.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
TryTake Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove an item from the `BlockingCollection<Of <(T).Uint32, CancellationToken>`.

A `OperationCanceledException` is thrown if the `CancellationToken` is canceled.

**Namespace:** `System.Collections.Concurrent`
**Assembly:** `System.Threading` (in `System.Threading.dll`)
### Syntax

**Visual Basic (Declaration)**

Public Function TryTake ( _
    <OutAttribute> ByRef item As T, _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Boolean

**C#**

```csharp
public bool TryTake(
    out T item,
    int millisecondsTimeout,
    CancellationToken cancellationToken
)
```

### Parameters

**item**
- Type: `T`%
- The item removed from the collection.

**millisecondsTimeout**
- Type: `System:::Int32`
- The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

**cancellationToken**
- Type: `System.Threading:::CancellationToken`
- A cancellation token to observe.

### Return Value

true if an item could be removed from the collection within the alloted time; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>If the <a href="https://learn.microsoft.com/en-us/dotnet/api/system.cancellationtoken">CancellationToken</a> is canceled.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The <code>BlockingCollection&lt;Of </code>(Of <code>T&gt;</code>)&gt;` has been disposed.</td>
</tr>
<tr>
<td><code>System:::ArgumentOutOfRangeException</code></td>
<td>The underlying collection was modified outside of this <code>BlockingCollection&lt;Of </code>(Of <code>T&gt;</code>)&gt;`) instance.</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
TryTake Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
BlockingCollection<(Of <(T)>)>...TryTakeFromAny Method
BlockingCollection<(Of <(T)>)> Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Overload List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>TryTakeFromAny(array&lt;BlockingCollection&lt;Of T&gt;&gt;)[[]], T%)</td>
</tr>
<tr>
<td>TryTakeFromAny(array&lt;BlockingCollection&lt;Of T&gt;&gt;)[[]], T%, Int32)</td>
</tr>
<tr>
<td>TryTakeFromAny(array&lt;BlockingCollection&lt;Of T&gt;&gt;)[[]], T%, TimeSpan)</td>
</tr>
<tr>
<td>TryTakeFromAny(array&lt;BlockingCollection&lt;Of T&gt;&gt;)[[]], T%, Int32, CancellationToken)</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove an item from any one of the specified BlockingCollection<(Of <(T)>)> instances.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TryTakeFromAny ( _
   collections As BlockingCollection(Of T)(), _
   <OutAttribute> ByRef item As T _
) As Integer

C#

public static int TryTakeFromAny(
   BlockingCollection<T>[] collections,
   out T item
)

Parameters

collections
   Type: array< System.Collections.Concurrent..::.BlockingCollection<Of
   <(T)>>)[]
   The array of collections.

item
   Type: T %
   The item removed from one of the collections.

Return Value

The index of the collection in the collections array from which the item was removed, or -1 if an item could not be removed.
Remarks

A call to TryTakeFromAny may block until an item is available to be removed.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The collections argument is a 0-length array or contains a null element.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>At least one of the BlockingCollection&lt;Of &lt;(T)&gt;&gt;) instances has been disposed.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>At least one of the underlying collections was modified outside of its BlockingCollection&lt;Of &lt;(T)&gt;&gt;) instance.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

**BlockingCollection(Of (Of T)> Class**
*TryTakeFromAny Overload*
**System.Collections.Concurrent Namespace**

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Attempts to remove an item from any one of the specified BlockingCollection<Of <(T)>) instances.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TryTakeFromAny ( _
collections As BlockingCollection(Of T)(), _
<OutAttribute> ByRef item As T, _
millisecondsTimeout As Integer _
) As Integer

C#

public static int TryTakeFromAny(
    BlockingCollection<T>[] collections,
    out T item,
    int millisecondsTimeout
)

Parameters

collections
    Type: array< System.Collections.Concurrent..::.BlockingCollection<Of
    <(T)>)>[]()[]
    The array of collections.

item
    Type: T %
    The item removed from one of the collections.

millisecondsTimeout
    Type: System....:..Int32
    The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

Return Value

The index of the collection in the collections array from which the item was
removed, or -1 if an item could not be removed.
Remarks

A call to TryTakeFromAny may block until an item is available to be removed.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is a 0-length array or contains a null element.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>At least one of the BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; instances has been disposed.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>At least one of the underlying collections was modified outside of its BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; instance.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
TryTakeFromAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove an item from any one of the specified BlockingCollection<(Of (<T>))> instances.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TryTakeFromAny ( _
    collections As BlockinCollection(Of T)(), _
    <OutAttribute> ByRef item As T, _
    timeout As TimeSpan _
) As Integer

C#

public static int TryTakeFromAny(
    BlockingCollection<T>[] collections,
    out T item,
    TimeSpan timeout
)

Parameters

collections
    Type: array< System.Collections.Concurrent..::..BlockingCollection<T>[]>
    The array of collections.

timeout
    Type: System..::..TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

The index of the collection in the collections array from which the item was
removed, or -1 if an item could not be removed.
Remarks

A call to TryTakeFromAny may block until an item is available to be removed.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The collections argument is a 0-length array or contains a null element.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>At least one of the BlockingCollection&lt;Of &lt;(T)&gt;&gt; instances has been disposed.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out -or- timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>At least one of the underlying collections was modified outside of its BlockingCollection&lt;Of &lt;(T)&gt;&gt; instance.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection<(Of (T)>)> Class
TryTakeFromAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove an item from any one of the specified BlockingCollection<Of <(T)>>) instances. A OperationCanceledException is thrown if the CancellationToken is canceled.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function TryTakeFromAny ( _
    collections As BlockingCollection(Of T)(), _
    <OutAttribute> ByRef item As T, _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Integer
```

#### C#

```csharp
public static int TryTakeFromAny(  
    BlockingCollection<T>[] collections,  
    out T item,  
    int millisecondsTimeout,  
    CancellationToken cancellationToken  
)
```

#### Parameters

- **collections**
  - Type: `array<System.Collections.Concurrent.BlockingCollection<T>[]>`
  - The array of collections.

- **item**
  - Type: `T`
  - The item removed from one of the collections.

- **millisecondsTimeout**
  - Type: `System.Int32`
  - The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

- **cancellationToken**
  - Type: `System.Threading.CancellationToken`
  - A cancellation token to observe.
**Return Value**

The index of the collection in the collections array from which the item was removed, or -1 if an item could not be removed.
Remarks

A call to TryTakeFromAny may block until an item is available to be removed.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::OperationContract</td>
<td>If the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken">CancellationToken</a> is canceled.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The collections argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The collections argument is a 0-length array or contains a null element.</td>
</tr>
<tr>
<td></td>
<td>At least one of the BlockingCollection&lt;T&gt; instances has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The count of collections is greater than the maximum size of 62 for STA and 63 for MTA.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>At least one of the underlying collections was modified outside of its BlockingCollection&lt;T&gt; instance.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
TryTakeFromAny Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `BlockingCollection<Of <T>>` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoundedCapacity</td>
<td>Gets the bounded capacity of this BlockingCollection(&lt;\text{T})) instance.</td>
</tr>
<tr>
<td>Count</td>
<td>Gets the number of items contained in the BlockingCollection(&lt;\text{T})).</td>
</tr>
<tr>
<td>IsAddingCompleted</td>
<td>Gets whether this BlockingCollection(&lt;\text{T})) has been marked as complete for adding.</td>
</tr>
<tr>
<td>IsCompleted</td>
<td>Gets whether this BlockingCollection(&lt;\text{T})) has been marked as complete for adding and is empty.</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ICollection:::IsSynchronized</code></td>
<td>Gets a value indicating whether access to the <code>ICollection</code> is synchronized. Gets an object that can be used to synchronize access to the <code>ICollection</code>. This property is not supported.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value indicating whether access to the ICollection is synchronized.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Private Readonly Property IsSynchronized As Boolean
Implements ICollection.IsSynchronized

**C#**

bool ICollection.IsSynchronized { get; }

**Implements**

ICollection..::.IsSynchronized
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;(T)&gt;)&gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets an object that can be used to synchronize access to the ICollection. This property is not supported.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Readonly Property SyncRoot As Object
Implements ICollection.SyncRoot

C#

Object ICollection.SyncRoot { get; }

Implements

IICollection...::SyncRoot
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.NotSupportedException</td>
<td>The SyncRoot property is not supported.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets the bounded capacity of this `BlockingCollection(Of T)` instance.

**Namespace:** `System.Collections.Concurrent`  
**Assembly:** `System.Threading` (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property BoundedCapacity As Integer

C#

public int BoundedCapacity { get; }

Field Value

The bounded capacity of this collection, or int.MaxValue if no bound was supplied.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of &lt;(T)&gt;&gt;) has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets the number of items contained in the `BlockingCollection(Of T)`. 

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Count As Integer

C#

public int Count { get; }

Field Value

The number of items contained in the BlockingCollection(Of T).

Implements

IEnumerable.::Count
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The BlockingCollection&lt;T&gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets whether this BlockingCollection<(Of '(T)>) has been marked as complete for adding.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsAddingCompleted As Boolean

C#

public bool IsAddingCompleted { get; }

Field Value

Whether this collection has been marked as complete for adding.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The BlockingCollection&lt;Of&lt;(T)&gt;) has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection<(Of (T)> ) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets whether this `BlockingCollection<>` has been marked as complete for adding and is empty.

**Namespace:**  [System.Collections.Concurrent](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsCompleted As Boolean

C#

public bool IsCompleted { get; }

Field Value

Whether this collection has been marked as complete for adding and is empty.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The BlockingCollection&lt;(Of &lt;(T)&gt;)) has been disposed.</td>
</tr>
</tbody>
</table>
See Also

BlockingCollection(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Represents a thread-safe, unordered collection of objects.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute> _
<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
  ExternalThreading := True)> _
Public Class ConcurrentBag(Of T) _
  Implements IProducerConsumerCollection(Of T), IEnumerable(Of
  ICollection, IEnumerable

C#

[SerializableAttribute]
[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
  ExternalThreading = true)]
public class ConcurrentBag<T> : IProducerConsumerCollection<T>,
  IEnumerable<T>, ICollection, IEnumerable
Type Parameters

T

Specifies the type of elements in the bag.
Remarks

Bags are useful for storing objects when ordering doesn't matter, and unlike sets, bags support duplicates. ConcurrentBag<Of <(T)>) is a thread-safe bag implementation, optimized for scenarios where the same thread will be both producing and consuming data stored in the bag.

ConcurrentBag<Of <(T)>) accepts null reference (Nothing in Visual Basic) as a valid value for reference types.

All public and protected members of ConcurrentBag<Of <(T)>) are thread-safe and may be used concurrently from multiple threads.
Inheritance Hierarchy

System..::.Object
System.Collections.Concurrent..::.ConcurrentBag<Of <(T)>>


See Also

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| ConcurrentBag<
  <T>><(Of
  <T>))>()() | Initializes a new instance of the ConcurrentBag<
  <T>><(Of
  <T>)) class. |
| ConcurrentBag<
  <T>><(Of
  <T>))((IEnumerable<
  <T>)) | Initializes a new instance of the ConcurrentBag<
  <T>><(Of
  <T>)) class that contains elements copied from the specified collection. |
See Also

 ConcurrentBag(Of (Of (T) ) ) Class
 System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentBag(Of (T)*)` class.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public ConcurrentBag()
See Also

ConcurrentBag(Of <T>) Class
ConcurrentBag(Of <T>) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentBag(Of <T>`) class that contains elements copied from the specified collection.

**Namespace:** `System.Collections.Concurrent`  
**Assembly:** `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    collection As IEnumerable(Of T) _
)

C#

public ConcurrentBag(
    IEnumerable<T> collection
)

Parameters

collection
    Type: System.Collections.Generic.IEnumerable(Of T)
    The collection whose elements are copied to the new ConcurrentBag(Of T).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>collection is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentBag(Of (T)> ) Class
ConcurrentBag(Of (T)> ) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `ConcurrentBag(Of (T)>)` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add</strong></td>
<td>Adds an object to the ConcurrentBag(Of &lt;(T)&gt;).</td>
</tr>
<tr>
<td><strong>CopyTo</strong></td>
<td>Copies the ConcurrentBag(Of &lt;(T)&gt;). Copies the ConcurrentBag(Of &lt;(T)&gt;).</td>
</tr>
<tr>
<td><strong>Equals</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>Finalize</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetEnumerator</strong></td>
<td>Returns an enumerator that iterates through the ConcurrentBag(Of &lt;(T)&gt;).</td>
</tr>
<tr>
<td><strong>GetHashCode</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetType</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>MemberwiseClone</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>ToArray</strong></td>
<td>Copies the ConcurrentBag(Of &lt;(T)&gt;). Copies the ConcurrentBag(Of &lt;(T)&gt;).</td>
</tr>
<tr>
<td><strong>ToString</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>TryPeek</strong></td>
<td>Attempts to return an object from the ConcurrentBag(Of &lt;(T)&gt;). without removing it.</td>
</tr>
<tr>
<td><strong>TryTake</strong></td>
<td>Attempts to remove and return an object from the ConcurrentBag(Of &lt;(T)&gt;).</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICollection::&lt;CopyTo&gt;</td>
<td>Copies the elements of the ICollection to an Array, starting at a particular Array index.</td>
</tr>
<tr>
<td>IEnumerable::&lt;GetEnumerator&gt;</td>
<td>Returns an enumerator that iterates through the ConcurrentBag::&lt;(Of &lt;(T)&gt;&gt;).</td>
</tr>
<tr>
<td>IProducerConsumerCollection::&lt;TryAdd&gt;</td>
<td>Attempts to add an object to the ConcurrentBag::&lt;(Of &lt;(T)&gt;)).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentBag(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Copies the elements of the ICollection to an Array, starting at a particular Array index.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Sub CopyTo (_
    array As Array, _
    index As Integer _
) Implements ICollection.CopyTo

C#

void ICollection.CopyTo(
    Array array,
    int index
)

Parameters

array
    Type: System...::Array
    The one-dimensional Array that is the destination of the elements copied from the ConcurrentBag<T>. The Array must have zero-based indexing.

index
    Type: System...::Int32
    The zero-based index in array at which copying begins.

Implements

ICollection...::CopyTo(Array, Int32)
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>array is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System..::.ArgumentOutOfRangeException</code></td>
<td>index is less than zero.</td>
</tr>
<tr>
<td><code>System..::.ArgumentException</code></td>
<td>array is multidimensional. -or- array does not have zero-based indexing. -or- index is equal to or greater than the length of the array -or- The number of elements in the source ICollection is greater than the available space from index to the end of the destination array. -or- The type of the source ICollection cannot be cast automatically to the type of the destination array.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentBag(Of (T)> Class  
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Returns an enumerator that iterates through the `ConcurrentBag<(Of <(T)>).::IEnumerable.::.GetEnumerator` method.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function GetEnumerator As IEnumerator
    Implements IEnumerable.GetEnumerator

C#

IEnumerator IEnumerable.GetEnumerator()

Return Value

An enumerator for the contents of the `ConcurrentBag(Of <T>)`.

Implements

IEnumerable...::GetEnumerator()()
Remarks

The items enumerated represent a moment-in-time snapshot of the contents of the bag. It does not reflect any update to the collection after GetEnumerator() was called.
See Also

ConcurrentBag(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Adds an object to the `ConcurrentBag<Of <(T)>>`.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Add ( _
    item As T _
)

C#

public void Add(
    T item
)

Parameters

item
    Type: T
    The object to be added to the ConcurrentBag(Of (Of T)). The value can be a null reference (Nothing in Visual Basic) for reference types.
See Also

ConcurrentBag(Of (T)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
C# .NET Framework Class Library

ConcurrentBag(Of (T)>)..::.CopyTo Method

ConcurrentBag(Of (T)>) Class  See Also  Send Feedback

Covers the ConcurrentBag(Of (T)>) elements to an existing one-dimensional Array, starting at the specified array index.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Sub CopyTo ( _
    array As T(), _
    index As Integer _
)
```

**C#**

```csharp
public void CopyTo(  
    T[] array,  
    int index  
)
```

**Parameters**

array

Type: `array<T>[]()`

The one-dimensional Array that is the destination of the elements copied from the `ConcurrentBag(Of <T>).` The Array must have zero-based indexing.

index

Type: `System..::.Int32`

The zero-based index in array at which copying begins.

**Implements**

`IProducerConsumerCollection(Of <T>),..::.CopyTo(array<T>[][], Int32)`
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>array is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>index is less than zero.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>index is equal to or greater than the length of the array -or- the number of elements in the source <code>ConcurrentBag&lt;Of &lt;(T)&gt;&gt;</code> is greater than the available space from index to the end of the destination array.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentBag(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Returns an enumerator that iterates through the `ConcurrentBag<Of <(T)>`).

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Function GetEnumerator As IEnumerator(Of T)

C#

public IEnumerator<T> GetEnumerator()

Return Value

An enumerator for the contents of the ConcurrentBag(Of (Of T)).
Remarks

The enumeration represents a moment-in-time snapshot of the contents of the bag. It does not reflect any updates to the collection after GetEnumerator() was called. The enumerator is safe to use concurrently with reads from and writes to the bag.
See Also

ConcurrentBag<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add an object to the **ConcurrentBag<(Of <(T)>)>**.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

Visual Basic (Declaration)

```vbnet
Private Function TryAdd ( _
    item As T _
) As Boolean Implements IProducerConsumerCollection(Of T).TryAdd
```

C#

```csharp
bool IProducerConsumerCollection<T>.TryAdd(
    T item
)
```

**Parameters**

item

Type: `T`

The object to be added to the `ConcurrentBag(Of (T))`. The value can be a null reference (Nothing in Visual Basic) for reference types.

**Return Value**

Always returns true

**Implements**

`IProducerConsumerCollection(Of (T)).TryAdd(T)`
See Also

ConcurrentBag(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentBag(Of <(T)>)::.ToArray Method

ConcurrentBag(Of <(T)>) Class  See Also  Send Feedback

Copies the ConcurrentBag(Of <(T)>) elements to a new array.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ToArray As T()

C#

public T[] ToArray()

Return Value

A new array containing a snapshot of elements copied from the ConcurrentBag(Of(Of(T)>)).
See Also

ConcurrentBag<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentBag(Of <T>):::TryPeek Method

Attempts to return an object from the ConcurrentBag(Of <T>) without removing it.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function TryPeek ( _
    <OutAttribute> ByRef result As T _
) As Boolean
```

**C#**

```csharp
public bool TryPeek(
    out T result
)
```

### Parameters

**result**

Type: `T`%

When this method returns, result contains an object from the `ConcurrentBag<Of <(T)>>` or the default value of `T` if the operation failed.

### Return Value

true if and object was returned successfully; otherwise, false.
See Also

ConcurrentBag(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove and return an object from the `ConcurrentBag(Of `(Of T)`>`).

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function TryTake ( _
    <OutAttribute> ByRef result As T _
) As Boolean
```

**C#**

```csharp
public bool TryTake(
    out T result
)
```

**Parameters**

result

Type: `T`%

When this method returns, result contains the object removed from the `ConcurrentBag(Of `(T>`)> or the default value of T if the operation failed.

**Return Value**

true if an object was removed successfully; otherwise, false.

**Implements**

`IProducerConsumerCollection(Of `(T>`)>..::.TryTake(T%)`
See Also

ConcurrentBag<(Of <(T)>)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The ConcurrentBag<(Of <(T)>)> type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Gets the number of elements contained in the ConcurrentBag&lt;Of&lt;T&gt;&gt;.</td>
</tr>
<tr>
<td>isEmpty</td>
<td>Gets a value that indicates whether the ConcurrentBag&lt;Of&lt;T&gt;&gt; is empty.</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ICollection::IsSynchronized</code></td>
<td>Gets a value indicating whether access to the <code>ICollection</code> is synchronized with the <code>SyncRoot</code>.</td>
</tr>
<tr>
<td><code>ICollection::SyncRoot</code></td>
<td>Gets an object that can be used to synchronize access to the <code>ICollection</code>. This property is not supported.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentBag(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentBag(Of (Of T)>).::.ICollection.::.IsSynchronized Property

ConcurrentBag(Of (Of T)>) Class  See Also  Send Feedback

Gets a value indicating whether access to the ICollection is synchronized with the SyncRoot.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
# Syntax

**Visual Basic (Declaration)**

Private ReadOnly Property IsSynchronized As Boolean
   Implements ICollection.IsSynchronized

**C#**

bool ICollection.IsSynchronized { get; }

**Field Value**

true if access to the ICollection is synchronized with the SyncRoot; otherwise, false. For `ConcurrentBag<Of <T>>`, this property always returns false.

**Implements**

ICollection...:::IsSynchronized
See Also

ConcurrentBag(Of (T)> ) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets an object that can be used to synchronize access to the ICollection. This property is not supported.

**Namespace:**  System.Collections.Concurrent

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Readonly Property SyncRoot As Object
    Implements ICollection.SyncRoot

C#

Object ICollection.SyncRoot { get; }

Implements

ICollection.ICollection.SyncRoot
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.NotSupportedException</td>
<td>The SyncRoot property is not supported.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentBag<(Of <(T)> )> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets the number of elements contained in the `ConcurrentBag<Of <T>>`. 

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
### Syntax

**Visual Basic (Declaration)**

Public ReadOnly Property Count As Integer

**C#**

```csharp
public int Count { get; }
```

### Field Value

The number of elements contained in the `ConcurrentBag<T>`.

### Implements

ICollection...Count
Remarks

The count returned represents a moment-in-time snapshot of the contents of the bag. It does not reflect any updates to the collection after `GetEnumerator()` was called.
See Also

ConcurrentBag(Of (T)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value that indicates whether the ConcurrentBag<T> is empty.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property IsEmpty As Boolean

C#

public bool IsEmpty { get; }

Field Value

true if the ConcurrentBag(Of (Of T)) is empty; otherwise, false.
See Also

ConcurrentBag(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Represents a thread-safe collection of keys and values.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute> _
<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
  ExternalThreading := True)> _
Public Class ConcurrentDictionary(Of TKey, TValue) _
  Implements IDictionary(Of TKey, TValue), ICollection(Of KeyValuePair(Of TKey, TValue)), IDictionary, ICollection, IEnumerable

C#

[SerializableAttribute]
[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
  ExternalThreading = true)]
public class ConcurrentDictionary<TKey, TValue> : IDictionary<TKey, TValue>, ICollection<KeyValuePair<TKey, TValue>>, IEnumerable<KeyValuePair<TKey, TValue>>, IDictionary, ICollection, IEnumerable
**Type Parameters**

**TKey**
The type of the keys in the dictionary.

**TValue**
The type of the values in the dictionary.
Remarks

All public and protected members of ConcurrentDictionary<\(\text{Of } \langle\text{TKey, TValue}\rangle\rangle\) are thread-safe and may be used concurrently from multiple threads.
Inheritance Hierarchy

System...:::Object
   System.Collections.Concurrent...:::ConcurrentDictionary<(Of <(TKey, TValue)>)>
See Also

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
ConcurrentDictionary(Of (Of TKey, TValue)>)) Constructor
ConcurrentDictionary(Of Of TKey, TValue)>)> Class  See Also  Send Feedback
# Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConcurrentDictionary&lt;OF (TKey, TValue)&gt;()()</code></td>
<td>Initializes a new instance of the <code>ConcurrentDictionary&lt;OF (TKey, TValue)&gt;</code> class that is empty, has the default concurrency level, has the default initial capacity, and uses the default comparer for the key type.</td>
</tr>
<tr>
<td><code>ConcurrentDictionary&lt;OF (TKey, TValue&gt;)(IEnumerable&lt;OF (KeyValuePair&lt;OF (TKey, TValue&gt;)&gt;)&gt;)</code></td>
<td>Initializes a new instance of the <code>ConcurrentDictionary&lt;OF (TKey, TValue)&gt;</code> class that contains elements copied from the specified <code>IEnumerable&lt;OF (KeyValuePair&lt;OF (TKey, TValue)&gt;)&gt;</code>), has the default concurrency level, has the default initial capacity, and uses the default comparer for the key type.</td>
</tr>
<tr>
<td><code>ConcurrentDictionary&lt;OF (TKey, TValue)&gt;)(IEqualityComparer&lt;OF (TKey)&gt;)</code></td>
<td>Initializes a new instance of the <code>ConcurrentDictionary&lt;OF (TKey, TValue)&gt;</code> class that is empty, has the specified concurrency level and capacity, and uses the specified <code>IEqualityComparer&lt;OF (TKey)&gt;</code>).</td>
</tr>
<tr>
<td><code>ConcurrentDictionary&lt;OF (TKey, TValue&gt;)(IEnumerable&lt;OF (KeyValuePair&lt;OF (TKey, TValue&gt;)&gt;&gt;, IEqualityComparer&lt;OF (TKey)&gt;)</code></td>
<td>Initializes a new instance of the <code>ConcurrentDictionary&lt;OF (TKey, TValue)&gt;</code> class that contains elements copied from the specified `IEnumerable&lt;OF (KeyValuePair&lt;OF (TKey, TValue&gt;)&gt;), and uses the default concurrency level, has the default initial capacity, and uses</td>
</tr>
</tbody>
</table>
The specified IEqualityComparer<(Of <(TKey)>)>.

Initializes a new instance of the ConcurrentDictionary<(Of <(TKey, TValue)>)> class that is empty, has the specified concurrency level and capacity, and uses the default comparer for the key type.

Initializes a new instance of the ConcurrentDictionary<(Of <(TKey, TValue)>)> class that contains elements copied from the specified IEnumerable, has the specified concurrency level, has the specified initial capacity, and uses the specified IEqualityComparer<(Of <(TKey)>)>.

Initializes a new instance of the ConcurrentDictionary<(Of <(TKey, TValue)>)> class that is empty, has the specified concurrency level, has the specified initial capacity, and uses the specified IEqualityComparer<(Of <(TKey)>)>.
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentDictionary<(Of <(TKey, TValue)>)>` class that is empty, has the default concurrency level, has the default initial capacity, and uses the default comparer for the key type.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public ConcurrentDictionary()
See Also

ConcurrentDictionary(Of (TKey, TValue)>) Class
ConcurrentDictionary(Of (TKey, TValue)>) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of (TKey, TValue)>) Constructor (IEnumerable<(Of (KeyValuePair<(Of (TKey, TValue)>), TValue)>), IEnumerable<(Of (KeyValuePair<(Of (TKey, TValue)>), TValue)>), IEnumerable<(Of (KeyValuePair<(Of (TKey, TValue)>), TValue)>), IEnumerable<(Of (KeyValuePair<(Of (TKey, TValue)>), TValue)>))

Initializes a new instance of the ConcurrentDictionary<(Of (TKey, TValue)>) class that contains elements copied from the specified IEnumerable<(Of (KeyValuePair<(Of (TKey, TValue)>), TValue)>), IEnumerable<(Of (KeyValuePair<(Of (TKey, TValue)>), TValue)>), IEnumerable<(Of (KeyValuePair<(Of (TKey, TValue)>), TValue)>), IEnumerable<(Of (KeyValuePair<(Of (TKey, TValue)>), TValue)>)) instance, has the default concurrency level, has the default initial capacity, and uses the default comparer for the key type.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    collection As IEnumerable(Of KeyValuePair(Of TKey, TValue))
)

C#

public ConcurrentDictionary(
    IEnumerable<KeyValuePair<TKey, TValue>> collection
)

Parameters

collection
    Type: System.Collections.Generic.IEnumerable(Of KeyValuePair(Of TKey, TValue))
The IEnumerable(Of KeyValuePair(Of TKey, TValue)) whose elements are copied to the new ConcurrentDictionary(Of TKey, TValue).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>collection is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>collection contains one or more duplicate keys.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
ConcurrentDictionary(Of TKey, TValue) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)> Constructor

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class  See Also  Send Feedback

Initializes a new instance of the ConcurrentDictionary<(Of <(TKey, TValue)>)> class that is empty, has the specified concurrency level and capacity, and uses the specified IEqualityComparer<(Of <(TKey)>)>.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Sub New ( _
    comparer As IEqualityComparer(Of TKey) _
)
```

**C#**

```csharp
public ConcurrentDictionary(
    IEqualityComparer<TKey> comparer
)
```

**Parameters**

comparer

Type: System.Collections.Generic.IEqualityComparer(Of TKey)

The IEqualityComparer(Of TKey) implementation to use when comparing keys.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>comparer is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
ConcurrentDictionary(Of TKey, TValue) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)> Constructor (IEnumerable<(Of <(KeyValuePair<(Of <(TKey, TValue)>))>)>), IEqualityComparer<(Of <(TKey)>)>)

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class  See Also  Send Feedback

Initializes a new instance of the ConcurrentDictionary<(Of <(TKey, TValue)>)> class that contains elements copied from the specified IEnumerable, has the default concurrency level, has the default initial capacity, and uses the specified IEqualityComparer<(Of <(TKey)>)>.

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Sub New ( _
    collection As IEnumerable(Of KeyValuePair(Of TKey, TValue)),
    comparer As IEqualityComparer(Of TKey) _
)
```

**C#**

```csharp
public ConcurrentDictionary(  
    IEnumerable<KeyValuePair<TKey, TValue>> collection,
    IEqualityComparer<TKey> comparer
)
```

### Parameters

**collection**
- Type: System.Collections.Generic.IEnumerable<(Of (Of KeyValuePair<(Of TKey, TValue>>)>)
- The `IEnumerable<(Of (Of KeyValuePair<(Of TKey, TValue>>)>)` whose elements are copied to the new `ConcurrentDictionary<(Of (TKey, TValue))>`. 

**comparer**
- Type: System.Collections.Generic.IEqualityComparer<(Of TKey)>)
- The `IEqualityComparer<(Of TKey)>)` implementation to use when comparing keys.
## Exceptions

<table>
<thead>
<tr>
<th><strong>Exception</strong></th>
<th><strong>Condition</strong></th>
</tr>
</thead>
</table>
See Also

ConcurrentDictionary<(Of (TKey, TValue)>>) Class
ConcurrentDictionary<(Of (TKey, TValue)>>) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentDictionary(Of (TKey, TValue)>)` class that is empty, has the specified concurrency level and capacity, and uses the default comparer for the key type.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    concurrencyLevel As Integer, _
    capacity As Integer _
)

C#

public ConcurrentDictionary(
    int concurrencyLevel,
    int capacity
)
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>concurrencyLevel is less than 1.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>capacity is less than 0.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
ConcurrentDictionary(Of TKey, TValue) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentDictionary(Of (TKey, TValue)>)` class that contains elements copied from the specified `IEnumerable`, has the specified concurrency level, has the specified initial capacity, and uses the specified `IEqualityComparer(Of (TKey)>)`.

**Namespace:** `System.Collections.Concurrent`  
**Assembly:** `System.Threading` (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New (_
    concurrencyLevel As Integer, _
    collection As IEnumerable(Of KeyValuePair(Of TKey, TValue)),
    comparer As IEqualityComparer(Of TKey) _
)

C#

public ConcurrentDictionary(
    int concurrencyLevel,
    IEnumerable<KeyValuePair<TKey, TValue>> collection,
    IEqualityComparer<TKey> comparer
)

Parameters

concurrencyLevel
    Type: System..::.Int32
    The estimated number of threads that will update the
    ConcurrentDictionary(Of <(TKey, TValue)>)
    concurrently.

collection
    Type: System.Collections.Generic..::.IEnumerable(Of
        <(KeyValuePair(Of <(TKey, TValue)>)>)
    The IEnumerable(Of <(KeyValuePair(Of <(TKey, TValue)>)>)
    whose elements are copied to the new ConcurrentDictionary(Of <(TKey, TValue)>).

comparer
    Type: System.Collections.Generic..::.IEqualityComparer(Of <(TKey)>)
    The IEqualityComparer(Of <(TKey)>) implementation to use when
    comparing keys.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>collection is a null reference (Nothing in Visual Basic). -or- comparer is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...:::ArgumentOutOfRangeException</td>
<td>concurrencyLevel is less than 1.</td>
</tr>
<tr>
<td>System...:::ArgumentException</td>
<td>collection contains one or more duplicate keys.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue>) Class
ConcurrentDictionary(Of TKey, TValue>) Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentDictionary<Of <(TKey, TValue)>>` class that is empty, has the specified concurrency level, has the specified initial capacity, and uses the specified `IEqualityComparer<Of <(TKey)>>`.

**Namespace:**  System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    concurrencyLevel As Integer, _
    capacity As Integer, _
    comparer As IEqualityComparer(Of TKey) _
)

C#

public ConcurrentDictionary(
    int concurrencyLevel,
    int capacity,
    IEqualityComparer<TKey> comparer
)

Parameters

concurrencyLevel
Type: System..::.Int32
The estimated number of threads that will update the
ConcurrentDictionary(Of <(TKey, TValue)>)
concurrently.

capacity
Type: System..::.Int32
The initial number of elements that the ConcurrentDictionary(Of
<(TKey, TValue)>)
can contain.

comparer
Type: System.Collections.Generic..::.IEqualityComparer(Of
<(TKey)>)
The IEqualityComparer(Of <(TKey)>)
implementation to use when
comparing keys.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>concurrencyLevel is less than 1. - or- capacity is less than 0.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>comparer is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)>)

ConcurrentDictionary(Of (TKey, TValue)>)

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `ConcurrentDictionary(Of TKey, TValue)` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddOrUpdate</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Clear</td>
<td>Removes all keys and values from the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt; &gt;</code>.</td>
</tr>
<tr>
<td>ContainsKey</td>
<td>Determines whether the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt; &gt;</code> contains the specified key.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetEnumerator</td>
<td>Returns an enumerator that iterates through the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt; &gt;</code>.</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToArray</td>
<td>Copies the key and value pairs stored in the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt; &gt;</code> to a new array.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>TryAdd</td>
<td>Attempts to add the specified key and value to the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt; &gt;</code>.</td>
</tr>
<tr>
<td>TryGetValue</td>
<td>Attempts to get the value associated with the specified key from the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt; &gt;</code>.</td>
</tr>
<tr>
<td>TryRemove</td>
<td>Attempts to remove and return the the value with the specified key from the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt; &gt;</code>.</td>
</tr>
<tr>
<td>TryUpdate</td>
<td>Compares the existing value for the specified key with a specified value, and if they’re equal, updates the key with a third value.</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ICollection&lt;(Of KeyValuePair&lt;(Of (TKey, TValue)&gt;)&gt;).::Add</code></td>
<td>Adds the specified key and value to the <code>IDictionary&lt;(Of (TKey, TValue)&gt;)&gt;</code>.</td>
</tr>
<tr>
<td><code>IDictionary&lt;(Of (TKey, TValue)&gt;)&gt;.::Add</code></td>
<td>Adds the specified key and value to the dictionary.</td>
</tr>
<tr>
<td><code>IDictionary.::Add</code></td>
<td></td>
</tr>
<tr>
<td><code>ICollection&lt;(Of KeyValuePair&lt;(Of (TKey, TValue)&gt;)&gt;)&gt;).::Contains</code></td>
<td>Gets whether the <code>IDictionary&lt;(Of (TKey, TValue)&gt;)&gt;</code> contains an element with the specified key.</td>
</tr>
<tr>
<td><code>IDictionary.::Contains</code></td>
<td></td>
</tr>
<tr>
<td><code>ICollection&lt;(Of KeyValuePair&lt;(Of (TKey, TValue)&gt;)&gt;)&gt;).::CopyTo</code></td>
<td>Copies the elements of the <code>ICollection</code> to an array, starting at the specified array index.</td>
</tr>
<tr>
<td><code>ICollection.::CopyTo</code></td>
<td>Provides an <code>IDictionaryEnumerator</code> for the <code>IDictionary&lt;(Of (TKey, TValue)&gt;)&gt;</code>.</td>
</tr>
<tr>
<td><code>IDictionary.::GetEnumerator</code></td>
<td>Returns an enumerator that iterates through the <code>ConcurrentDictionary&lt;(Of (TKey, TValue)&gt;)&gt;</code>.</td>
</tr>
<tr>
<td><code>IEnumerator.::GetEnumerator</code></td>
<td></td>
</tr>
<tr>
<td><code>ICollection&lt;(Of KeyValuePair&lt;(Of (TKey, TValue)&gt;)&gt;)&gt;).::Remove</code></td>
<td>Removes the element with the specified key from the <code>IDictionary&lt;(Of (TKey, TValue)&gt;)&gt;</code>.</td>
</tr>
<tr>
<td><code>IDictionary&lt;(Of (TKey, TValue)&gt;)&gt;).::Remove</code></td>
<td></td>
</tr>
<tr>
<td>IDictionary:::Remove</td>
<td>Removes the element with the specified key from the IDictionary.</td>
</tr>
</tbody>
</table>
See Also

`ConcurrentDictionary<Of <(TKey, TValue)>>()` Class
`System.Collections.Concurrent` Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>::ICollection<(Of <(KeyValuePair<(Of <(TKey, TValue)>)>>)>::Add Method

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class  See Also  Send Feedback

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Sub Add ( _
    keyValuePair As KeyValuePair(Of TKey, TValue) _
) Implements ICollection(Of KeyValuePair(Of TKey, TValue)).Add

C#

void ICollection<KeyValuePair<TKey, TValue>>.Add(
    KeyValuePair<TKey, TValue> keyValuePair
)

Parameters

keyValuePair
    Type: System.Collections.Generic..::.KeyValuePair(Of TKey, TValue>)

Implements

ICollection(Of (Of (T)>)...Add(T)
See Also

ConcurrentDictionary(Of (TKey, TValue)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>..::.IDictionary<(Of <(TKey, TValue)>)>..::.Add Method

**ConcurrentDictionary<(Of <(TKey, TValue)>)>** Class  See Also  Send Feedback

Adds the specified key and value to the IDictionary<(Of <(TKey, TValue)>)>.

**Namespace:**  System.Collections.Concurrent

**Assembly:**  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Private Sub Add ( _
    key As TKey, _
    value As TValue  _
) Implements IDictionary(Of TKey, TValue).Add
```

### C#

```csharp
void IDictionary<TKey, TValue>.Add(
    TKey key,
    TValue value
)
```

## Parameters

**key**

Type: **TKey**

The object to use as the key of the element to add.

**value**

Type: **TValue**

The object to use as the value of the element to add.

## Implements

IDictionary<(Of <(TKey, TValue)>)>...:..:Add(TKey, TValue)
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::OverflowException</td>
<td>The dictionary contains too many elements. An element with the same key already exists in the ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt;&gt;.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>..::.IDictionary..::.Add Method

`ConcurrentDictionary<(Of <(TKey, TValue)>)> Class`  See Also  Send Feedback

Adds the specified key and value to the dictionary.

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading` (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Sub Add (_
    key As Object, _
    value As Object _
) Implements IDictionary.Add

C#

do void IDictionary.Add(
    Object key,
    Object value
)

Parameters

key
   Type: System:::Object
   The object to use as the key.

value
   Type: System:::Object
   The object to use as the value.

Implements

IDictionary:::Add(Object, Object)
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::OverflowException</td>
<td>The dictionary contains too many elements.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>key is of a type that is not assignable to the key type TKey of the Dictionary&lt;(Of&lt;TKey, TValue&gt;). -or- value is of a type that is not assignable to TValue, the type of values in the Dictionary&lt;(Of&lt;TKey, TValue&gt;). -or- A value with the same key already exists in the Dictionary&lt;(Of&lt;TKey, TValue&gt;).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>::ICollection<(Of <(KeyValuePair<(Of <(TKey, TValue)>)>)>)::Contains Method

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function Contains ( _
    keyValuePair As KeyValuePair(Of TKey, TValue) _
) As Boolean Implements ICollection(Of KeyValuePair(Of TKey, TValue))

C#

bool ICollection<KeyValuePair<TKey, TValue>>.Contains(
    KeyValuePair<TKey, TValue> keyValuePair
)

Parameters

keyValuePair
    Type: System.Collections.Generic..::.KeyValuePair(Of (TKey, TValue))

Implements

ICollection(Of (T)...)..::.Contains(T)
See Also

ConcurrentDictionary(Of (TKey, TValue)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets whether the IDictionary<(Of <(TKey, TValue)>)> contains an element with the specified key.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function Contains (_
    key As Object _
) As Boolean Implements IDictionary.Contains

C#

bool IDictionary.Contains(
    Object key
)

Parameters

key
    Type: System::Object
    The key to locate in the IDictionary<((TKey, TValue)>).

Return Value

true if the IDictionary<((TKey, TValue)> contains an element with the specified key; otherwise, false.

Implements

IDictionary::.Contains(Object)
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::ArgumentOutOfRangeException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>.::.ICollection<(Of <(KeyValuePair<(Of <(TKey, TValue)>))>)>.::.CopyTo Method

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class  See Also  Send Feedback

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Sub CopyTo ( _
    array As KeyValuePair(Of TKey, TValue)(), _
    index As Integer _
) Implements ICollection(Of KeyValuePair(Of TKey, TValue)).CopyTo

C#

void ICollection<KeyValuePair<TKey, TValue>>.CopyTo(
    KeyValuePair<TKey, TValue>[] array,
    int index
)

Parameters

array
    Type: array<System.Collections.Generic.KeyValuePair(Of TKey, TValue)>[]

index
    Type: System.Int32

Implements

ICollection(Of TKey, TValue).CopyTo(array[T]()[], Int32)
See Also

`ConcurrentDictionary(Of TKey, TValue) Class`
`System.Collections.Concurrent Namespace`

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>..::.ICollection..::.CopyTo Method

**ConcurrentDictionary<(Of <(TKey, TValue)>)> Class**  See Also  Send Feedback

Copies the elements of the ICollection to an array, starting at the specified array index.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Private Sub CopyTo (_
    array As Array, _
    index As Integer _
) Implements ICollection.CopyTo

**C#**

void ICollection.CopyTo(
    Array array,
    int index
)

**Parameters**

array
    Type: System:::Array
    The one-dimensional array that is the destination of the elements copied from the ICollection. The array must have zero-based indexing.

index
    Type: System:::Int32
    The zero-based index in array at which copying begins.

**Implements**

ICollection:::CopyTo(Array, Int32)
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>array is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>index is less than 0.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>index is equal to or greater than the length of the array. -or- The number of elements in the source ICollection is greater than the available space from index to the end of the destination array.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary(Of (TKey, TValue)>).::.IDictionary.::.GetEnumerator Method

ConcurrentDictionary(Of (TKey, TValue)>).::.IDictionary.::.GetEnumerator Method

Provides an IDictionaryEnumerator for the IDictionary(Of (TKey, TValue)>).

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Private Function GetEnumerator As IDictionaryEnumerator
Implements IDictionary.GetEnumerator

**C#**

IDictionaryEnumerator IDictionary.GetEnumerator()

**Return Value**

An IDictionaryEnumerator for the IDictionary(Of TKey, TValue>).

**Implements**

IDictionary...:::GetEnumerator()
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary(Of (TKey, TValue))::<IEnumerable>::GetEnumerator Method

ConcurrentDictionary(Of (TKey, TValue)) Class  See Also  Send Feedback

Returns an enumerator that iterates through the ConcurrentDictionary(Of (TKey, TValue)).

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function GetEnumerator As IEnumerator
    Implements IEnumerable.GetEnumerator

C#

IEnumerator IEnumerable.GetEnumerator()

Return Value

An enumerator for the `ConcurrentDictionary(Of (TKey, TValue))).

Implements

IEnumerable...::GetEnumerator()()
Remarks

The enumerator returned from the dictionary is safe to use concurrently with reads and writes to the dictionary, however it does not represent a moment-in-time snapshot of the dictionary. The contents exposed through the enumerator may contain modifications made to the dictionary after GetEnumerator() was called.
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
ConcurrentDictionary<(Of <(TKey, TValue)>)).::.ICollection<(Of <(KeyValuePair<(Of <(TKey, TValue)>))>)).::.Remove Method
ConcurrentDictionary<(Of <(TKey, TValue)>)> Class  See Also  Send Feedback

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Private Function Remove ( _
    keyValuePair As KeyValuePair(Of TKey, TValue) _
) As Boolean Implements ICollection(Of KeyValuePair(Of TKey, TValue))

**C#**

bool ICollection<KeyValuePair<TKey, TValue>>.Remove(
    KeyValuePair<TKey, TValue> keyValuePair
)

**Parameters**

keyValuePair
    Type: System.Collections.Generic..::.KeyValuePair<(Of <TKey, TValue>)>

**Implements**

ICollection<(Of <(T)>)..::.Remove(T)
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Removes the element with the specified key from the IDictionary<(Of <(TKey, TValue)>)>.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function Remove ( _
    key As TKey _
) As Boolean Implements IDictionary(Of TKey, TValue).Remove

C#

bool IDictionary<TKey, TValue>.Remove(
    TKey key
)

Parameters

key
    Type: TKey
    The key of the element to remove.

Return Value

true if the element is successfully remove; otherwise false. This method also returns false if key was not found in the original IDictionary<(Of <<(TKey, TValue>>)>. 

Implements

IDictionary<(Of <<(TKey, TValue>>)>.Remove(TKey)
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

`ConcurrentDictionary<,(Of <(TKey, TValue)>)> Class`  
`System.Collections.Concurrent Namespace`

Send feedback on this topic to Microsoft.
ConcurrentDictionary(Of TKey, TValue)::.IDictionary::.Remove Method

ConcurrentDictionary(Of TKey, TValue) Class  See Also  Send Feedback

Removes the element with the specified key from the IDictionary.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Private Sub Remove ( _
    key As Object _
  ) Implements IDictionary.Remove
```

**C#**

```csharp
void IDictionary.Remove(
    Object key
)
```

### Parameters

**key**

Type: System::Object

The key of the element to remove.

### Implements

IDictionary:::Remove(Object)
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
ConcurrentDictionary(Of (TKey, TValue))::.AddOrUpdate Method
ConcurrentDictionary(Of (TKey, TValue)) Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddOrUpdate(TKey, Func&lt;(Of &lt;(TKey, TValue)&gt;), Func&lt;(Of &lt;(TKey, TValue, TValue)&gt;)&gt;)</td>
<td>Adds a key/value pair to the ConcurrentDictionary&lt;(Of &lt;(TKey, TValue)&gt;)&gt; if the key does not already exist, or updates a key/value pair in the ConcurrentDictionary&lt;(Of &lt;(TKey, TValue)&gt;)&gt; if the key already exists.</td>
</tr>
<tr>
<td>AddOrUpdate(TKey, TValue, Func&lt;(Of &lt;(TKey, TValue, TValue)&gt;)&gt;)</td>
<td>Adds a key/value pair to the ConcurrentDictionary&lt;(Of &lt;(TKey, TValue)&gt;)&gt; if the key does not already exist, or updates a key/value pair in the ConcurrentDictionary&lt;(Of &lt;(TKey, TValue)&gt;)&gt; if the key already exists.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>. AddOrUpdate Method (TKey, Func<(Of <(TKey, TValue)>)>, Func<(Of <(TKey, TValue, TValue)>)>))

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class  See Also  Send Feedback

Adds a key/value pair to the ConcurrentDictionary<(Of <(TKey, TValue)>)> if the key does not already exist, or updates a key/value pair in the ConcurrentDictionary<(Of <(TKey, TValue)>)> if the key already exists.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function AddOrUpdate ( _
    key As TKey, _
    addValueFactory As Func(Of TKey, TValue), _
    updateValueFactory As Func(Of TKey, TValue, TValue) _
) As TValue
```

### C#

```csharp
public TValue AddOrUpdate(
    TKey key,
    Func<TKey, TValue> addValueFactory,
    Func<TKey, TValue, TValue> updateValueFactory
)
```

## Parameters

**key**

Type: **TKey**
The key to be added or whose value should be updated

**addValueFactory**

Type: System...:::Func<(Of <(TKey, TValue)>)>)
The function used to generate a value for an absent key

**updateValueFactory**

Type: System...:::Func<(Of <(TKey, TValue, TValue)>)>)
The function used to generate a new value for an existing key based on the key's existing value

## Return Value

The new value for the key. This will be either be the result of addValueFactory (if the key was absent) or the result of updateValueFactory (if the key was
present).
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>addValueFactory is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>updateValueFactory is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...:::OverflowException</td>
<td>The dictionary contains too many elements.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)) Class
AddOrUpdate Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>.AddOrUpdate Method (TKey, TValue, Func<(Of <(TKey, TValue, TValue)>)>)

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class See Also Send Feedback

Add a key/value pair to the ConcurrentDictionary<(Of <(TKey, TValue)>)> if the key does not already exist, or updates a key/value pair in the ConcurrentDictionary<(Of <(TKey, TValue)>)> if the key already exists.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function AddOrUpdate ( _
    key As TKey, _
    addValue As TValue, _
    updateValueFactory As Func(Of TKey, TValue, TValue) _
) As TValue

C#

public TValue AddOrUpdate(  
    TKey key,  
    TValue addValue,  
    Func<TKey, TValue, TValue> updateValueFactory  
)

Parameters

key
    Type: TKey
    The key to be added or whose value should be updated

addValue
    Type: TValue
    The value to be added for an absent key

updateValueFactory
    Type: System..::.Func(Of <(TKey, TValue, TValue)>)
    The function used to generate a new value for an existing key based on the key's existing value

Return Value

The new value for the key. This will be either be the result of addValueFactory (if the key was absent) or the result of updateValueFactory (if the key was
present).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>updateValueFactory is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.OverflowException</td>
<td>The dictionary contains too many elements.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
AddOrUpdate Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Removes all keys and values from the `ConcurrentDictionary<(Of <(TKey, TValue)>)>`.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Clear

C#

public void Clear()

Implements

IICollection(Of T)::.Clear()
IDictionary::.Clear()
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Determines whether the `ConcurrentDictionary(Of TKey, TValue)`.::ContainsKey Method contains the specified key.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function ContainsKey ( _
    key As TKey _
) As Boolean
```

**C#**

```csharp
public bool ContainsKey(
    TKey key
)
```

**Parameters**

key
Type: `TKey`
The key to locate in the `ConcurrentDictionary(Of (TKey, TValue)>).`

**Return Value**

true if the `ConcurrentDictionary(Of (TKey, TValue)>)` contains an element with the specified key; otherwise, false.

**Implements**

`IDictionary(Of (TKey, TValue)>).ContainsKey(TKey)`
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Returns an enumerator that iterates through the `ConcurrentDictionary<(Of <(TKey, TValue)>)>.GetEnumerator Method`.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Function GetEnumerator As IEnumerator(Of KeyValuePair(Of TKey, TValue))

C#
public IEnumerator<KeyValuePair<TKey, TValue>> GetEnumerator()

Return Value
An enumerator for the ConcurrentDictionary(Of (Of TKey, TValue))

Implements
IEnumerable(Of (Of T))...GetEnumerator()()}
Remarks

The enumerator returned from the dictionary is safe to use concurrently with reads and writes to the dictionary, however it does not represent a moment-in-time snapshot of the dictionary. The contents exposed through the enumerator may contain modifications made to the dictionary after GetEnumerator() was called.
See Also

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
ConcurrentDictionary(Of (TKey, TValue)) Class
See Also
Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetOrAdd(TKey, Func(Of&lt;(TKey, TValue)&gt;))</td>
<td>Adds a key/value pair to the ConcurrentDictionary(Of&lt;(TKey, TValue)&gt;), if the key does not already exist.</td>
</tr>
<tr>
<td>GetOrAdd(TKey, TValue)</td>
<td>Adds a key/value pair to the ConcurrentDictionary(Of&lt;(TKey, TValue)&gt;), if the key does not already exist.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>\textvisiblespace\textvisiblespace\textvisiblespace::GetOrAdd Method (TKey, Func<(Of <(TKey, TValue)>)>)

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class

See Also

Send Feedback

Adds a key/value pair to the ConcurrentDictionary<(Of <(TKey, TValue)>)> if the key does not already exist.

Namespace: System.Collections.Concurrent

Assembly: System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function GetOrAdd (_
    key As TKey,
    valueFactory As Func(Of TKey, TValue)
) As TValue
```

**C#**

```csharp
public TValue GetOrAdd(
    TKey key,
    Func<TKey, TValue> valueFactory
)
```

**Parameters**

- **key**
  - Type: **TKey**
  - The key of the element to add.

- **valueFactory**
  - Type: System::Func(Of **(TKey, TValue)>)
  - The function used to generate a value for the key

**Return Value**

The value for the key. This will be either the existing value for the key if the key is already in the dictionary, or the new value for the key as returned by `valueFactory` if the key was not in the dictionary.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>valueFactory is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::OverflowException</td>
<td>The dictionary contains too many elements.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class
GetOrAdd Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Adds a key/value pair to the `ConcurrentDictionary<Of <(TKey, TValue)>)` if the key does not already exist.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function GetOrAdd ( _
    key As TKey, _
    value As TValue _
) As TValue

C#

public TValue GetOrAdd(
    TKey key,
    TValue value
)

Parameters

key
Type: TKey
The key of the element to add.

value
Type: TValue
the value to be added, if the key does not already exist

Return Value

The value for the key. This will be either the existing value for the key if the key is already in the dictionary, or the new value if the key was not in the dictionary.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::OverflowException</td>
<td>The dictionary contains too many elements.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary<(Of (TKey, TValue)>) Class
GetOrAdd Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue>>)>::ToArray Method

Copies the key and value pairs stored in the ConcurrentDictionary<(Of <(TKey, TValue)>)> to a new array.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ToArray As KeyValuePair(Of TKey, TValue)()

C#

public KeyValuePair<TKey, TValue>[] ToArray()

Return Value

A new array containing a snapshot of key and value pairs copied from the ConcurrentDictionary(Of KeyValuePair(Of TKey, TValue)>).
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add the specified key and value to the `ConcurrentDictionary<(Of <(TKey, TValue)>)>`. 

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Public Function TryAdd (_
    key As TKey, _
    value As TValue _
) As Boolean

C#

public bool TryAdd(
    TKey key,
    TValue value
)

Parameters

key
   Type: TKey
   The key of the element to add.

value
   Type: TValue
   The value of the element to add. The value can be a null reference (Nothing in Visual Basic) for reference types.

Return Value

true if the key/value pair was added to the ConcurrentDictionary(Of (TKey, TValue)) successfully; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>key is null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::OverflowException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentdictionary-1">ConcurrentDictionary</a> contains too many elements.</td>
</tr>
</tbody>
</table>
**See Also**

*ConcurrentDictionary*(Of *(TKey, TValue)*<>*) Class
*System.Collections.Concurrent Namespace*

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Attempts to get the value associated with the specified key from the `ConcurrentDictionary(Of TKey, TValue)`.  


**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryGetValue ( _
    key As TKey, _
    <OutAttribute> ByRef value As TValue _
) As Boolean

C#

public bool TryGetValue(
    TKey key,
    out TValue value
)

Parameters

key
  Type: TKey
  The key of the value to get.

value
  Type: TValue
  When this method returns, value contains the object from the
  ConcurrentDictionary(Of TKey, TValue) with the specified key or
  the default value of TValue, if the operation failed.

Return Value

true if the key was found in the ConcurrentDictionary(Of TKey, TValue); otherwise, false.

Implements

IDictionary(Of TKey, TValue).TryGetValue(TKey, TValue%)
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove and return the value with the specified key from the `ConcurrentDictionary<(Of <(TKey, TValue)>)>`.  

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryRemove ( _
    key As TKey, _
    <OutAttribute> ByRef value As TValue _
) As Boolean

C#

public bool TryRemove(
    TKey key,
    out TValue value
)

Parameters

key
Type: TKey
The key of the element to remove and return.

value
Type: TValue
When this method returns, value contains the object removed from the ConcurrentDictionary(Of<TKey, TValue>) or the default value of TValue if the operation failed.

Return Value

true if an object was removed successfully; otherwise, false.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<(Of <(TKey, TValue)>)>.TryUpdate Method

ConcurrentDictionary<(Of <(TKey, TValue)>)> Class  See Also  Send Feedback

Compares the existing value for the specified key with a specified value, and if they’re equal, updates the key with a third value.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function TryUpdate (_
    key As TKey, _
    newValue As TValue, _
    comparisonValue As TValue _
) As Boolean
```

**C#**

```csharp
public bool TryUpdate(
    TKey key,
    TValue newValue,
    TValue comparisonValue
)
```

**Parameters**

**key**
Type: `TKey`
The key whose value is compared with `comparisonValue` and possibly replaced.

**newValue**
Type: `TValue`
The value that replaces the value of the element with `key` if the comparison results in equality.

**comparisonValue**
Type: `TValue`
The value that is compared to the value of the element with `key`.

**Return Value**

true if the value with `key` was equal to `comparisonValue` and replaced with
newValue; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>key is a null reference.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `ConcurrentDictionary(Of TKey, TValue)<>` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Gets the number of key/value pairs contained in the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt;&gt;</code>.</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>Gets a value that indicates whether the <code>ConcurrentDictionary&lt;Of &lt;(TKey, TValue)&gt;&gt;</code> is empty.</td>
</tr>
<tr>
<td>Item</td>
<td>Gets or sets the value associated with the specified key.</td>
</tr>
<tr>
<td>Keys</td>
<td>Gets a collection containing the keys in the Dictionary&lt;Of &lt;(TKey, TValue)&gt;&gt;.</td>
</tr>
<tr>
<td>Values</td>
<td>Gets a collection containing the values in the Dictionary&lt;Of &lt;(TKey, TValue)&gt;&gt;.</td>
</tr>
</tbody>
</table>
# Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IDictionary::&lt;T&gt;.::IsFixedSize</code></td>
<td>Gets a value indicating whether the IDictionary&lt;Of (TKey, TValue)&gt; has a fixed size.</td>
</tr>
<tr>
<td><code>ICollection&lt;Of (KeyValuePair&lt;T&gt;).&gt;::IsReadOnly</code></td>
<td>Gets a value indicating whether the IDictionary&lt;Of (TKey, TValue)&gt; is read-only.</td>
</tr>
<tr>
<td><code>IDictionary::&lt;T&gt;.::IsReadOnly</code></td>
<td>Gets a value indicating whether access to the ICollection is synchronized with the SyncRoot.</td>
</tr>
<tr>
<td><code>ICollection::&lt;T&gt;.::IsSynchronized</code></td>
<td>Gets a value indicating whether the specified key.</td>
</tr>
<tr>
<td><code>IDictionary::&lt;T&gt;.::Item</code></td>
<td>Gets or sets the value associated with the specified key.</td>
</tr>
<tr>
<td><code>IDictionary::&lt;T&gt;.::Keys</code></td>
<td>Gets an ICollection containing the keys of the IDictionary&lt;Of (TKey, TValue)&gt;).</td>
</tr>
<tr>
<td><code>ICollection::&lt;T&gt;.::SyncRoot</code></td>
<td>Gets an object that can be used to synchronize access to the ICollection. This property is not supported.</td>
</tr>
<tr>
<td><code>IDictionary::&lt;T&gt;.::Values</code></td>
<td>Gets an ICollection containing the values in the IDictionary.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value indicating whether the IDictionary\<(Of \<(TKey, TValue\)>\> has a fixed size.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Private Readonly Property IsFixedSize As Boolean
    Implements IDictionary.IsFixedSize
```

**C#**

```csharp
bool IDictionary.IsFixedSize { get; }
```

**Field Value**

true if the IDictionary<(Of <(TKey, TValue)>)> has a fixed size; otherwise, false. For ConcurrentDictionary<(Of <(TKey, TValue)>)>), this property always returns false.

**Implements**

IDictionary...:..IsFixedSize
See Also

ConcurrentDictionary(Of (TKey, TValue)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library
ConcurrentDictionary(Of (TKey, TValue))::ICollection(Of KeyValuePair(Of (TKey, TValue)))::.IsReadOnly Property
ConcurrentDictionary(Of (TKey, TValue)) Class

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private ReadOnly Property IsReadOnly As Boolean
    Implements ICollection(Of KeyValuePair(Of TKey, TValue)).IsF

C#

bool ICollection<KeyValuePair<TKey, TValue>>.IsReadOnly { get; }  

Implements

ICollection(Of ((T)))...:..IsReadOnly
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value indicating whether the IDictionary<(Of <(TKey, TValue)>)> is read-only.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Private Readonly Property IsReadOnly As Boolean
Implements IDictionary.IsReadOnly

**C#**

bool IDictionary.IsReadOnly { get; }

**Field Value**

true if the IDictionary(Of TKey, TValue>) is read-only; otherwise, false. For ConcurrentDictionary(Of<TKey, TValue>), this property always returns false.

**Implements**

IDictionary..::.IsReadOnly
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value indicating whether access to the ICollection is synchronized with the SyncRoot.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Readonly Property IsSynchronized As Boolean
Implements ICollection.IsSynchronized

C#

bool ICollection.IsSynchronized { get; }

Field Value

ture if access to the ICollection is synchronized (thread safe); otherwise, false. For ConcurrentDictionary(Of TKey, TValue), this property always returns false.

Implements

ICollection..::.IsSynchronized
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets or sets the value associated with the specified key.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Property Item (_
    key As Object _
) As Object Implements IDictionary.Item

C#

Object IDictionary.Item[
    Object key
] { get; set; }

Parameters

key
    Type: System..::.Object
    The key of the value to get or set.

Field Value

The value associated with the specified key, or a null reference (Nothing in Visual Basic) if key is not in the dictionary or key is of a type that is not assignable to the key type TKey of the ConcurrentDictionary<(Of <(TKey, TValue)>)).

Implements

IDictionary..::.Item[([(Object)])]
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td></td>
<td>A value is being assigned, and key is of a type that is not assignable to the key type TKey of the ConcurrentDictionary(Of&lt;TKey, TValue&gt; ). -or- A value is being assigned, and key is of a type that is not assignable to the value type TValue of the ConcurrentDictionary(Of&lt;TKey, TValue&gt; ).</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary(Of (TKey, TValue))::IDictionary::Keys Property

ConcurrentDictionary(Of (TKey, TValue)) Class

See Also

Send Feedback

Gets an ICollection containing the keys of the IDictionary(Of (TKey, TValue)).

Namespace: System.Collections.Concurrent

Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Readonly Property Keys As ICollection
    Implements IDictionary.Keys

C#

ICollection IDictionary.Keys { get; }

Field Value

An ICollection containing the keys of the IDictionary(Of (TKey, TValue)).

Implements

IDictionary..::.Keys
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
.NET Framework Class Library
ConcurrentDictionary(Of (TKey, TValue))..::.ICollection..::.SyncRoot Property

ConcurrentDictionary(Of (TKey, TValue)) Class  See Also  Send Feedback

Gets an object that can be used to synchronize access to the ICollection. This property is not supported.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Private Readonly Property SyncRoot As Object
Implements ICollection.SyncRoot

**C#**

Object ICollection.SyncRoot { get; }

**Implements**

ICollection...:..SyncRoot
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>The SyncRoot property is not supported.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of (TKey, TValue)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentDictionary<( Of <(TKey, TValue)> )> .::. IDictionary .::. Values Property

ConcurrentDictionary<( Of <(TKey, TValue)> )> Class  See Also  Send Feedback

Gets an ICollection containing the values in the IDictionary.

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private ReadOnly Property Values As ICollection
Implements IDictionary.Values

C#

ICollection IDictionary.Values { get; }

Field Value

An ICollection containing the values in the IDictionary.

Implements

IDictionary..::.Values
See Also

ConcurrentDictionary(Of (TKey, TValue)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets the number of key/value pairs contained in the `ConcurrentDictionary<(Of <(TKey, TValue)>)>.`

**Namespace:** `System.Collections.Concurrent`

**Assembly:** `System.Threading` (in `System.Threading.dll`)

---

Visual Basic  □  C#

.NET Framework Class Library

`ConcurrentDictionary<(Of <(TKey, TValue)>)>`. Count Property

`ConcurrentDictionary<(Of <(TKey, TValue)>)>` Class  See Also  Send Feedback
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Count As Integer

C#

public int Count { get; }

Field Value

The number of key/value pairs contained in the `ConcurrentDictionary(Of TKey, TValue)`. 

Implements

`ICollection(Of T)`.:::Count
`ICollection`.:::Count
Remarks

Count has snapshot semantics and represents the number of items in the `ConcurrentDictionary<Of<(TKey, TValue)>>` at the moment when Count was accessed.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.OverflowException</td>
<td>The dictionary contains too many elements.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value that indicates whether the `ConcurrentDictionary(Of TKey, TValue)` is empty.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsEmpty As Boolean

C#

public bool IsEmpty { get; }

Field Value

ture if the **ConcurrentDictionary(Of TKey, TValue>)** is empty; otherwise, false.
See Also

ConcurrentDictionary(Of (TKey, TValue)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets or sets the value associated with the specified key.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Property Item ( _
   key As TKey _
) As TValue
```

**C#**

```csharp
public TValue Item[
   TKey key
] { get; set; }
```

**Parameters**

- **key**
  - Type: `TKey`
  - The key of the value to get or set.

**Field Value**

The value associated with the specified key. If the specified key is not found, a get operation throws a `KeyNotFoundException`, and a set operation creates a new element with the specified key.

**Implements**

`IDictionary<(Of `<(TKey, TValue)>`)>>(IDictionary<(Of `<(TKey, TValue)>`)>>.`
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>key is a null reference (Nothing in Visual Basic). The property is retrieved and key does not exist in the collection.</td>
</tr>
<tr>
<td>System.Collections.Generic..::.KeyNotFoundException</td>
<td></td>
</tr>
</tbody>
</table>

- System..::.ArgumentNullException
- System.Collections.Generic..::.KeyNotFoundException
See Also

ConcurrentDictionary(Of (Of TKey, TValue)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a collection containing the keys in the Dictionary\<(Of \<(TKey, TValue)>\)>.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property Keys As ICollection(Of TKey)

C#

public ICollection<TKey> Keys { get; }

Field Value

An ICollection(Of TKey) containing the keys in the Dictionary(Of TKey, TValue).

Implements

IDictionary(Of TKey, TValue).::.Keys
See Also

**ConcurrentDictionary(Of TKey, TValue) Class**
System.Collections.Concurrent Namespace

Send [feedback](mailto:support@microsoft.com) on this topic to Microsoft.
Gets a collection containing the values in the Dictionary<(TKey, TValue)>.

**Namespace:** System.Collections.Concurrent
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Values As ICollection(Of TValue)

C#

public ICollection<TValue> Values { get; }

Field Value

An ICollection(Of (Of TValue>) containing the values in the Dictionary(Of (Of TKey, TValue>)

Implements

IDictionary(Of (Of TKey, TValue>)..::.Values
See Also

ConcurrentDictionary(Of TKey, TValue) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Represents a thread-safe first-in, first-out collection of objects.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
<SerializableAttribute> _
<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
  ExternalThreading := True)> _
Public Class ConcurrentQueue(Of T) _
    Implements IProducerConsumerCollection(Of T), IEnumerable(Of ICollection, IEnumerable

C#

[SerializableAttribute]
[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
  ExternalThreading = true)]
public class ConcurrentQueue<T> : IProducerConsumerCollection<T>, IEnumerable<T>, ICollection, IEnumerable
```
Type Parameters

T

Specifies the type of elements in the queue.
Remarks

All public and protected members of ConcurrentQueue<Of <(T)>>) are thread-safe and may be used concurrently from multiple threads.
Inheritance Hierarchy

System..::.Object
System.Collections.Concurrent..::.ConcurrentQueue<(Of <(T)>)>
See Also

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentQueue<(Of <(T)>)> Constructor

ConcurrentQueue<(Of <(T)>)> Class

See Also

Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConcurrentQueue&lt;Of &lt;(T)&gt;(&gt;)&gt;()()</code></td>
<td>Initializes a new instance of the <code>ConcurrentQueue&lt;Of &lt;(T)&gt;(&gt;)&gt;()</code> class.</td>
</tr>
<tr>
<td><code>ConcurrentQueue&lt;Of &lt;(T)&gt;(&gt;)((IEnumerable&lt;Of &lt;(T)&gt;(&gt;))())</code></td>
<td>Initializes a new instance of the <code>ConcurrentQueue&lt;Of &lt;(T)&gt;(&gt;)((IEnumerable&lt;Of &lt;(T)&gt;(&gt;))()</code> class that contains elements copied from the specified collection</td>
</tr>
</tbody>
</table>
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentQueue<Of <(T)>)` class.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public ConcurrentQueue()
See Also

ConcurrentQueue<Of<T>> Class
ConcurrentQueue<Of<T>> Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentQueue<T>` class that contains elements copied from the specified collection.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    collection As IEnumerable(Of T) _
)

C#

public ConcurrentQueue(  
    IEnumerable<T> collection  
)

Parameters

collection

Type: System.Collections.Generic.IEnumerable(Of (T))

The collection whose elements are copied to the new ConcurrentQueue(Of (T)).
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The collection argument is null.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentQueue<Of <(T)>> Class
ConcurrentQueue<Of <(T)>> Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
ConcurrentQueue<(Of <(T)>)> Methods

The ConcurrentQueue<(Of <(T)>)> type exposes the following members.
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyTo</td>
<td>Copies the <code>ConcurrentQueue&lt; Of &lt;(T)&gt; &gt;</code> elements to an existing one-dimensional Array, starting at the specified array index.</td>
</tr>
<tr>
<td>Enqueue</td>
<td>Adds an object to the end of the <code>ConcurrentQueue&lt; Of &lt;(T)&gt; &gt;</code>.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetEnumerator</td>
<td>Returns an enumerator that iterates through the <code>ConcurrentQueue&lt; Of &lt;(T)&gt; &gt;</code>.</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToArray</td>
<td>Copies the elements stored in the <code>ConcurrentQueue&lt; Of &lt;(T)&gt; &gt;</code> to a new array.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>TryDequeue</td>
<td>Attempts to remove and return the object at the beginning of the <code>ConcurrentQueue&lt; Of &lt;(T)&gt; &gt;</code>.</td>
</tr>
<tr>
<td>TryPeek</td>
<td>Attempts to return an object from the beginning of the <code>ConcurrentQueue&lt; Of &lt;(T)&gt; &gt;</code> without removing it.</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICollection:::CopyTo</td>
<td>Copies the elements of the ICollection to an Array, starting at a particular Array index.</td>
</tr>
<tr>
<td>IEnumerable:::GetEnumerator</td>
<td>Returns an enumerator that iterates through a collection.</td>
</tr>
</tbody>
</table>
| IProducerConsumerCollection<T>:::TryAdd | Attempts to add an object to the IProducerConsumerCollection<T>..
| IProducerConsumerCollection<T>:::TryTake | Attempts to remove and return an object from the IProducerConsumerCollection<T>.. |
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
C# .NET Framework Class Library
ConcurrentQueue<(Of (T)>).::.ICollection.::.CopyTo Method

ConcurrentQueue<(Of (T)>).::.ICollection.::.CopyTo Method

Copies the elements of the ICollection to an Array, starting at a particular Array index.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

```
Private Sub CopyTo ( _
    array As Array, _
    index As Integer _
) Implements ICollection.CopyTo
```

C#

```
void ICollection.CopyTo(
    Array array,
    int index
)
```

Parameters

array
Type: System..::.Array
The one-dimensional Array that is the destination of the elements copied from the ConcurrentBag. The Array must have zero-based indexing.

index
Type: System..::.Int32
The zero-based index in array at which copying begins.

Implements

ICollection..::.CopyTo(Array, Int32)
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>array is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>index is less than zero.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>array is multidimensional. -or- array does not have zero-based indexing. -or- index is equal to or greater than the length of the array -or- The number of elements in the source ICollection is greater than the available space from index to the end of the destination array. -or- The type of the source ICollection cannot be cast automatically to the type of the destination array.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Returns an enumerator that iterates through a collection.

**Namespace:**  [System.Collections.Concurrent](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent)

**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Private Function GetEnumerator As IEnumerator
Implements IEnumerable.GetEnumerator

**C#**

IEnumerator IEnumerable.GetEnumerator()

**Return Value**

An IEnumerator that can be used to iterate through the collection.

**Implements**

IEnumerable...::GetEnumerator()()
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Copies the `ConcurrentQueue<(Of <(T)>)>` elements to an existing one-dimensional Array, starting at the specified array index.

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub CopyTo (_
    array As T(), _
    index As Integer _
)

C#

public void CopyTo(
    T[] array,
    int index
)

Parameters

array
    Type: array< T >[][]
The one-dimensional Array that is the destination of the elements copied from the ConcurrentQueue< Of <(T)> >. The Array must have zero-based indexing.

index
    Type: System..::.Int32
    The zero-based index in array at which copying begins.

Implements

IProducerConsumerCollection< Of <(T)> >,..;, CopyTo(array<T>[],[], Int32)
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>array is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>index is less than zero.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>index is equal to or greater than the length of the array -or- The number of elements in the source <strong>ConcurrentQueue&lt;Of &lt;(T)&gt;)</strong> is greater than the available space from index to the end of the destination array.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentQueue<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Adds an object to the end of the **ConcurrentQueue<Of <(T)>>**.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Sub Enqueue ( _
    item As T _
)
```

**C#**

```csharp
public void Enqueue(
    T item
)
```

**Parameters**

`item`  
Type: `T`  
The object to add to the end of the `ConcurrentQueue(Of `(T`)`)`. The value can be a null reference (Nothing in Visual Basic) for reference types.
See Also

ConcurrentQueue<(Of <(T)>>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentQueue<(Of <(T)>)>...::GetEnumerator Method

ConcurrentQueue<(Of <(T)>)> Class  See Also  Send Feedback

Returns an enumerator that iterates through the ConcurrentQueue<(Of <(T)>).

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function GetEnumerator As IEnumerator(Of T)

C#

public IEnumerator<T> GetEnumerator()

Return Value

An enumerator for the contents of the ConcurrentQueue(Of (Of T)).
Remarks

The enumeration represents a moment-in-time snapshot of the contents of the queue. It does not reflect any updates to the collection after GetEnumerator() was called. The enumerator is safe to use concurrently with reads from and writes to the queue.
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add an object to the `IProducerConsumerCollection<Of <(T)▻)`.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function TryAdd ( _
    item As T _
) As Boolean Implements IProducerConsumerCollection(Of T).TryAdd

C#

bool IProducerConsumerCollection<T>.TryAdd(
    T item
)

Parameters

item
    Type: T
    The object to add to the IProducerConsumerCollection(Of <(T)>). The value can be a null reference (Nothing in Visual Basic) for reference types.

Return Value

true if the object was added successfully; otherwise, false.

Implements

IProducerConsumerCollection(Of <(T)>).TryAdd(T)
Remarks

For `ConcurrentQueue<Of <(T)>`, this operation will always add the object to the end of the `ConcurrentQueue<Of <(T)>` and return true.
See Also

**ConcurrentQueue(Of `(T)`>) Class**
*System.Collections.Concurrent Namespace*

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Attempts to remove and return an object from the
IProducerConsumerCollection<(Of <(T)>).

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function TryTake ( _
  <OutAttribute> ByRef item As T _
) As Boolean Implements IProducerConsumerCollection(Of T).TryTake

C#

bool IProducerConsumerCollection<T>.TryTake (out T item)

Parameters

item

Type: T %
When this method returns, if the operation was successful, item contains the object removed. If no object was available to be removed, the value is unspecified.

Return Value

true if an element was removed and returned succesfully; otherwise, false.

Implements

IProducerConsumerCollection(Of <T>).TryTake(T%)
Remarks

For ConcurrentQueue<Of (T)>, this operation will attempt to remove the object from the beginning of the ConcurrentQueue<Of (T)>.
See Also

ConcurrentQueue<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Copies the elements stored in the `ConcurrentQueue<(Of <(T)>)` to a new array.

**Namespace:** `System.Collections.Concurrent`  
**Assembly:** `System.Threading` (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function ToArray As T()
```

**C#**

```csharp
public T[] ToArray()
```

**Return Value**

A new array containing a snapshot of elements copied from the `ConcurrentQueue<Of <T>>`.

**Implements**

`IProducerConsumerCollection<Of <T>>,..:ToArray()()`
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove and return the object at the beginning of the `ConcurrentQueue(Of T)`. 

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryDequeue ( _
    <OutAttribute> ByRef result As T _
) As Boolean

C#

public bool TryDequeue(
    out T result
)

Parameters

result
    Type: T %
    When this method returns, if the operation was successful, result contains the object removed. If no object was available to be removed, the value is unspecified.

Return Value

true if an element was removed and returned from the beggining of the ConcurrentQueue(Of (T)>) succesfully; otherwise, false.
See Also

ConcurrentQueue<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to return an object from the beginning of the `ConcurrentQueue<(Of <(T)>))` without removing it.

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading` (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryPeek ( _
            <OutAttribute> ByRef result As T _
    ) As Boolean

C#

public bool TryPeek(
    out T result
)

Parameters

result
    Type: T %
    When this method returns, result contains an object from the beginning of the ConcurrentQueue(Of (T)> or an unspecified value if the operation failed.

Return Value

true if and object was returned successfully; otherwise, false.
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `ConcurrentQueue<(Of <(T)>)>` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Gets the number of elements contained in the <code>ConcurrentQueue&lt;Of (T)&gt;</code>.</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>Gets a value that indicates whether the <code>ConcurrentQueue&lt;Of (T)&gt;</code> is empty.</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ICollection::&lt;&gt; IsSynchronized</code></td>
<td>Gets a value indicating whether access to the ICollection is synchronized with the SyncRoot.</td>
</tr>
<tr>
<td><code>ICollection::&lt;&gt; SyncRoot</code></td>
<td>Gets an object that can be used to synchronize access to the ICollection. This property is not supported.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentQueue<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value indicating whether access to the ICollection is synchronized with the SyncRoot.

**Namespace:**  System.Collections.Concurrent

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private ReadOnly Property IsSynchronized As Boolean
Implements ICollection.IsSynchronized

C#

bool ICollection.IsSynchronized { get; }

Field Value

ture if access to the ICollection is synchronized with the SyncRoot; otherwise, false. For ConcurrentQueue(Of T), this property always returns false.

Implements

ICollection...:::IsSynchronized
See Also

ConcurrentQueue<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets an object that can be used to synchronize access to the ICollection. This property is not supported.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Private Readonly Property SyncRoot As Object
Implements ICollection.SyncRoot

**C#**

Object ICollection.SyncRoot { get; }

**Implements**

ICollection..::.SyncRoot
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.NotSupportedException</code></td>
<td>The SyncRoot property is not supported.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentQueue<(Of <(T)>)> Class

Gets the number of elements contained in the ConcurrentQueue<(Of <(T)>)>.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Count As Integer

C#

public int Count { get; }

Field Value

The number of elements contained in the ConcurrentQueue(Of T).

Implements

ICollection.Count
Remarks

For determining whether the collection contains any items, use of the IsEmpty property is recommended rather than retrieving the number of items from the Count property and comparing it to 0.
See Also

ConcurrentQueue(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentQueue<(Of <(T)>)>::IsEmpty Property

Gets a value that indicates whether the ConcurrentQueue<(Of <(T)>)> is empty.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsEmpty As Boolean

C#

public bool IsEmpty { get; }

Field Value

ture if the ConcurrentQueue(Of (T)> is empty; otherwise, false.
Remarks

For determining whether the collection contains any items, use of this property is recommended rather than retrieving the number of items from the `Count` property and comparing it to 0. However, as this collection is intended to be accessed concurrently, it may be the case that another thread will modify the collection after `IsEmpty` returns, thus invalidating the result.
See Also

ConcurrentQueue(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Represents a thread-safe last-in, first-out collection of objects.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

```vbnet
<SerializableAttribute> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading := True)> _
Public Class ConcurrentStack(Of T) _
   Implements IProducerConsumerCollection(Of T), IEnumerable(Of T), ICollection, IEnumerable
```

C#

```csharp
[SerializableAttribute]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading = true)]
public class ConcurrentStack<T> : IProducerConsumerCollection<T>,
   IEnumerable<T>, ICollection, IEnumerable
```
Type Parameters

T

Specifies the type of elements in the stack.
Remarks

All public and protected members of ConcurrentStack<Of <(T)>>) are thread-safe and may be used concurrently from multiple threads.
Inheritance Hierarchy

System...:::Object
System.Collections.Concurrent...:::ConcurrentStack<Of <(T)>>>>
See Also

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentStack(Of (T)>) Constructor

ConcurrentStack(Of (T)>) Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConcurrentStack&lt;Of </code>&lt;T&gt;<code>&gt;()()</code></td>
<td>Initializes a new instance of the <code>ConcurrentStack&lt;Of </code>&lt;T&gt;<code>&gt;()</code> class.</td>
</tr>
<tr>
<td><code>ConcurrentStack&lt;Of </code>&lt;T&gt;<code>&gt;()IEnumerator&lt;Of </code>&lt;T&gt;<code>())</code></td>
<td>Initializes a new instance of the <code>ConcurrentStack&lt;Of </code>&lt;T&gt;<code>&gt;()</code> class that contains elements copied from the specified collection</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentStack(Of <(T)>)` class.

**Namespace:** `System.Collections.Concurrent`  
**Assembly:** `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public ConcurrentStack()
See Also

ConcurrentStack<(Of <(T)>)> Class
ConcurrentStack<(Of <(T)>)> Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ConcurrentStack(Of <T>)` class that contains elements copied from the specified collection.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)


Syntax

Visual Basic (Declaration)

Public Sub New ( _
    collection As IEnumerable(Of T) _
)

C#

public ConcurrentStack(
    IEnumerable<T> collection
)

Parameters

collection
    Type: System.Collections.Generic.IEnumerable(Of T)
The collection whose elements are copied to the new ConcurrentStack(Of T).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The collection argument is null.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack<(Of <(T)>)> Class
ConcurrentStack<(Of <(T)>)> Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `ConcurrentStack<Of <(T)>>` type exposes the following members.
# Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clear</strong></td>
<td>Removes all objects from the <code>ConcurrentStack&lt;Of &lt;(T)&gt; &gt;</code>.</td>
</tr>
<tr>
<td><strong>CopyTo</strong></td>
<td>Copies the <code>ConcurrentStack&lt;Of &lt;(T)&gt; &gt;</code> elements to an existing one-dimensional Array, starting at the specified array index.</td>
</tr>
<tr>
<td><strong>Equals</strong></td>
<td>(Inherited from <code>Object</code>.)</td>
</tr>
<tr>
<td><strong>Finalize</strong></td>
<td>(Inherited from <code>Object</code>).</td>
</tr>
<tr>
<td><strong>GetEnumerator</strong></td>
<td>Returns an enumerator that iterates through the <code>ConcurrentStack&lt;Of &lt;(T)&gt; &gt;</code>.</td>
</tr>
<tr>
<td><strong>GetHashCode</strong></td>
<td>(Inherited from <code>Object</code>).</td>
</tr>
<tr>
<td><strong>GetType</strong></td>
<td>(Inherited from <code>Object</code>).</td>
</tr>
<tr>
<td><strong>MemberwiseClone</strong></td>
<td>(Inherited from <code>Object</code>).</td>
</tr>
<tr>
<td><strong>Push</strong></td>
<td>Inserts an object at the top of the <code>ConcurrentStack&lt;Of &lt;(T)&gt; &gt;</code>.</td>
</tr>
<tr>
<td><strong>PushRange</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>ToArray</strong></td>
<td>Copies the items stored in the <code>ConcurrentStack&lt;Of &lt;(T)&gt; &gt;</code> to a new array.</td>
</tr>
<tr>
<td><strong>ToString</strong></td>
<td>(Inherited from <code>Object</code>).</td>
</tr>
<tr>
<td><strong>TryPeek</strong></td>
<td>Attempts to return an object from the top of the <code>ConcurrentStack&lt;Of &lt;(T)&gt; &gt;</code> without removing it.</td>
</tr>
<tr>
<td><strong>TryPop</strong></td>
<td>Attempts to pop and return the object at the top of the <code>ConcurrentStack&lt;Of &lt;(T)&gt; &gt;</code>.</td>
</tr>
<tr>
<td><strong>TryPopRange</strong></td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ICollection:::CopyTo</code></td>
<td>Copies the elements of the ICollection to an Array, starting at a particular Array index.</td>
</tr>
<tr>
<td><code>IEnumerable:::GetEnumerator</code></td>
<td>Returns an enumerator that iterates through a collection.</td>
</tr>
<tr>
<td><code>IProducerConsumerCollection&lt;OF (T)&gt;:::TryAdd</code></td>
<td>Attempts to add an object to the IProducerConsumerCollection&lt;OF (T)&gt;).</td>
</tr>
<tr>
<td><code>IProducerConsumerCollection&lt;OF (T)&gt;:::TryTake</code></td>
<td>Attempts to remove and return an object from the IProducerConsumerCollection&lt;OF (T)&gt;).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack(Of <T>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Copies the elements of the ICollection to an Array, starting at a particular Array index.

**Namespace:**  [System.Collections.Concurrent](#)  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Sub CopyTo ( _
    array As Array, _
    index As Integer _
) Implements ICollection.CopyTo

C#

void ICollection.CopyTo(
    Array array,
    int index
)

Parameters

array
    Type: System..::.Array
    The one-dimensional Array that is the destination of the elements copied from the ConcurrentStack(Of(Of<T>)). The Array must have zero-based indexing.

index
    Type: System..::.Int32
    The zero-based index in array at which copying begins.

Implements

ICollection..::.CopyTo(Array, Int32)
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>array is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::ArgumentOutOfRangeException</code></td>
<td>index is less than zero.</td>
</tr>
<tr>
<td><code>System:::ArgumentException</code></td>
<td>array is multidimensional. -or- array does not have zero-based indexing. -or- index is equal to or greater than the length of the array -or- The number of elements in the source ICollection is greater than the available space from index to the end of the destination array. -or- The type of the source ICollection cannot be cast automatically to the type of the destination array.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Returns an enumerator that iterates through a collection.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function GetEnumerator As IEnumerator
       Implements IEnumerable.GetEnumerator

C#

IEnumerator IEnumerable.GetEnumerator()

Return Value

An IEnumerator that can be used to iterate through the collection.

Implements

IEnumerable..::.GetEnumerator()()
Remarks

The enumeration represents a moment-in-time snapshot of the contents of the stack. It does not reflect any updates to the collection after `GetEnumerator()` was called. The enumerator is safe to use concurrently with reads from and writes to the stack.
See Also

ConcurrentStack(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Removes all objects from the `ConcurrentStack<(Of <(T)>).`.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Clear

C#

public void Clear()
See Also

ConcurrentStack<(Of (T)>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
C# .NET Framework Class Library
ConcurrentStack(Of <T>),::.CopyTo Method
ConcurrentStack(Of <T>), Class

Copies the ConcurrentStack(Of <T>) elements to an existing one-dimensional Array, starting at the specified array index.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Sub CopyTo ( _
    array As T(), _
    index As Integer _
)
```

**C#**

```csharp
public void CopyTo(
    T[] array,
    int index
)
```

**Parameters**

**array**

Type: `array<T>[]()`

The one-dimensional Array that is the destination of the elements copied from the `ConcurrentStack(Of <T>)`. The Array must have zero-based indexing.

**index**

Type: `System:::Int32`

The zero-based index in array at which copying begins.

**Implements**

`IProducerConsumerCollection(Of <T>)::: CopyTo(array<T>[][], Int32)`
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>array is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>index is less than zero.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>index is equal to or greater than the length of the array -or- The number of elements in the source ( \text{ConcurrentStack&lt;Of (Of \langle T\rangle)&gt;} ) is greater than the available space from index to the end of the destination array.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Returns an enumerator that iterates through the `ConcurrentStack<(Of <T>)>`. 

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function GetEnumerator As IEnumerator(Of T)

C#

public IEnumerable<T> GetEnumerator()

Return Value

An enumerator for the `ConcurrentStack(Of (Of (T)>).`
Remarks

The enumeration represents a moment-in-time snapshot of the contents of the stack. It does not reflect any updates to the collection after GetEnumerator()() was called. The enumerator is safe to use concurrently with reads from and writes to the stack.
See Also

ConcurrentStack(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to add an object to the IProducerConsumerCollection<Of <(T)>).

**Namespace:** System.Collections.Concurrent
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Private Function TryAdd ( _
    item As T _
) As Boolean Implements IProducerConsumerCollection(Of T).TryAdd

**C#**

bool IProducerConsumerCollection&lt;T&gt;.TryAdd(
    T item
)

**Parameters**

item

  Type: T
  The object to add to the IProducerConsumerCollection&lt;(Of &lt;(T)&gt;)&gt;. The value can be a null reference (Nothing in Visual Basic) for reference types.

**Return Value**

true if the object was added successfully; otherwise, false.

**Implements**

IProducerConsumerCollection&lt;(Of &lt;(T)&gt;)&gt;.TryAdd(T)
Remarks

For ConcurrentStack(Of (T)), this operation will always insert the object onto the top of the ConcurrentStack(Of (T)) and return true.
See Also

ConcurrentStack(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to remove and return an object from the IProducerConsumerCollection<(Of <(T)>).

**Namespace:**  System.Collections.Concurrent
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function TryTake ( _
    <OutAttribute> ByRef item As T _
) As Boolean Implements IProducerConsumerCollection(Of T).TryTake

C#

bool IProducerConsumerCollection<T>.TryTake(
    out T item
)

Parameters

item
    Type: T %
    When this method returns, if the operation was successful, item contains the object removed. If no object was available to be removed, the value is unspecified.

Return Value

true if an element was removed and returned succesfully; otherwise, false.

Implements

IProducerConsumerCollection(Of (T)).TryTake(T%)
Remarks

For ConcurrentStack<(Of <(T)>), this operation will attempt to pope the object at the top of the ConcurrentStack<(Of <(T)>).
See Also

ConcurrentStack(Of (T)>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentStack\<(Of <(T\()\rangle)\>.::.Push Method

**ConcurrentStack\<(Of <(T\()\rangle)\>.::.Push Method**

Inserts an object at the top of the **ConcurrentStack\<(Of <(T\()\rangle)\>**.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Push ( _
    item As T _
)

C#

public void Push(
    T item
)

Parameters

item

Type: T
The object to push onto the ConcurrentStack(Of T). The value can be a null reference (Nothing in Visual Basic) for reference types.
See Also

ConcurrentStack(Of (T)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentStack(Of `<T>`):::PushRange Method

**ConcurrentStack(Of `<T>`) Class**  
**See Also**  
**Send Feedback**
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PushRange(array&lt;T&gt;[][])</td>
<td>Inserts multiple objects at the top of the ConcurrentStack&amp;lt(Of &lt;(T)&gt;&gt;) atomically.</td>
</tr>
<tr>
<td>PushRange(array&lt;T&gt;[][], Int32, Int32)</td>
<td>Inserts multiple objects at the top of the ConcurrentStack&amp;lt(Of &lt;(T)&gt;&gt;) atomically.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack(Of(T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Inserts multiple objects at the top of the `ConcurrentStack<(Of <(T)>)>` atomically.

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub PushRange ( _
    items As T() _
)

C#

public void PushRange(
    T[] items
)

Parameters

items
    Type: array<T>[]()
    The objects to push onto the ConcurrentStack(Of (T)).
Remarks

When adding multiple items to the stack, using PushRange is a more efficient mechanism than using `Push(T)` one item at a time. Additionally, PushRange guarantees that all of the elements will be added atomically, meaning that no other threads will be able to inject elements between the elements being pushed. Items at lower indices in the items array will be pushed before items at higher indices.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>items is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack(Of (T)) Class
PushRange Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentStack(Of (T)) PushRange Method (array<T>[][], Int32, Int32)

Inserts multiple objects at the top of the ConcurrentStack(Of (T)) atomically.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub PushRange (_
    items As T(), _
    startIndex As Integer, _
    count As Integer _
)

C#

public void PushRange(
    T[] items,
    int startIndex,
    int count
)

Parameters

items
Type: array< T >[]
The objects to push onto the ConcurrentStack(Of <T>).

startIndex
Type: System..::.Int32
The zero-based offset in items at which to begin inserting elements onto the top of the ConcurrentStack(Of <T>).

count
Type: System..::.Int32
The number of elements to be inserted onto the top of the ConcurrentStack(Of <T>).
Remarks

When adding multiple items to the stack, using PushRange is a more efficient mechanism than using `Push(T)` one item at a time. Additionally, PushRange guarantees that all of the elements will be added atomically, meaning that no other threads will be able to inject elements between the elements being pushed. Items at lower indices in the items array will be pushed before items at higher indices.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>items is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>startIndex or count is negative. Or startIndex is greater than or equal to the length of items.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>startIndex + count is greater than the length of items.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack<(Of <(T)>)> Class
PushRange Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentStack<(Of <(T)>).::ToArray Method

Copies the items stored in the ConcurrentStack<(Of <(T)>)> to a new array.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ToArray As T()

C#

public T[] ToArray()

Return Value

A new array containing a snapshot of elements copied from the ConcurrentStack(Of (Of T>)).
See Also

ConcurrentStack<Of <(T)>>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to return an object from the top of the `ConcurrentStack<Of <T>>` without removing it.

**Namespace:**  System.Collections.Concurrent

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryPeek ( _
    <OutAttribute> ByRef result As T _
) As Boolean

C#

public bool TryPeek(
    out T result
)

Parameters

result
    Type: T
    When this method returns, result contains an object from the top of the
    ConcurrentStack(Of (T)) or an unspecified value if the operation
    failed.

Return Value

true if and object was returned successfully; otherwise, false.
See Also

ConcurrentStack(Of (Of T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to pop and return the object at the top of the ConcurrentStack(Of <T>).

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryPop ( _
    <OutAttribute> ByRef result As T _
) As Boolean

C#

public bool TryPop(
    out T result
)

Parameters

result
Type: T %
When this method returns, if the operation was successful, result contains the object removed. If no object was available to be removed, the value is unspecified.

Return Value

true if an element was removed and returned from the top of the ConcurrentStack(Of (T)>) successfully; otherwise, false.
See Also

ConcurrentStack(Of T) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
ConcurrentStack(Of <(T)>).::.TryPopRange Method
ConcurrentStack(Of <(T)>).::.Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="array%3CT%3E%5BJ%5D%5B%5D" alt="TryPopRange" /></td>
<td>Attempts to pop and return multiple objects from the top of the <code>ConcurrentStack&lt;Of &lt;(T)&gt;&gt;</code> atomically.</td>
</tr>
<tr>
<td>![TryPopRange](array&lt;T&gt;[J][], Int32, Int32)</td>
<td>Attempts to pop and return multiple objects from the top of the <code>ConcurrentStack&lt;Of &lt;(T)&gt;&gt;</code> atomically.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack(Of <T>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to pop and return multiple objects from the top of the `ConcurrentStack(Of T)` atomically.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryPopRange ( _
          items As T() _
    ) As Integer

C#

public int TryPopRange(  
    T[] items
)

Parameters

items
    Type: array< T >[][()]  
The Array to which objects popped from the top of the 
ConcurrentStack<(Of <(T)>)> will be added.

Return Value

The number of objects successfully popped from the top of the 
ConcurrentStack<(Of <(T)>)> and inserted in items.
Remarks

When popping multiple items, if there is little contention on the stack, using TryPopRange can be more efficient than using `TryPop(T%)` once per item to be removed. Nodes fill the items with the first node to be popped at the startIndex, the second node to be popped at startIndex + 1, and so on.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>items is a null argument (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack<Of<(T)>) Class
TryPopRange Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Attempts to pop and return multiple objects from the top of the `ConcurrentStack<Of <T>>` atomically.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryPopRange ( _
    items As T(), _
    startIndex As Integer, _
    count As Integer _
) As Integer

C#

public int TryPopRange(
    T[] items,
    int startIndex,
    int count
)

Parameters

items
Type: array<T>[]()[
The Array to which objects popped from the top of the ConcurrentStack(Of <(T)> ) will be added.

startIndex
Type: System:::Int32
The zero-based offset in items at which to begin inserting elements from the top of the ConcurrentStack(Of <(T)> ).

count
Type: System:::Int32
The number of elements to be popped from top of the ConcurrentStack(Of <(T)> ) and inserted into items.
Remarks

When popping multiple items, if there is little contention on the stack, using TryPopRange can be more efficient than using TryPop(T%) once per item to be removed. Nodes fill the items with the first node to be popped at the startIndex, the second node to be popped at startIndex + 1, and so on.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>items is a null reference (Nothing in Visual Basic). startIndex or count is negative. Or startIndex is greater than or equal to the length of items.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>startIndex + count is greater than the length of items.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack(Of T) Class
TryPopRange Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The ConcurrentStack<T> type exposes the following members.
# Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Count  | Gets the number of elements contained in the **ConcurrentStack**<**(T)**>.
| isEmpty | Gets a value that indicates whether the **ConcurrentStack**<**(T)**> is empty. |
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICollection::IsSynchronized</td>
<td>Gets a value indicating whether access to the ICollection is synchronized with the SyncRoot.</td>
</tr>
<tr>
<td>ICollection::SyncRoot</td>
<td>Gets an object that can be used to synchronize access to the ICollection. This property is not supported.</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack(Of (Of (T)>)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value indicating whether access to the ICollection is synchronized with the SyncRoot.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Private ReadOnly Property IsSynchronized As Boolean
Implements ICollection(IsSynchronized

**C#**

bool ICollection.IsSynchronized { get; }

**Field Value**

true if access to the ICollection is synchronized with the SyncRoot; otherwise, false. For ConcurrentStack(Of &lt;T&gt;), this property always returns false.

**Implements**

ICollection...:::IsSynchronized
See Also

ConcurrentStack(Of (T)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
ConcurrentStack(Of Of T)>..::.ICollection..::.SyncRoot Property
ConcurrentStack(Of Of T)> Class  See Also  Send Feedback

Gets an object that can be used to synchronize access to the ICollection. This property is not supported.

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private ReadOnly Property SyncRoot As Object
Implements ICollection.SyncRoot

C#

Object ICollection.SyncRoot { get; }

Implements

ICollection..::.SyncRoot
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.::.NotSupportedException</td>
<td>The SyncRoot property is not supported</td>
</tr>
</tbody>
</table>
See Also

ConcurrentStack<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets the number of elements contained in the `ConcurrentStack(Of T)`.  

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public ReadOnly Property Count As Integer

**C#**

public int Count { get; }

**Field Value**

The number of elements contained in the ConcurrentStack`< Of `<T>`>`.

**Implements**

IICollection...:::Count
Remarks

For determining whether the collection contains any items, use of the `IsEmpty` property is recommended rather than retrieving the number of items from the Count property and comparing it to 0.
See Also

ConcurrentStack<T> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets a value that indicates whether the `ConcurrentStack(Of T)` is empty.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsEmpty As Boolean

C#

public bool IsEmpty { get; }

Field Value

ture if the `ConcurrentStack(Of Of T)Type` is empty; otherwise, false.
Remarks

For determining whether the collection contains any items, use of this property is recommended rather than retrieving the number of items from the Count property and comparing it to 0. However, as this collection is intended to be accessed concurrently, it may be the case that another thread will modify the collection after IsEmpty returns, thus invalidating the result.
See Also

ConcurrentStack<(Of <(T)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Defines methods to manipulate thread-safe collections intended for producer/consumer usage.

**Namespace:**  System.Collections.Concurrent

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Interface IProducerConsumerCollection(Of T) _
    Implements IEnumerable(Of T), ICollection, IEnumerable

C#

class IProducerConsumerCollection<T> : IEnumerable<T>, 
    ICollection, IEnumerable
Type Parameters

T
Specifies the type of elements in the collection.
Remarks

All implementations of this interface must enable all members of this interface to be used concurrently from multiple threads.
See Also

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `IProducerConsumerCollection(Of T)` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyTo</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>GetEnumerator</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ToArray</td>
<td>Copies the elements contained in the IProducerConsumerCollection&lt;T&gt; to a new array.</td>
</tr>
<tr>
<td>TryAdd</td>
<td>Attempts to add an object to the IProducerConsumerCollection&lt;T&gt;.</td>
</tr>
<tr>
<td>TryTake</td>
<td>Attempts to remove and return an object from the IProducerConsumerCollection&lt;T&gt;.</td>
</tr>
</tbody>
</table>
See Also

**IProducerConsumerCollection(Of `(Of `<T`>`)) Interface**

**System.Collections.Concurrent Namespace**

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
IProducerConsumerCollection(Of T).:::CopyTo Method
IProducerConsumerCollection(Of T) Interface  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyTo(Array, Int32)</td>
<td>(Inherited from ICollection.) Copies the elements of the IProducerConsumerCollection&lt;Of &lt;T&gt;&gt; to an Array, starting at a specified index.</td>
</tr>
<tr>
<td>CopyTo(array&lt;T&gt;[], Int32)</td>
<td></td>
</tr>
</tbody>
</table>
See Also

IProducerConsumerCollection(Of `<T>`>) Interface
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
C# .NET Framework Class Library
IProducerConsumerCollection<(Of <(T)>)>...:::CopyTo Method (array<T>[], Int32)
IProducerConsumerCollection<(Of <(T)>)> Interface  See Also  Send Feedback

Copies the elements of the IProducerConsumerCollection<(Of <(T)>)> to an Array, starting at a specified index.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Sub CopyTo ( _
    array As T(), _
    index As Integer _
)

C#

void CopyTo(
    T[] array,
    int index
)

Parameters

array
    Type: array< T >[]()
    The one-dimensional Array that is the destination of the elements copied from the IProducerConsumerCollection<Of <(T)>>. The array must have zero-based indexing.

index
    Type: System..::.Int32
    The zero-based index in array at which copying begins.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>array is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>index is less than zero.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>index is equal to or greater than the length of the array -or- The number of elements in the source <code>ConcurrentQueue(Of &lt;(T)&gt;)</code> is greater than the available space from index to the end of the destination array.</td>
</tr>
</tbody>
</table>
See Also

IProducerConsumerCollection<(Of <(T)>)> Interface
CopyTo Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
IProducerConsumerCollection<(Of <(T)>)).GetEnumerator Method
IProducerConsumerCollection<(Of <(T)>)) Interface
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetEnumerator()() (Inherited from IEnumerable&lt;Of &lt;(T)&gt;).)</td>
<td></td>
</tr>
<tr>
<td>GetEnumerator()() (Inherited from IEnumerable.)</td>
<td></td>
</tr>
</tbody>
</table>
See Also

`IProducerConsumerCollection(Of `T`)` Interface
`System.Collections.Concurrent Namespace`

Send feedback on this topic to Microsoft.
Copies the elements contained in the `IProducerConsumerCollection(Of `(Of `T`>)`)` to a new array.

**Namespace:**  `System.Collections.Concurrent`  
**Assembly:**  `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Function ToArray As T()

C#

T[] ToArray()

Return Value

A new array containing the elements copied from the IProducerConsumerCollection(Of (Of T)>).
See Also

**IProducerConsumerCollection(Of (T))** Interface
**System.Collections.Concurrent Namespace**

Send feedback on this topic to Microsoft.
Attempts to add an object to the `IProducerConsumerCollection(Of T)`.  

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Function TryAdd ( _
        item As T _
    ) As Boolean

C#

bool TryAdd(
            T item
        )

Parameters

item
    Type: T
    The object to add to the IProducerConsumerCollection(Of (T)).

Return Value

true if the object was added successfully; otherwise, false.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentException</td>
<td>The item was invalid for this collection.</td>
</tr>
</tbody>
</table>
See Also

**IProducerConsumerCollection(Of T) Interface**
System.Collections.Concurrent Namespace

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Attempts to remove and return an object from the `IProducerConsumerCollection(Of <T>)`.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Function TryTake (_
    <OutAttribute> ByRef item As T _
) As Boolean
```

**C#**

```csharp
bool TryTake(
    out T item
)
```

**Parameters**

- **item**
  
  Type: `T`
  
  When this method returns, if the object was removed and returned successfully, `item` contains the removed object. If no object was available to be removed, the value is unspecified.

**Return Value**

true if an object was removed and returned successfully; otherwise, false.
See Also

`IProducerConsumerCollection(Of T)` Interface
`System.Collections.Concurrent Namespace`

Send feedback on this topic to Microsoft.
The `IProducerConsumerCollection(Of `T`)` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>(Inherited from ICollection.)</td>
</tr>
<tr>
<td>IsSynchronized</td>
<td>(Inherited from ICollection.)</td>
</tr>
<tr>
<td>SyncRoot</td>
<td>(Inherited from ICollection.)</td>
</tr>
</tbody>
</table>
See Also

IProducerConsumerCollection(Of T) Interface
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
OrderablePartitioner(Of TSource) Class

Represents a particular manner of splitting an orderable data source into multiple partitions.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading := True)> _
Public MustInherit Class OrderablePartitioner(Of TSource) _
   Inherits Partitioner(Of TSource)

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading = true)]
public abstract class OrderablePartitioner<TSource> : Partitioner<TSource>
Type Parameters

TSource

Type of the elements in the collection.
Remarks

Each element in each partition has an integer index associated with it, which determines the relative order of that element against elements in other partitions.

Inheritors of OrderablePartitioner<Of (TSource)> must adhere to the following rules:

1. All indices must be unique, such that there may not be duplicate indices. If all indices are not unique, the output ordering may be scrambled.
2. All indices must be non-negative. If any indices are negative, consumers of the implementation may throw exceptions.
3. GetPartitions(Int32) and GetOrderablePartitions(Int32) should throw a ArgumentOutOfRangeException if the requested partition count is less than or equal to zero.
4. GetPartitions(Int32) and GetOrderablePartitions(Int32) should always return a number of enumerables equal to the requested partition count. If the partitioner runs out of data and cannot create as many partitions as requested, an empty enumerator should be returned for each of the remaining partitions. If this rule is not followed, consumers of the implementation may throw a InvalidOperationException.
5. GetPartitions(Int32), GetOrderablePartitions(Int32), GetDynamicPartitions(), and GetOrderableDynamicPartitions() should never return null. If null is returned, a consumer of the implementation may throw a InvalidOperationException.
6. GetPartitions(Int32), GetOrderablePartitions(Int32), GetDynamicPartitions(), and GetOrderableDynamicPartitions() should always return partitions that can fully and uniquely enumerate the input data source. All of the data and only the data contained in the input source should be enumerated, with no duplication that was not already in the input, unless specifically required by the particular partitioner's design. If this is not followed, the output ordering may be scrambled.
7. If KeysOrderedInEachPartition returns true, each partition must return elements with increasing key indices.
8. If KeysOrderedAcrossPartitions returns true, all the keys in partition numbered N must be larger than all the keys in partition numbered N-1.
9. If $\text{KeysNormalized}$ returns true, all indices must be monotonically increasing from 0, though not necessarily within a single partition.
Inheritance Hierarchy

System:::Object
System.Collections.Concurrent:::Partitioner<Of <(TSource)>)
System.Collections.Concurrent:::OrderablePartitioner<Of <(TSource)>)}
See Also

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `OrderablePartitioner<Of <(TSource)>>)` class with the specified constraints on the index keys.

**Namespace:**  `System.Collections.Concurrent`

**Assembly:**  `System.Threading` (in `System.Threading.dll`)
## Syntax

### Visual Basic (Declaration)

```vbnet
Protected Sub New (_
    keysOrderedInEachPartition As Boolean, _
    keysOrderedAcrossPartitions As Boolean, _
    keysNormalized As Boolean _
)
```

### C#

```csharp
protected OrderablePartitioner(
    bool keysOrderedInEachPartition,
    bool keysOrderedAcrossPartitions,
    bool keysNormalized
)
```

## Parameters

**keysOrderedInEachPartition**
- **Type:** System..::.Boolean
- Indicates whether the elements in each partition are yielded in the order of increasing keys.

**keysOrderedAcrossPartitions**
- **Type:** System..::.Boolean
- Indicates whether elements in an earlier partition always come before elements in a later partition. If true, each element in partition 0 has a smaller order key than any element in partition 1, each element in partition 1 has a smaller order key than any element in partition 2, and so on.

**keysNormalized**
- **Type:** System..::.Boolean
- Indicates whether keys are normalized. If true, all order keys are distinct integers in the range [0 .. numberOfElements-1]. If false, order keys must still be distinct, but only their relative order is considered, not their absolute values.
See Also

OrderablePartitioner<(Of <(TSource)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `OrderablePartitioner<T>` type exposes the following members.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetDynamicPartitions</td>
<td>Creates an object that can partition the underlying collection into a variable number of partitions. (Overrides <code>Partitioner&lt;TKey, TSource&gt;</code>::GetDynamicPartitions())</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetOrderableDynamicPartitions</td>
<td>Creates an object that can partition the underlying collection into a variable number of partitions.</td>
</tr>
<tr>
<td>GetOrderablePartitions</td>
<td>Partitions the underlying collection into the specified number of orderable partitions.</td>
</tr>
<tr>
<td>GetPartitions</td>
<td>Partitions the underlying collection into the given number of ordered partitions. (Overrides <code>Partitioner&lt;TKey, TSource&gt;</code>::GetPartitions(Int32))</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
See Also

OrderablePartitioner(Of TSource) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
OrderablePartitioner(Of TSource).GetDynamicPartitions Method

OrderablePartitioner(Of TSource) Class

Creates an object that can partition the underlying collection into a variable number of partitions.

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overrides Function GetDynamicPartitions As IEnumerable(Of TSource)

C#

can.

public override IEnumerable<TSource> GetDynamicPartitions()

Return Value

An object that can create partitions over the underlying data source.
Remarks

The returned object implements the `IEnumerable<Of <(TSource)>>` interface. Calling GetEnumerator on the object creates another partition over the sequence.

The default implementation provides the same behavior as `GetOrderableDynamicPartitions()` except that the returned set of partitions does not provide the keys for the elements.

The GetDynamicPartitions() method is only supported if the `SupportsDynamicPartitions` property returns true.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.NotSupportedException</td>
<td>Dynamic partitioning is not supported by this partitioner.</td>
</tr>
</tbody>
</table>
See Also

**OrderablePartitioner(Of (TSource)>)** Class
**System.Collections.Concurrent Namespace**

Send [feedback](mailto:) on this topic to Microsoft.
Create an object that can partition the underlying collection into a variable number of partitions.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overridable Function GetOrderableDynamicPartitions As IEnumerable

C#

public virtual IEnumerable<KeyValuePair<long, TSource>> GetOrderableDynamicPartitions

Return Value

An object that can create partitions over the underlying data source.
Remarks

The returned object implements the IEnumerable<(Of TSource)>) interface. Calling GetEnumerator on the object creates another partition over the sequence.

Each partition is represented as an enumerator over key-value pairs. The value in the pair is the element itself, and the key is an integer which determines the relative ordering of this element against other elements.

The GetOrderableDynamicPartitions()() method is only supported if the SupportsDynamicPartitions property returns true.
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.NotSupportedException</td>
<td>Dynamic partitioning is not supported by this partitioner.</td>
</tr>
</tbody>
</table>
See Also

OrderablePartitioner<(Of <(TSource)>)> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Partitions the underlying collection into the specified number of orderable partitions.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public MustOverride Function GetOrderablePartitions ( _
    partitionCount As Integer _
) As IList(Of IEnumerable(Of KeyValuePair(Of Long, TSource)))

C#

public abstract IList<IEnumerator<KeyValuePair<long, TSource>>> Get(
    int partitionCount
)

Parameters

partitionCount
  Type: System..::.Int32
  The number of partitions to create.

Return Value

A list containing partitionCount enumerators.
Remarks

Each partition is represented as an enumerator over key-value pairs. The value of the pair is the element itself, and the key is an integer which determines the relative ordering of this element against other elements in the data source.
See Also

**OrderablePartitioner(Of TSource) Class**
*System.Collections.Concurrent Namespace*

Send feedback on this topic to Microsoft.
Partitions the underlying collection into the given number of ordered partitions.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overrides Function GetPartitions ( _
    partitionCount As Integer _
) As IList(Of IEnumerator(Of TSource))

C#

public override IList<IEnumerator<TSource>> GetPartitions(
    int partitionCount
)

Parameters

partitionCount
    Type: System::Int32
    The number of partitions to create.

Return Value

A list containing partitionCount enumerators.
Remarks

The default implementation provides the same behavior as `GetOrderablePartitions(Int32)` except that the returned set of partitions does not provide the keys for the elements.
See Also

`OrderablePartitioner<(Of <(TSource)>)>` Class
`System.Collections.Concurrent Namespace`

Send feedback on this topic to Microsoft.
The `OrderablePartitioner(Of TSource)<>` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeysNormalized</td>
<td>Gets whether order keys are normalized. Gets whether elements in an earlier partition always come before elements in a later partition.</td>
</tr>
<tr>
<td>KeysOrderedAcrossPartitions</td>
<td>Gets whether elements in each partition are yielded in the order of increasing keys.</td>
</tr>
<tr>
<td>KeysOrderedInEachPartition</td>
<td>Gets whether additional partitions can be created dynamically. (Inherited from Partitioner(Of&lt;TSource&gt;)).</td>
</tr>
<tr>
<td>SupportsDynamicPartitions</td>
<td></td>
</tr>
</tbody>
</table>
See Also

OrderIdabler\langle\langle TSource\rangle\rangle Class
System.Collections.Concurrent Namespace
Send feedback on this topic to Microsoft.
Gets whether order keys are normalized.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Property KeysNormalized As Boolean

C#

public bool KeysNormalized { get; private set; }
Remarks

If KeysNormalized returns true, all order keys are distinct integers in the range [0 .. numberOfElements-1]. If the property returns false, order keys must still be distinct, but only their relative order is considered, not their absolute values.
See Also

OrderByPartitioner(Of TSource) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets whether elements in an earlier partition always come before elements in a later partition.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Property KeysOrderedAcrossPartitions As Boolean

C#

public bool KeysOrderedAcrossPartitions { get; private set; }
Remarks

If KeysOrderedAcrossPartitions returns true, each element in partition 0 has a smaller order key than any element in partition 1, each element in partition 1 has a smaller order key than any element in partition 2, and so on.
See Also

OrderablePartitioner<(Of <(TSource)>)) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Gets whether elements in each partition are yielded in the order of increasing keys.

**Namespace:**  System.Collections.Concurrent  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Property KeysOrderedInEachPartition As Boolean

C#

public bool KeysOrderedInEachPartition { get; private set; }
See Also

OrderablePartitioner(Of TSource) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Provides common partitioning strategies for arrays, lists, and enumerables.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading := True)> _
Public NotInheritable Class Partitioner

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading = true)]
public static class Partitioner
Remarks

The static methods on Partitioner are all thread-safe and may be used concurrently from multiple threads. However, while a created partitioner is in use, the underlying data source should not be modified, whether from the same thread that's using a partitioner or from a separate thread.
Inheritance Hierarchy

System...:::Object
System.Collections.Concurrent...:::Partitioner
See Also

[System.Collections.Concurrent Namespace]

Send [feedback] on this topic to Microsoft.
Partitioner Methods

Partitioner Class  See Also  Send Feedback
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

Partitioner Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
Partitioner::Create Method
Partitioner Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create(Of TSource&gt;) (IEnumerable(Of TSource)&gt;))</td>
<td>Creates an orderable partitioner from a IEnumerable&lt;T&gt; instance.</td>
</tr>
<tr>
<td>Create(Of TSource&gt;)(IList(Of TSource&gt;), Boolean)</td>
<td>Creates an orderable partitioner from an IList&lt;T&gt; instance.</td>
</tr>
<tr>
<td>Create(Of TSource&gt;) (array&lt;TSource&gt;[][], Boolean)</td>
<td>Creates an orderable partitioner from a Array instance.</td>
</tr>
</tbody>
</table>
See Also

Partitioner Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Partitioner...Create(Of (TSource)) Method (IEnumerable(Of (TSource)))

Partitioner Class  See Also  Send Feedback

Creates an orderable partitioner from a IEnumerable(Of T) instance.

Namespace:  System.Collections.Concurrent
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Create(Of TSource) ( _
    source As IEnumerable(Of TSource) _
) As OrderablePartitioner(Of TSource)

C#

public static OrderablePartitioner<TSource> Create<TSource>(
    IEnumerable<TSource> source
)

Parameters

source
    Type: System.Collections.Generic.IEnumerable(Of TSource)
The enumerable to be partitioned.
### Type Parameters

**TSource**
Type of the elements in source enumerable.

### Return Value

An orderable partitioner based on the input array.
Remarks

The ordering used in the created partitioner is determined by the natural order of the elements as retrieved from the source enumerable.
See Also

Partitioner Class
Create Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Creates an orderable partitioner from an IList<T> instance.

**Namespace:** System.Collections.Concurrent  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Create(Of TSource) ( _
    list As IList(Of TSource), _
    loadBalance As Boolean _
) As OrderablePartitioner(Of TSource)

C#

public static OrderablePartitioner<TSource> Create<TSource>(
    IList<TSource> list,
    bool loadBalance
)

Parameters

list
    Type: System.Collections.Generic..::.IList<(Of <(TSource)>)>)
    The list to be partitioned.

loadBalance
    Type: System..::.Boolean
    A Boolean value that indicates whether the created partitioner should
dynamically load balance between partitions rather than statically partition.
**Type Parameters**

**TSource**
Type of the elements in source list.

**Return Value**
An orderable partitioner based on the input list.
See Also

Partitioner Class
Create Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Partitions to orderable partitioner from a Array instance.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Create(Of TSource) ( _
    array As TSource(), _
    loadBalance As Boolean _
) As OrderablePartitioner(Of TSource)

C#

public static OrderablePartitioner<TSource> Create<TSource>(
    TSource[] array,
    bool loadBalance
)

Parameters

array
    Type: array< TSource >[]
    The array to be partitioned.

loadBalance
    Type: System:::Boolean
    A Boolean value that indicates whether the created partitioner should dynamically load balance between partitions rather than statically partition.
Type Parameters

TSource
Type of the elements in source array.

Return Value
An orderable partitioner based on the input array.
See Also

Partitioner Class
Create Overload
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Represents a particular manner of splitting a data source into multiple partitions.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
  ExternalThreading := True)> _
Public MustInherit Class Partitioner(Of TSource)

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
  ExternalThreading = true)]
public abstract class Partitioner<TSource>
Type Parameters

**TSource**

Type of the elements in the collection.
Remarks

Inheritors of Partitioner<(Of <(TSource>>)>) must adhere to the following rules:

1. `GetPartitions(Int32)` should throw a ArgumentOutOfRangeException if the requested partition count is less than or equal to zero.
2. `GetPartitions(Int32)` should always return a number of enumerables equal to the requested partition count. If the partitioner runs out of data and cannot create as many partitions as requested, an empty enumerator should be returned for each of the remaining partitions. If this rule is not followed, consumers of the implementation may throw a InvalidOperationException.
3. `GetPartitions(Int32)` and `GetDynamicPartitions()` should never return null. If null is returned, a consumer of the implementation may throw a InvalidOperationException.
4. `GetPartitions(Int32)` and `GetDynamicPartitions()` should always return partitions that can fully and uniquely enumerate the input data source. All of the data and only the data contained in the input source should be enumerated, with no duplication that was not already in the input, unless specifically required by the particular partitioner's design. If this is not followed, the output ordering may be scrambled.
Inheritance Hierarchy

System...:::Object
System.Collections.Concurrent...:::Partitioner(Of (TSource))>
  System.Collections.Concurrent...:::OrderablePartitioner(Of (TSource))>
See Also

System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Partitioner<(Of <(TSource)>)> Constructor

Namespace: System.Collections.Concurrent
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Protected Sub New

C#

protected Partitioner()
See Also

Partitioner(Of (TSource)>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `Partitioner(Of TSource)`) type exposes the following members.
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetDynamicPartitions</strong></td>
<td>Creates an object that can partition the underlying collection into a variable number of partitions.</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetPartitions</strong></td>
<td>Partitions the underlying collection into the given number of partitions.</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
See Also

Partitioner<Of<TSource>> Class  
System.Collections.Concurrent Namespace  

Send feedback on this topic to Microsoft.
Creates an object that can partition the underlying collection into a variable number of partitions.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Overridable Function GetDynamicPartitions As IEnumerable(Of T)
```

**C#**

```csharp
public virtual IEnumerable<TSource> GetDynamicPartitions()
```

**Return Value**

An object that can create partitions over the underlying data source.
Remarks

The returned object implements the IEnumerable<(Of<TSource>>)> interface. Calling GetEnumerator on the object creates another partition over the sequence.

The GetDynamicPartitions() method is only supported if the SupportsDynamicPartitions property returns true.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.NotSupportedException</td>
<td>Dynamic partitioning is not supported by this partitioner.</td>
</tr>
</tbody>
</table>
See Also

Partitioner<Of<TSource>> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Partitions the underlying collection into the given number of partitions.

**Namespace:**  [System.Collections.Concurrent](#)

**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public MustOverride Function GetPartitions ( _
    partitionCount As Integer _
) As IList(Of IEnumerator(Of TSource))
```

**C#**

```csharp
public abstract IList<IEnumerator<TSource>> GetPartitions(
    int partitionCount
)
```

**Parameters**

- **partitionCount**
  - Type: System..::.Int32
  - The number of partitions to create.

**Return Value**

A list containing partitionCount enumerators.
See Also

Partitioner(Of<TSource>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
The `Partitioner<Of (TSource)>` type exposes the following members.
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupportsDynamicPartitions</td>
<td>Gets whether additional partitions can be created dynamically.</td>
</tr>
</tbody>
</table>
See Also

Partitioner<(Of <(TSource>>)>) Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
Partitions Property

Partitioner<(Of <(TSource)>)> Class

See Also
Send Feedback

Gets whether additional partitions can be created dynamically.

**Namespace:** System.Collections.Concurrent

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overridable ReadOnly Property SupportsDynamicPartitions As Boolean

C#

public virtual bool SupportsDynamicPartitions { get; }

Return Value

true if the `<TSource>` can create partitions dynamically as they are requested; false if the `<TSource>` can only allocate partitions statically.
Remarks

If a derived class does not override and implement `GetDynamicPartitions()`, `SupportsDynamicPartitions` should return false. The value of `SupportsDynamicPartitions` should not vary over the lifetime of this instance.
See Also

Partitioner<Of<TSource>> Class
System.Collections.Concurrent Namespace

Send feedback on this topic to Microsoft.
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OrderedParallelQuery&lt;TSource&gt;</code></td>
<td>Represents a sorted, parallel sequence.</td>
</tr>
<tr>
<td><code>ParallelEnumerable</code></td>
<td>Provides a set of methods for querying objects that implement <code>ParallelQuery&lt;TSource&gt;</code>. This is the parallel equivalent of <code>Enumerable</code>.</td>
</tr>
<tr>
<td><code>ParallelQuery</code></td>
<td>Represents a parallel sequence.</td>
</tr>
<tr>
<td><code>ParallelQuery&lt;TSource&gt;</code></td>
<td>Represents a parallel sequence.</td>
</tr>
</tbody>
</table>
## Enumerations

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ParallelExecutionMode</strong></td>
<td>The query execution mode is a hint that specifies how the system should handle performance trade-offs when parallelizing queries.</td>
</tr>
<tr>
<td></td>
<td>Specifies the preferred type of output merge to use in a query. This is a hint only, and may not be respected by the system when parallelizing all queries.</td>
</tr>
</tbody>
</table>

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Represented a sorted, parallel sequence.

**Namespace:**  System.Linq

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Class OrderedParallelQuery(Of TSource) _
Inherits ParallelQuery(Of T)

C#

public class OrderedParallelQuery<TSource> : ParallelQuery<T>
Type Parameters

TSource
Inheritance Hierarchy

System..::.Object
System.Linq..::.ParallelQuery
 System.Linq..::.ParallelQuery(Of (T))
 System.Linq..::.OrderedParallelQuery(Of (TSource))
See Also

System.Linq Namespace

Send feedback on this topic to Microsoft.
The `OrderedParallelQuery<`Of `<(TSource)>`>` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetEnumerator</td>
<td>Returns an enumerator that iterates through the sequence. (Overrides <code>ParallelQuery&lt;Of &lt;(TSource)&gt;).::GetEnumerator()().</code>)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `IEnumerable.GetEnumerator` | Returns an enumerator that iterates through the sequence.  
(Inherited from `ParallelQuery`.) |
See Also

`OrderedParallelQuery<TSource>` Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns an enumerator that iterates through the sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Overrides Function GetEnumerator As IEnumerator(Of TSource)

C#

public override IEnumerator<TSource> GetEnumerator()

Return Value

An enumerator that iterates through the sequence.
See Also

`OrderedParallelQuery<Of <(TSource)>>) Class System.Linq Namespace`

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Provides a set of methods for querying objects that implement ParallelQuery{TSource}. This is the parallel equivalent of Enumerable.

**Namespace:**  [System.Linq](#)  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public NotInheritable Class ParallelEnumerable

C#

public static class ParallelEnumerable
Inheritance Hierarchy

System...:::Object
System.Linq...:::ParallelEnumerable
See Also

System.Linq Namespace

Send feedback on this topic to Microsoft.
The **ParallelEnumerable** type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aggregate</strong></td>
<td>Overloaded. Determines in parallel whether all elements of a sequence satisfy a condition.</td>
</tr>
<tr>
<td><strong>All(Of (TSource)&gt;)</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Any</strong></td>
<td>Converts a <code>ParallelQuery(Of (TSource)&gt;)</code> into an <code>IEnumerable(Of (T)&gt;)</code> to force sequential evaluation of the query.</td>
</tr>
<tr>
<td><strong>AsEnumerable(Of (TSource)&gt;)</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>AsOrdered</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>AsParallel</strong></td>
<td>Converts a <code>ParallelQuery(Of (TSource)&gt;)</code> into an <code>IEnumerable(Of (T)&gt;)</code> to force sequential evaluation of the query.</td>
</tr>
<tr>
<td><strong>AsSequential(Of (TSource)&gt;)</strong></td>
<td>Allows an intermediate query to be treated as if no ordering is implied among the elements.</td>
</tr>
<tr>
<td><strong>AsUnordered(Of (TSource)&gt;)</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Cast(Of (TResult)&gt;)</strong></td>
<td>Converts the elements of a <code>ParallelQuery</code> to the specified type.</td>
</tr>
<tr>
<td><strong>Concat</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Contains</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Count</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>DefaultIfEmpty</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Distinct</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>ElementAt(Of (TSource)&gt;)</strong></td>
<td>Returns the element at a specified index in a parallel sequence.</td>
</tr>
<tr>
<td><strong>ElementAtOrDefault(Of (TSource)&gt;)</strong></td>
<td>Returns the element at a specified index in a parallel sequence or a default value.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>Empty&lt;Of &lt;TResult&gt;&gt;</code></td>
<td>Returns an empty <code>ParallelQuery{TResult}</code> that has the specified type argument.</td>
</tr>
<tr>
<td><code>Except</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>First</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>FirstOrDefault</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>ForAll&lt;Of &lt;(TSource)&gt;&gt;</code></td>
<td>Invokes in parallel the specified action for each element in the source.</td>
</tr>
<tr>
<td><code>GroupBy</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>GroupJoin</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Intersect</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Join</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Last</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>LastOrDefault</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>LongCount</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Max</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Min</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>OfType&lt;Of &lt;(TResult)&gt;&gt;</code></td>
<td>Filters the elements of a <code>ParallelQuery</code> based on a specified type.</td>
</tr>
<tr>
<td><code>OrderBy</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>OrderByDescending</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Range</code></td>
<td>Generates a parallel sequence of integral numbers within a specified range.</td>
</tr>
<tr>
<td><code>Repeat&lt;Of &lt;(TResult)&gt;&gt;</code></td>
<td>Generates a parallel sequence that contains one repeated value.</td>
</tr>
<tr>
<td><code>Reverse&lt;Of &lt;(TSource)&gt;&gt;</code></td>
<td>Inverts the order of the elements in a parallel sequence.</td>
</tr>
<tr>
<td><code>Select</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>SelectMany</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>SequenceEqual</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Single</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>SingleOrDefault</code></td>
<td>Bypasses a specified number of elements.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>Skip&lt;(Of &lt;(TSource)&gt;)&gt;</code></td>
<td>Returns the remaining elements in a parallel sequence and then returns the remaining elements.</td>
</tr>
<tr>
<td><code>SkipWhile</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Sum</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Take&lt;(Of &lt;(TSource)&gt;)&gt;</code></td>
<td>Returns a specified number of contiguous elements from the start of a parallel sequence.</td>
</tr>
<tr>
<td><code>TakeWhile</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>ThenBy</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>ThenByDescending</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>ToArray&lt;(Of &lt;(TSource)&gt;)&gt;</code></td>
<td>Creates an array from a ParallelQuery{T}.</td>
</tr>
<tr>
<td><code>ToDictionary</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>ToList&lt;(Of &lt;(TSource)&gt;)&gt;</code></td>
<td>Creates a List{T} from an ParallelQuery{T}.</td>
</tr>
<tr>
<td><code>ToLookup</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Union</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Where</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>WithCancellation&lt;(Of &lt;(TSource)&gt;)&gt;</code></td>
<td>Sets the CancellationToken to associate with the query.</td>
</tr>
<tr>
<td><code>WithDegreeOfParallelism&lt;(Of &lt;(TSource)&gt;)&gt;</code></td>
<td>Sets the degree of parallelism to use in a query. Degree of parallelism is the maximum number of concurrently executing tasks that will be used to process the query.</td>
</tr>
<tr>
<td><code>WithExecutionMode&lt;(Of &lt;(TSource)&gt;)&gt;</code></td>
<td>Sets the execution mode of the query.</td>
</tr>
<tr>
<td><code>WithMergeOptions&lt;(Of &lt;(TSource)&gt;)&gt;</code></td>
<td>Sets the merge options for this query, which specify how the query will buffer output.</td>
</tr>
<tr>
<td><code>Zip</code></td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Aggregate Method

ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate&lt;(Of &lt;(TSource)&gt;)) (ParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TSource, TSource, TSource)&gt;))</td>
<td>Applies in parallel an accumulator function over a sequence.</td>
</tr>
<tr>
<td>Aggregate&lt;(Of &lt;(TSource, TAccumulate)&gt;)(ParallelQuery&lt;(Of &lt;(TSource)&gt;), TAccumulate, Func&lt;(Of &lt;(TAccumulate, TSource, TAccumulate)&gt;)))</td>
<td>Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.</td>
</tr>
<tr>
<td>Aggregate&lt;(Of &lt;(TSource, TAccumulate, TResult)&gt;)(ParallelQuery&lt;(Of &lt;(TSource)&gt;), TAccumulate, Func&lt;(Of &lt;(TAccumulate, TSource, TAccumulate)&gt;), Func&lt;(Of &lt;(TAccumulate, TAccumulate, TAccumulate)&gt;), Func&lt;(Of &lt;(TAccumulate, TResult)&gt;)))</td>
<td>Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.</td>
</tr>
<tr>
<td>Aggregate&lt;(Of &lt;(TSource, TAccumulate, TResult)&gt;)(ParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TAccumulate)&gt;), Func&lt;(Of &lt;(TAccumulate, TSource, TAccumulate)&gt;), Func&lt;(Of &lt;(TAccumulate, TAccumulate)&gt;), Func&lt;(Of &lt;(TAccumulate, TResult)&gt;)))</td>
<td>Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.</td>
</tr>
<tr>
<td>Aggregate&lt;(Of &lt;(TSource, TAccumulate, TResult)&gt;)(ParallelQuery&lt;(Of &lt;(TSource)&gt;), TAccumulate, Func&lt;(Of &lt;(TAccumulate, TSource, TAccumulate)&gt;), Func&lt;(Of &lt;(TAccumulate, TAccumulate)&gt;), Func&lt;(Of &lt;(TAccumulate, TResult)&gt;)))</td>
<td>Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.</td>
</tr>
</tbody>
</table>

The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
TAccumulate>, Func(Of (TAccumulate, TResult))

sequential implementation.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.Aggregate(Of (TSource)>)) Method
(ParallelQuery(Of (TSource)>), Func(Of (TSource, TSource, TSource)>))

Applies in parallel an accumulator function over a sequence.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Aggregate(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    func As Func(Of TSource, TSource, TSource) _
) As TSource

C#

public static TSource Aggregate<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, TSource, TSource> func
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource)>)
    A sequence to aggregate over.

func
    Type: System.Func(Of (Of TSource, TSource, TSource)>)
    An accumulator function to be invoked on each element.
Type Parameters

TSource
   The type of the elements of source.

Return Value

The final accumulator value.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or func is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Aggregate Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Aggregate(Of TSource, TAccumulate) ( _
   source As ParallelQuery(Of TSource), _
   seed As TAccumulate, _
   func As Func(Of TAccumulate, TSource, TAccumulate) _
) As TAccumulate

C#

class

public static TAccumulate Aggregate<TSource, TAccumulate>(
   ParallelQuery<TSource> source,
   TAccumulate seed,
   Func<TAccumulate, TSource, TAccumulate> func
)

Parameters

source
   Type: System.Linq.ParallelQuery(Of TSource>)
   A sequence to aggregate over.

seed
   Type: TAccumulate
   The initial accumulator value.

func
   Type: System.Func(Of TAccumulate, TSource, TAccumulate)>
   An accumulator function to be invoked on each element.
### Type Parameters

TSource
The type of the elements of source.

TAccumulate
The type of the accumulator value.

### Return Value

The final accumulator value.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source or func is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System..::.AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System..::.OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Aggregate Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Aggregate(Of TSource, TAccumulate, TResult) (  
    source As ParallelQuery(Of TSource),  
    seed As TAccumulate,  
    func As Func(Of TAccumulate, TSource, TAccumulate),  
    resultSelector As Func(Of TAccumulate, TResult)  
) As TResult

C#

public static TResult Aggregate<TSource, TAccumulate, TResult>(  
    ParallelQuery<TSource> source,  
    TAccumulate seed,  
    Func<TAccumulate, TSource, TAccumulate> func,  
    Func<TAccumulate, TResult> resultSelector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of (Of TSource))>)
    A sequence to aggregate over.

seed
    Type: TAccumulate
    The initial accumulator value.

func
    Type: System.Func(Of (Of (Of TAccumulate, TSource, TAccumulate))>)
    An accumulator function to be invoked on each element.

resultSelector
    Type: System.Func(Of (Of (Of TAccumulate, TResult))>)
    A function to transform the final accumulator value into the result value.
**Type Parameters**

**TSource**
The type of the elements of source.

**TAccumulate**
The type of the accumulator value.

**TResult**
The type of the resulting value.

**Return Value**

The transformed final accumulator value.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or func or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Aggregate Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::Aggregate(Of (TSource, TAccumulate, TResult)>)
Method (ParallelQuery(Of (TSource)<>), Func(Of (TAccumulate)<>), Func(Of (TAccumulate, TSource, TAccumulate)<>), Func(Of (TAccumulate, TAccumulate, TAccumulate)<>), Func(Of (TAccumulate, TResult)<>))

ParallelEnumerable Class  See Also  Send Feedback

Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Aggregate(Of TSource, TAccumulate, TResult) (source As ParallelQuery(Of TSource), _
seedFactory As Func(Of TAccumulate), _
updateAccumulatorFunc As Func(Of TAccumulate, TSource, TAccumulate, TAccumulate, TResult), _
combineAccumulatorsFunc As Func(Of TAccumulate, TAccumulate, TAccumulate), _
resultSelector As Func(Of TAccumulate, TResult)) _
) As TResult

C#

public static TResult Aggregate<TSource, TAccumulate, TResult>(
ParallelQuery<TSource> source,
Func<TAccumulate> seedFactory,
Func<TAccumulate, TSource, TAccumulate> updateAccumulatorFunc,
Func<TAccumulate, TAccumulate, TAccumulate> combineAccumulatorsFunc,
Func<TAccumulate, TResult> resultSelector
)

Parameters

source
Type: System.Linq.ParallelQuery(Of (Of (TSource}))
A sequence to aggregate over.

seedFactory
Type: System.Func(Of (TAccumulate))
A function that returns the initial accumulator value.

updateAccumulatorFunc
Type: System.Func(Of (TAccumulate, TSource, TAccumulate, TAccumulate))
An accumulator function to be invoked on each element in a partition.

combineAccumulatorsFunc
Type: System.Func(Of (TAccumulate, TAccumulate, TAccumulate))

An accumulator function to be invoked on the yielded element from each partition.

resultSelector
Type: System::Func<(Of <(TAccumulate, TResult)>))
A function to transform the final accumulator value into the result value.
Type Parameters

TSource
    The type of the elements of source.
TAccumulate
    The type of the accumulator value.
TResult
    The type of the resulting value.

Return Value

The transformed final accumulator value.
Remarks

This overload is specific to parallelized queries. A parallelized query may partition the data source sequence into several sub-sequences (partitions). The updateAccumulatorFunc is invoked on each element within partitions. Each partition then yields a single accumulated result. The combineAccumulatorsFunc is then invoked on the results of each partition to yield a single element. This element is then transformed by the resultSelector function.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source or seedFactory or updateAccumulatorFunc or combineAccumulatorsFunc or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System..::.AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System..::.OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Aggregate Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library
ParallelEnumerable...:::Aggregate(Of (TSOURCE, TAccumulate, TRESULT)>)
Method (ParallelQuery(Of (TSOURCE)>, TAccumulate, Func(Of (TAccumulate, TSOURCE, TAccumulate)>)>, Func(Of (TAccumulate, TAccumulate, TAccumulate)>)>, Func(Of (TAccumulate, TRESULT)>)

ParallelEnumerable Class  See Also  Send Feedback

Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function Aggregate(Of TSource, TAccumulate, TResult) (  
    source As ParallelQuery(Of TSource), _  
    seed As TAccumulate, _  
    updateAccumulatorFunc As Func(Of TAccumulate, TSource, TAccumulate), _  
    combineAccumulatorsFunc As Func(Of TAccumulate, TAccumulate, TAccumulate), _  
    resultSelector As Func(Of TAccumulate, TResult)) _

) As TResult
```

#### C#

```csharp
public static TResult Aggregate<TSource, TAccumulate, TResult>(  
    ParallelQuery<TSource> source,  
    TAccumulate seed,  
    Func<TAccumulate, TSource, TAccumulate> updateAccumulatorFunc  
    Func<TAccumulate, TAccumulate, TAccumulate> combineAccumulatorsFunc  
    Func<TAccumulate, TResult> resultSelector
)
```

### Parameters

**source**

Type: `.System.Linq.ParallelQuery<Of <(TSource)>>`

A sequence to aggregate over.

**seed**

Type: TAccumulate

The initial accumulator value.

**updateAccumulatorFunc**

Type: `.System.Func<Of <(TAccumulate, TSource, TAccumulate)>>`

An accumulator function to be invoked on each element in a partition.

**combineAccumulatorsFunc**

Type: `.System.Func<Of <(TAccumulate, TAccumulate, TAccumulate)>>`


An accumulator function to be invoked on the yielded element from each partition.

**resultSelector**
- **Type:** `System::Func<Of<TAccumulate, TResult>>`
- A function to transform the final accumulator value into the result value.
Type Parameters

TSource
   The type of the elements of source.
TAccumulate
   The type of the accumulator value.
TResult
   The type of the resulting value.

Return Value

The transformed final accumulator value.
Remarks

This overload is specific to processing a parallelized query. A parallelized query may partition the data source sequence into several sub-sequences (partitions). The `updateAccumulatorFunc` is invoked on each element within partitions. Each partition then yields a single accumulated result. The `combineAccumulatorsFunc` is then invoked on the results of each partition to yield a single element. This element is then transformed by the `resultSelector` function.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source or <code>updateAccumulatorFunc</code> or <code>combineAccumulatorsFunc</code> or <code>resultSelector</code> is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System..::.AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System..::.OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Aggregate Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Determines in parallel whether all elements of a sequence satisfy a condition.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function All(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As Boolean

C#

public static bool All<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
    Type: System.Linq.ParallelQuery<TSource>(Of (Of<TSource>)>)
    A sequence whose elements to apply the predicate to.

predicate
    Type: System.Func(Of (Of<TSource, Boolean>)>)
    A function to test each element for a condition.
Type Parameters

TSource
   The type of elements of source.

Return Value

true if all elements in the source sequence pass the test in the specified predicate; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Any Method
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any&lt;(Of &lt;(TSource)&gt;&gt;) ((ParallelQuery&lt;(Of &lt;(TSource)&gt;&gt;)))</td>
<td>Determines whether a parallel sequence contains any elements.</td>
</tr>
<tr>
<td>Any&lt;(Of &lt;(TSource)&gt;&gt;) ((ParallelQuery&lt;(Of &lt;(TSource)&gt;&gt;), Func&lt;(Of &lt;(TSource, Boolean)&gt;&gt;)))</td>
<td>Determines in parallel whether any element of a sequence satisfies a condition.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Determines whether a parallel sequence contains any elements.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

```vbnet
Public Shared Function Any(Of TSource) ( _
    source As ParallelQuery(Of TSource) _) _
) As Boolean
```

C#

```csharp
public static bool Any<TSource>(
    ParallelQuery<TSource> source
)
```

Parameters

source
   Type: System.Linq.Enumerable.ParallelQuery(Of (Of (Of )>)
The IEnumerable to check for emptiness.
Type Parameters

TSource
The type of elements of source.

Return Value

true if the source sequence contains any elements; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Any Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Determines in parallel whether any element of a sequence satisfies a condition.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Any(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As Boolean

C#

public static bool Any<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of (Of TSource)>))
    An IEnumerable whose elements to apply the predicate to.

predicate
    Type: System.Func(Of (Of (Of TSource, Boolean)>))
    A function to test each element for a condition.
Type Parameters

TSource
The type of elements of source.

Return Value

true if any elements in the source sequence pass the test in the specified predicate; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Any Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Converting a `ParallelQuery<Of <(TSource)>>` into an `IEnumerable<Of <(T)>>` to force sequential evaluation of the query.

**Namespace:** `System.Linq`

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function AsEnumerable(Of TSource) (_
    source As ParallelQuery(Of TSource) _
) As IEnumerable(Of TSource)

C#

public static IEnumerable<TSource> AsEnumerable<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of (TSource)>)
    The sequence to type as IEnumerable(Of (T)>).
Type Parameters

TSource
  The type of the elements of source.

Return Value

The input sequence types as IEnumerable<Of <(T)>>.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.AsOrdered Method

ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsOrdered(ParallelQuery)</td>
<td>Enables treatment of a data source as if it was ordered, overriding the default of unordered. AsOrdered may only be invoked on sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.</td>
</tr>
<tr>
<td>AsOrdered(Of (Of (TSource)&gt;)&gt;) (ParallelQuery(Of (TSource)&gt;)</td>
<td>Enables treatment of a data source as if it was ordered, overriding the default of unordered. AsOrdered may only be invoked on sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Enables treatment of a data source as if it was ordered, overriding the default of unordered. AsOrdered may only be invoked on sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function AsOrdered ( _
    source As ParallelQuery _
) As ParallelQuery

C#

public static ParallelQuery AsOrdered(
    ParallelQuery source
)

Parameters

source
    Type: System.Linq::ParallelQuery
    The input sequence.

Return Value

The source sequence which will maintain ordering in the query.
Remarks

A natural tension exists between performance and preserving order in parallel processing. By default, a parallelized query behaves as if the ordering of the results is arbitrary unless AsOrdered is applied or there is an explicit OrderBy operator in the query.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>Thrown if the source is not one of AsParallel, ParallelEnumerable.Range, or ParallelEnumerable.Repeat.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
AsOrdered Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Enables treatment of a data source as if it was ordered, overriding the default of unordered. AsOrdered may only be invoked on sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function AsOrdered(Of TSource) ( _
   source As ParallelQuery(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> AsOrdered<TSource>(
   ParallelQuery<TSource> source
)

Parameters

source
   Type: System.Linq.ParallelQuery(Of (Of (TSource))>)
   The input sequence.
Type Parameters

TSource
   The type of elements of source.

Return Value

The source sequence which will maintain ordering in the query.
Remarks

A natural tension exists between performance and preserving order in parallel processing. By default, a parallelized query behaves as if the ordering of the results is arbitrary unless AsOrdered is applied or there is an explicit OrderBy operator in the query.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>Thrown if source is not one of AsParallel, ParallelEnumerable.Range, or</td>
</tr>
<tr>
<td></td>
<td>ParallelEnumerable.Repeat.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
AsOrdered Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...::AsParallel Method
ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AsParallel(Of (Of TSource)&gt;) (Partitioner(Of (Of TSource)&gt;)])</code></td>
<td>Enables parallelization of a query, as sourced by a partitioner responsible for splitting the input sequence into partitions.</td>
</tr>
<tr>
<td><code>AsParallel(Of (Of TSource)&gt;) (IEnumerable(Of (Of TSource)&gt;)])</code></td>
<td>Enables parallelization of a query.</td>
</tr>
<tr>
<td><code>AsParallel(IEnumerable)</code></td>
<td>Enables parallelization of a query.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Enables parallelization of a query, as sourced by a partitioner responsible for splitting the input sequence into partitions.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

Public Shared Function AsParallel(Of TSource) ( _
    source As Partitioner(Of TSource) _
) As ParallelQuery(Of TSource)

**C#**

public static ParallelQuery<TSource> AsParallel<TSource>(
    Partitioner<TSource> source
)

**Parameters**

source

Type: System.Collections.Concurrent:::Partitioner(Of (Of (TSource))>

A partitioner over the input sequence.
Type Parameters

TSource
   The type of elements of source.

Return Value

The source as a ParallelQuery to bind to ParallelEnumerable extension methods.
Remarks

The source partitioner's GetOrderedPartitions method is used when ordering is enabled, whereas the partitioner's GetPartitions is used if ordering is not enabled (the default). The source partitioner's GetDynamicPartitions and GetDynamicOrderedPartitions are not used.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
AsParallel Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Enables parallelization of a query.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function AsParallel(Of TSource) ( _
    source As IEnumerable(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> AsParallel<TSource>(
    IEnumerable<TSource> source
)

Parameters

source
    Type: System.Collections.Generic.IEnumerable(Of TSource)
    An IEnumerable(Of T) to convert to a ParallelQuery(Of T).
### Type Parameters

**TSource**

The type of elements of source.

### Return Value

The source as a `ParallelQuery(Of <(TSource)>)` to bind to ParallelEnumerable extension methods.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
AsParallel Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Enables parallelization of a query.

**Namespace:**  System.Linq
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function AsParallel (_
    source As IEnumerable _
) As ParallelQuery

C#

public static ParallelQuery AsParallel(
    IEnumerable source
)

Parameters

source
    Type: System.Collections:::IEnumerable
    An IEnumerable<(Of <(T)>)) to convert to a ParallelQuery<(Of <(TSource)>)).

Return Value

The source as a ParallelQuery to bind to ParallelEnumerable extension methods.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
AsParallel Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Converts a `ParallelQuery(Of (Of<TSource>>)>)` into an `IEnumerable(Of (Of'T>>)` to force sequential evaluation of the query.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function AsSequential(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As IEnumerable(Of TSource)

C#

public static IEnumerable<TSource> AsSequential<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq.Enumerable.ParallelQuery(Of Of TSource)>
A ParallelQuery(Of Of TSource) to convert to an IEnumerable(Of Of TSource).
Type Parameters

TSource
   The type of elements of source.

Return Value

The source as an IEnumerable<(Of <(T)>)> to bind to sequential extension methods.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Allows an intermediate query to be treated as if no ordering is implied among the elements.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

Visual Basic (Declaration)

Public Shared Function AsUnordered(Of TSource) ( _
  source As ParallelQuery(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> AsUnordered<TSource>(
  ParallelQuery<TSource> source
)

**Parameters**

source
  Type: System.Linq.ParallelQuery(Of TSource)<(Of <(TSource)>>)
  The input sequence.
Type Parameters

TSource
The type of elements of source.

Return Value
The source sequence with arbitrary order.
Remarks

AsUnordered may provide performance benefits when ordering is not required in a portion of a query.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Average Method
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Decimal)&gt;&gt;)</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Double)&gt;&gt;)</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Int32)&gt;&gt;)</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Int64)&gt;&gt;)</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Decimal)&gt;)&gt;)&gt;))</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Double)&gt;)&gt;)&gt;))</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Int32)&gt;)&gt;)&gt;))</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Int64)&gt;)&gt;)&gt;))</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Single)&gt;)&gt;)&gt;))</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average(ParallelQuery&lt;Of &lt;(Single)&gt;)&gt;))</td>
<td>Computes in parallel the average of a sequence of values.</td>
</tr>
<tr>
<td>Average&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;Of &lt;(TSource)&gt;), Func&lt;Of &lt;(TSource, Decimal)&gt;&gt;))</td>
<td>Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.</td>
</tr>
</tbody>
</table>
| Average<(Of <(TSource)>)> (ParallelQuery<Of <(TSource)>), Func<Of <(TSource, Decimal)>>)) | Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the.
Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average(Of TSource>)
(ParallelQuery(Of TSource>), Func(Of TSource, Single>))
a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values.

**Namespace:** [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average ( _
    source As ParallelQuery(Of Decimal) _
) As Decimal

C#

public static decimal Average(
    ParallelQuery<decimal> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<<(Of <<(Decimal)>>)>
    A sequence of values that are used to calculate an average.

Return Value

The average of the sequence of values.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average ( source As ParallelQuery(Of Double) ) As Double

C#

public static double Average( ParallelQuery<double> source )

Parameters

source
Type: System.Linq.:::ParallelQuery<Of <(Double)>>
A sequence of values that are used to calculate an average.

Return Value

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average ( source As ParallelQuery(Of Integer) ) As Double

C#

public static double Average( ParallelQuery<int> source )

Parameters

source
Type: System.Linq.ParallelQuery(Of (Int32))
A sequence of values that are used to calculate an average.

Return Value

The average of the sequence of values.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>The sum or count of the elements in the sequence is larger than MaxValue().</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
.NET Framework Class Library
ParallelEnumerable...:::Average Method (ParallelQuery<Of <(Int64)>))
ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the average of a sequence of values.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function Average (source As ParallelQuery(Of Long)) As Double
```

**C#**

```csharp
public static double Average(ParallelQuery<long> source)
```

### Parameters

**source**

Type: `System.Linq.ParallelQuery(Of (Int64)>)`

A sequence of values that are used to calculate an average.

### Return Value

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The sum or count of the elements in the sequence is larger than <code>MaxValue()()</code>.</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Average Method (ParallelQuery<Of Nullable<Of (Of Value)>)>)

ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the average of a sequence of values.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average ( _
    source As ParallelQuery(Of Nullable(Of Decimal)) _
) As Nullable(Of Decimal)

C#

public static Nullable<decimal> Average(
    ParallelQuery<Nullable<decimal>> source
)

Parameters

source
    Type: System.Linq.Enumerable.ParallelQuery<(Of Nullable(Of Decimal))>
    A sequence of values that are used to calculate an average.

Return Value

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::Average Method (ParallelQuery<(Of <(Nullable<(Of <(Double>>)>>)>>)>

ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the average of a sequence of values.

**Namespace:**  System.Linq
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average ( _
    source As ParallelQuery(Of Nullable(Of Double)) _
) As Nullable(Of Double)

C#

public static Nullable<double> Average(    
ParallelQuery<Nullable<double>> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<Nullable<Nullable<Nullable<Double>>>>
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>A sequence of values that are used to calculate an average.</td>
<td>The average of the sequence of values. source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
<td></td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average ( _
    source As ParallelQuery(Of Nullable(Of Integer)) _
) As Nullable(Of Double)

C#

public static Nullable<double> Average(
    ParallelQuery<Nullable<int>> source
)

Parameters

source
    Type: System.Linq.:::ParallelQuery<(Of ((Of (Int32)>))>)
    A sequence of values that are used to calculate an average.

Return Value

The average of the sequence of values.
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td></td>
<td>The sum or count of the elements in the sequence is larger than MaxValue().</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>The query was canceled.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::Average Method (ParallelQuery<(Of <(Nullable<(Of <(Int64)>>)>)>)

ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the average of a sequence of values.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average (_
    source As ParallelQuery(Of Nullable(Of Long)) _) As Nullable(Of Double)

C#

public static Nullable<double> Average(
    ParallelQuery<Nullable<long>> source)

Parameters

source
    Type: System.Linq.Enumerable.ParallelQuery(Of Nullable(Of Int64))
    A sequence of values that are used to calculate an average.

Return Value

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The sum or count of the elements in the sequence is larger than MaxValue().</td>
</tr>
<tr>
<td>-or- One or more exceptions occurred during</td>
<td></td>
</tr>
<tr>
<td>the evaluation of the query.</td>
<td></td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
ParallelEnumerable..:::Average Method (ParallelQuery<(Of <(Nullable<(Of <(Single>>)>>)>)

ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the average of a sequence of values.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function Average (_
    source As ParallelQuery(Of Nullable(Of Single)) _
) As Nullable(Of Single)
```

**C#**

```csharp
public static Nullable<float> Average(
    ParallelQuery<Nullable<float>> source
)
```

**Parameters**

**source**

Type: `System.Linq.ParallelQuery<Nullable<float>>`<br>
A sequence of values that are used to calculate an average.

**Return Value**

The average of the sequence of values.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values.

**Namespace:**  [System.Linq]

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average ( _
    source As ParallelQuery(Of Single) _
) As Single

C#

public static float Average(
    ParallelQuery<float> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<Of <(Single)>>
    A sequence of values that are used to calculate an average.

Return Value

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class  
Average Overload  
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Decimal) _
) As Decimal

C#

public static decimal Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, decimal> selector
)

Parameters

source
    Type: System.Linq::.ParallelQuery(Of (Of TSource)>)
    A sequence of values that are used to calculate an average.

selector
    Type: System::.Func(Of (Of TSource, Decimal)>)
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The average of the sequence of values.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Double) _
) As Double
```

**C#**

```csharp
public static double Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, double> selector
)
```

**Parameters**

**source**

Type: `System.Linq.ParallelQuery<TSource>`

A sequence of values that are used to calculate an average.

**selector**

Type: `System.Func<TSource, double>`

A transform function to apply to each element.
Type Parameters

TSource
   The type of elements of source.

Return Value

The average of the sequence of values.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Average<TSource>() Method
(ParallelQuery<TSource>(), Func<TSource, Int32>))

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Integer) _
) As Double

C#

public static double Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, int> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery<(Of Of(TSource))>
    A sequence of values that are used to calculate an average.

selector
    Type: System.Func<(Of Of(TSource, Int32))>
    A transform function to apply to each element.
**Type Parameters**

TSource
   The type of elements of source.

**Return Value**

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
</tbody>
</table>
| **System...:::AggregateException** | The sum or count of the elements in the sequence is larger than MaxValue().  
|                                    | -or- One or more exceptions occurred during the evaluation of the query.  |
| System...:::OperationCanceledException | The query was canceled.                                                   |
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Long) _
) As Double

C#

public static double Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, long> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)
    A sequence of values that are used to calculate an average.

selector
    Type: System.Func(Of TSource, Int64)
    A transform function to apply to each element.
Type Parameters

TSource
   The type of elements of source.

Return Value

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::&lt;ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System::&lt;InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System::&lt;AggregateException</td>
<td>The sum or count of the elements in the sequence is larger than MaxValue().</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System::&lt;OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Average<TSource>() Method
(ParallelQuery<TSource>, Func<TSource, Nullable<decimal>>)
Syntax

Visual Basic (Declaration)

Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Decimal)) _
) As Nullable(Of Decimal)

C#

public static Nullable<decimal> Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<decimal>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)
    A sequence of values that are used to calculate an average.

selector
    Type: System.Func(Of TSource, Nullable<decimal>)
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The average of the sequence of values.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Average(Of TSource) Method

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Double)) _
) As Nullable(Of Double)

C#

public static Nullable<double> Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<double>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (TSource))
    A sequence of values that are used to calculate an average.

selector
    Type: System.Func(Of (TSource, Nullable(Of (Double))>)
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

ReturnValue
The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::Average(Of (Of TSource)>)> Method
(ParallelQuery(Of (Of TSource)>), Func(Of (Of TSource, Nullable(Of (Int32)>)>)>))

ParallelEnumerable Class   See Also   Send Feedback

 Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Integer)) ) _
) As Nullable(Of Double)

C#

public static Nullable<double> Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<int>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)>
    A sequence of values that are used to calculate an average.

selector
    Type: System.Func(Of Nullable<int>>(Of Nullable<TSource>)>)
    A transform function to apply to each element.
Type Parameters

TSource
   The type of elements of source.

Return Value

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>Source or selector is a null reference (Nothing in Visual Basic). The sum or count of the elements in the sequence is larger than <code>MaxValue()</code>(). -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td></td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Shared Function Average(Of TSource) (_
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Long))) _
) As Nullable(Of Double)

C#

public static Nullable<double> Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<long>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource)>)
    A sequence of values that are used to calculate an average.

selector
    Type: System.Func(Of (Of TSource, Nullable(Of (Of Int64)>)>)
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value

The average of the sequence of values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic). The sum or count of the elements in the sequence is larger than MaxValue(). -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td></td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Average(Of TSource) Method
(ParallelQuery(Of TSource), Func(Of TSource, Nullable(Of Single)>)
)

ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Single))) _
) As Nullable(Of Single)

### C#

public static Nullable<float> Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<float>> selector
)

### Parameters

**source**
- Type: System.Linq:::ParallelQuery<(Of <(TSource)>))
- A sequence of values that are used to calculate an average.

**selector**
- Type: System:::Func<(Of <(TSource, Nullable<(Of <(Single)>))>)>)
- A transform function to apply to each element.
Type Parameters

TSource
   The type of elements of source.

Return Value

The average of the sequence of values.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Average<TSource>(Func<TSource, Single>) Method (ParallelEnumerable, System.Linq)

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Average(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Single) _
) As Single

C#

public static float Average<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, float> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource>)
    A sequence of values that are used to calculate an average.

selector
    Type: System.Func(Of TSource, Single>)
    A transform function to apply to each element.
**Type Parameters**

**TSource**  
The type of elements of source.

**Return Value**

The average of the sequence of values.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Average Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Converts the elements of a ParallelQuery to the specified type.

**Namespace:**  System.Linq
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Cast(Of TResult) ( _
    source As ParallelQuery _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> Cast<TResult>(
    ParallelQuery source
)

Parameters

source
    Type: System.Linq. :: ParallelQuery
    The sequence that contains the elements to be converted.
Type Parameters

TResult

The type to convert the elements of source to.

Return Value

A sequence that contains each element of the source sequence converted to the specified type.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

Include Protected Members
Include Inherited Members

.NET Framework Class Library

ParallelEnumerable...:::Concat Method

ParallelEnumerable Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Obsoleted.**  
Concat<(Of <(TSource)>>)  
(ParallelQuery<(Of  
<TSource>>), IEnumerable<(Of  
<TSource>>)) | This Concat overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called. |
| Concat<(Of <(TSource)>>)  
(ParallelQuery<(Of  
<TSource>>),  
ParallelQuery<(Of  
<TSource>>)) | Concatenates two parallel sequences. |
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
This Concat overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

```csharp
[ObsoleteAttribute("The second data source of a binary operator must use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static ParallelQuery<TSource> Concat<TSource>(
    ParallelQuery<TSource> first,
    IEnumerable<TSource> second
) As ParallelQuery<TSource>
```

Parameters

**first**
- Type: `System.Linq.ParallelQuery<TSource>`
  - This parameter is not used.

**second**
- Type: `System.Collections.Generic.IEnumerable<TSource>`
  - This parameter is not used.
Type Parameters

TSource
   This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Concat with a left data source of type `ParallelQuery<Of <(TSourc)>>` and a right data source of type `IEnumerable<Of <(T)>>`. Otherwise, the Concat operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Concat Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Concatenates two parallel sequences.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Concat(Of TSource) ( _  
    first As ParallelQuery(Of TSource), _ 
    second As ParallelQuery(Of TSource) _ 
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Concat<TSource>(
    ParallelQuery<TSource> first,
    ParallelQuery<TSource> second
)

Parameters

first
    Type: System.Linq.ParallelQuery<TSource>(Of <=(TSource)>)
    The first sequence to concatenate.

second
    Type: System.Linq.ParallelQuery<TSource>(Of <=(TSource)>)
    The sequence to concatenate to the first sequence.
**Type Parameters**

**TSource**
The type of the elements of the input sequences.

**Return Value**

A sequence that contains the concatenated elements of the two input sequences.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.ArgumentNullException</code></td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Concat Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::Contains Method

ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `Contains<(Of <(TSourse)>)>
(ParallelQuery<(Of
<TSource>)>, TSource)` | Determines in parallel whether a sequence contains a specified element by using the default equality comparer. |
| `Contains<(Of <(TSourse)>)>
(ParallelQuery<(Of
<TSource>)>, TSource,
IEqualityComparer<(Of
<TSource>)>)` | Determines in parallel whether a sequence contains a specified element by using a specified IEqualityComparer{T}. |
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Determines in parallel whether a sequence contains a specified element by using the default equality comparer.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Contains(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    value As TSource _) As Boolean

C#

public static bool Contains<TSource>(
    ParallelQuery<TSource> source,
    TSource value
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of <TSource>)
    A sequence in which to locate a value.

value
    Type: TSource
    The value to locate in the sequence.
Type Parameters

TSource
    The type of elements of source.

Return Value

true if the source sequence contains an element that has the specified value; otherwise, false.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Contains Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Determines in parallel whether a sequence contains a specified element by using a specified IEqualityComparer{T}.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Contains(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    value As TSource, _
    comparer As IEqualityComparer(Of TSource) _
) As Boolean

C#

public static bool Contains<TSource>(
    ParallelQuery<TSource> source,
    TSource value,
    IEqualityComparer<TSource> comparer
)

Parameters

source
Type: System.Linq.ParallelQuery(Of TSource>)
A sequence in which to locate a value.

value
Type: TSource
The value to locate in the sequence.

comparer
Type: System.Collections.Generic.IEqualityComparer(Of TSource>)
An equality comparer to compare values.
Type Parameters

TSource
   The type of elements of source.

Return Value

true if the source sequence contains an element that has the specified value; otherwise, false.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Contains Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Count Method

ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;))</td>
<td>Returns the number of elements in a parallel sequence.</td>
</tr>
<tr>
<td>Count&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;&gt;), Func&lt;(Of &lt;(TSource, Boolean)&gt;)&gt;))</td>
<td>Returns a number that represents how many elements in the specified parallel sequence satisfy a condition.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Count<TSource>() Method (ParallelQuery<TSource>)

Returns the number of elements in a parallel sequence.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Count(Of TSource) (  
    source As ParallelQuery(Of TSource)  
) As Integer

C#

public static int Count<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq::ParallelQuery(Of <(TSource)>)
    A sequence that contains elements to be counted.
Type Parameters

TSource

The type of the elements of source.

Return Value

The number of elements in the input sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic). The number of elements in source is larger than MaxValue(). -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The query was canceled.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Count Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns a number that represents how many elements in the specified parallel sequence satisfy a condition.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Count(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As Integer

C#

public static int Count<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
    Type: System.Linq:::ParallelQuery<(Of <(TSource)>))
    A sequence that contains elements to be counted.

predicate
    Type: System:::Func<(Of <(TSource, Boolean)>)
    A function to test each element for a condition.
Type Parameters

TSource
The type of the elements of source.

Return Value

A number that represents how many elements in the sequence satisfy the condition in the predicate function.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The number of elements in source is larger than MaxValue(). -or- One or</td>
</tr>
<tr>
<td></td>
<td>more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Count Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:::DefaultIfEmpty Method
ParallelEnumerable Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DefaultIfEmpty(Of TSource)</code>(ParallelQuery(Of TSource))</td>
<td>Returns the elements of the specified parallel sequence or the type parameter's default value in a singleton collection if the sequence is empty.</td>
</tr>
<tr>
<td><code>DefaultIfEmpty(Of TSource)</code>(ParallelQuery(Of TSource), TSource)</td>
<td>Returns the elements of the specified parallel sequence or the specified value in a singleton collection if the sequence is empty.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the elements of the specified parallel sequence or the type parameter's default value in a singleton collection if the sequence is empty.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function DefaultIfEmpty(Of TSource) ( _
  source As ParallelQuery(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> DefaultIfEmpty<TSource>(
  ParallelQuery<TSource> source
)

Parameters

source
  Type: System.Linq.ParallelQuery(Of (Of (TSource)))
  The sequence to return a default value for if it is empty.
### Type Parameters

**TSource**

The type of the elements of source.

### Return Value

A sequence that contains default(TSource) if source is empty; otherwise, source.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
DefaultIfEmpty Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.DefaultIfEmpty(Of TSource) Method (ParallelQuery(Of TSource), TSource)

Returns the elements of the specified parallel sequence or the specified value in a singleton collection if the sequence is empty.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function DefaultIfEmpty(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    defaultValue As TSource _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> DefaultIfEmpty<TSource>(
    ParallelQuery<TSource> source,
    TSource defaultValue
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource)>)
    The sequence to return the specified value for if it is empty.

defaultValue
    Type: TSource
    The value to return if the sequence is empty.
**Type Parameters**

**TSource**

The type of the elements of source.

**Return Value**

A sequence that contains defaultValue if source is empty; otherwise, source.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
DefaultIfEmpty Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:Distinct Method

ParallelEnumerable Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct(Of (TSource&gt;)) (ParallelQuery(Of (TSource&gt;)))</td>
<td>Returns distinct elements from a parallel sequence by using the default equality comparer to compare values.</td>
</tr>
<tr>
<td>Distinct(Of (TSource&gt;)) (ParallelQuery(Of (TSource&gt;)), IEqualityComparer(Of (TSource&gt;)))</td>
<td>Returns distinct elements from a parallel sequence by using a specified IEqualityComparer{T} to compare values.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns distinct elements from a parallel sequence by using the default equality comparer to compare values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Distinct(Of TSource) (  _
    source As ParallelQuery(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Distinct<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq::ParallelQuery(Of (Of TSource))>
    The sequence to remove duplicate elements from.
Type Parameters

TSource
    The type of the elements of source.

Return Value

A sequence that contains distinct elements from the source sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Distinct Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns distinct elements from a parallel sequence by using a specified IEqualityComparer{T} to compare values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Distinct(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    comparer As IEqualityComparer(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Distinct<TSource>(
    ParallelQuery<TSource> source,
    IEqualityComparer<TSource> comparer
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)>
    The sequence to remove duplicate elements from.

comparer
    Type: System.Collections.Generic.IEqualityComparer(Of TSource)>
    An IEqualityComparer(Of TSource) to compare values.
Type Parameters

TSource
   The type of the elements of source.

Return Value

A sequence that contains distinct elements from the source sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Distinct Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the element at a specified index in a parallel sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ElementAt(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    index As Integer _
) As TSource

C#

public static TSource ElementAt<TSource>(
    ParallelQuery<TSource> source,
    int index
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource>)
    A sequence to return an element from.

index
    Type: System.Int32
    The zero-based index of the element to retrieve.
Type Parameters

TSource
   The type of the elements of source.

Return Value

The element at the specified position in the source sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>index is less than 0 or greater than or equal to the number of elements in source.</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the element at a specified index in a parallel sequence or a default value if the index is out of range.

**Namespace:**  [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ElementAtOrDefault(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    index As Integer _
) As TSource

C#

public static TSource ElementAtOrDefault<TSource>(
    ParallelQuery<TSource> source,
    int index
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)
    A sequence to return an element from.

index
    Type: System.Int32
    The zero-based index of the element to retrieve.
Type Parameters

TSource
   The type of the elements of source.

Return Value

default(TSource) if the index is outside the bounds of the source sequence; otherwise, the element at the specified position in the source sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns an empty ParallelQuery{TResult} that has the specified type argument.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Empty(Of TResult) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> Empty<TResult>()
Type Parameters

TResult

The type to assign to the type parameter of the returned generic sequence.

Return Value

An empty sequence whose type argument is TResult.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
☐ Include Protected Members
☐ Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...::Except Method
ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Except&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;), IEnumerable&lt;(Of &lt;(TSource)&gt;)&gt;))</td>
<td>Obsolete. This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called. Produces the set difference of two parallel sequences by using the default equality comparer to compare values.</td>
</tr>
<tr>
<td>Except&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;), ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;))</td>
<td>Obsolete. This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called. Produces the set difference of two parallel sequences by using the specified IEqualityComparer{T} to compare values.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

```csharp
<ObsoleteAttribute("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Linq.Enumerable<T>.")>

Public Shared Function Except(Of TSource) ( _
   first As ParallelQuery(Of TSource), _
   second As IEnumerable(Of TSource) _
) As ParallelQuery(Of TSource)
```

```csharp
public static ParallelQuery<TSource> Except<TSource>(
   ParallelQuery<TSource> first,
   IEnumerable<TSource> second
)
```

**Parameters**

**first**

Type: `System.Linq.ParallelQuery<TSource>()`  
This parameter is not used.

**second**

Type: `System.Collections.Generic.IEnumerable<TSource>()`  
This parameter is not used.
Type Parameters

TSource
  This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Except with a left data source of type `ParallelQuery(Of<TSource>)>` and a right data source of type `IEnumerable(Of<T>)>`. Otherwise, the Except operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Except Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.Except(Of (Of (TSource)>) Method (ParallelQuery(Of (Of (TSource)>)>, ParallelQuery(Of (Of (TSource)>)>))

ParallelEnumerable Class  See Also  Send Feedback

Produce the set difference of two parallel sequences by using the default equality comparer to compare values.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Except(Of TSource) ( _
  first As ParallelQuery(Of TSource), _
  second As ParallelQuery(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Except<TSource>(
  ParallelQuery<TSource> first,
  ParallelQuery<TSource> second
)

Parameters

first
  Type: System.Linq.ParallelQuery(Of (Of (TSource))>
  A sequence whose elements that are not also in second will be returned.

second
  Type: System.Linq.ParallelQuery(Of (Of (TSource))>
  A sequence whose elements that also occur in the first sequence will cause those elements to be removed from the returned sequence.
Type Parameters

TSource

The type of the elements of the input sequences.

Return Value

A sequence that contains the set difference of the elements of two sequences.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Except Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

```csharp
<ObsoleteAttribute("The second data source of a binary operator must
be of type System.Linq.ParallelQuery<TSource>.")>
Public Shared Function Except(Of TSource) ( _
  first As ParallelQuery(Of TSource), _
  second As IEnumerable(Of TSource), _
  comparer As IEqualityComparer(Of TSource) _
) As ParallelQuery(Of TSource)

[ObsoleteAttribute("The second data source of a binary operator must
be of type System.Linq.ParallelQuery<TSource>.")]
public static ParallelQuery<TSource> Except<TSource>(
  ParallelQuery<TSource> first,
  IEnumerable<TSource> second,
  IEqualityComparer<TSource> comparer
)
```

### Parameters

**first**
Type: `System.Linq::ParallelQuery<Of <(TSource)>>`
This parameter is not used.

**second**
Type: `System.Collections.Generic::IEnumerable<Of <(TSource)>>`
This parameter is not used.

**comparer**
Type: `System.Collections.Generic::IEqualityComparer<Of <(TSource)>>`
This parameter is not used.
Type Parameters

TSource
   This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Except with a left data source of type `ParallelQuery<Of <TSource>>` and a right data source of type `IEnumerable<Of <*T*>>`. Otherwise, the Except operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Except Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Produces the set difference of two parallel sequences by using the specified IEqualityComparer{T} to compare values.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Except(Of TSource) ( _
   first As ParallelQuery(Of TSource), _
   second As ParallelQuery(Of TSource), _
   comparer As IEqualityComparer(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Except<TSource>(
   ParallelQuery<TSource> first,
   ParallelQuery<TSource> second,
   IEqualityComparer<TSource> comparer
)

Parameters

first
   Type: System.Linq.ParallelQuery(Of (Of TSource)>)
   A sequence whose elements that are not also in second will be returned.

second
   Type: System.Linq.ParallelQuery(Of (Of TSource)>)
   A sequence whose elements that also occur in the first sequence will cause those elements to be removed from the returned sequence.

comparer
   Type: System.Collections.Generic.IEqualityComparer(Of (Of (TSource)>)
   An IEqualityComparer(Of (Of T)>) to compare values.
Type Parameters

TSourse
  The type of the elements of the input sequences.

Return Value

A sequence that contains the set difference of the elements of two sequences.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Except Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...First Method

ParallelEnumerable Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;))</td>
<td>Returns the first element of a parallel sequence.</td>
</tr>
<tr>
<td>First&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;, Func&lt;(Of &lt;(TSource, Boolean)&gt;)&gt;)</td>
<td>Returns the first element in a parallel sequence that satisfies a specified condition.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.First(Of (TSource)>) Method (ParallelQuery(Of (TSource)>))

ParallelEnumerable Class  See Also  Send Feedback

Returns the first element of a parallel sequence.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function First(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As TSource

C#

public static TSource First<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source

Type: System.Linq.Async.ParallelQuery(Of (Of TSource)>)
The sequence to return the first element of.
Type Parameters

TSource
    The type of the elements of source.

Return Value

The first element in the specified sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
First Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.First(Of (Of TSource)) Method (ParallelQuery(Of (Of TSource)), Func(Of (Of TSource, Boolean)))

ParallelEnumerable Class  See Also  Send Feedback

Returns the first element in a parallel sequence that satisfies a specified condition.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function First(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As TSource

C#

public static TSource First<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
    Type: System.Linq.ParallelQuery<TSource>
    The sequence to return an element from.

predicate
    Type: System.Func<TSource, Boolean>
    A function to test each element for a condition.
**Type Parameters**

**TSource**

The type of the elements of source.

**Return Value**

The first element in the sequence that passes the test in the specified predicate function.
Remarks

There's a temporary difference from LINQ to Objects, this does not throw ArgumentException when the predicate is null.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>No element in source satisfies the condition in predicate.</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
First Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...FirstOrDefault Method
ParallelEnumerable Class See Also Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FirstOrDefault(Of </code>TSOURCE<code>)</code></td>
<td>Returns the first element of a parallel sequence, or a default value if the sequence contains no elements.</td>
</tr>
<tr>
<td><code>FirstOrDefault(Of </code>TSOURCE<code>, </code>function(TSOURCE, Boolean)<code>)</code></td>
<td>Returns the first element of the parallel sequence that satisfies a condition or a default value if no such element is found.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the first element of a parallel sequence, or a default value if the sequence contains no elements.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function FirstOrDefault(Of TSource) ( _
    source As ParallelQuery(Of TSource) ) _
) As TSource
```

**C#**

```csharp
public static TSource FirstOrDefault<TSource>(
    ParallelQuery<TSource> source
)
```

### Parameters

**source**

Type: [System.Linq.ParallelQuery<TSource>](Of <(TSource)>)

The sequence to return the first element of.
Type Parameters

TSource
   The type of the elements of source.

Return Value

default(TSource) if source is empty; otherwise, the first element in source.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
FirstOrDefault Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.FirstOrDefault(Of (TSource)) Method
(ParallelQuery(Of (TSource)), Func(Of (TSource, Boolean)))

ParallelEnumerable Class  See Also  Send Feedback

Returns the first element of the parallel sequence that satisfies a condition or a default value if no such element is found.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function FirstOrDefault(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _) 
) As TSource
```

**C#**

```csharp
public static TSource FirstOrDefault<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)
```

### Parameters

**source**
- Type: `System.Linq.ParallelQuery<TSource>`
- The sequence to return an element from.

**predicate**
- Type: `System.Func<TSource, Boolean>`
- A function to test each element for a condition.
Type Parameters

TSource

The type of the elements of source.

Return Value

default(TSource) if source is empty or if no element passes the test specified by predicate; otherwise, the first element in source that passes the test specified by predicate.
Remarks

There's a temporary difference from LINQ to Objects, this does not throw ArgumentException when the predicate is null.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
FirstOrDefault Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel the specified action for each element in the source.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Shared Sub ForAll(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    action As Action(Of TSource)) _
)
```

**C#**

```csharp
public static void ForAll<TSource>(
    ParallelQuery<TSource> source,
    Action<TSource> action
)
```

**Parameters**

source

Type: `System.Linq.ParallelQuery<Of<Of<TSource>>>`
The `ParallelQuery<Of<Of<TSource>>>` whose elements will be processed by action.

action

Type: `System::Action<Of<Of<TSource>>>`
An Action to invoke on each element.
Type Parameters

TSource
   The type of elements of source.
Remarks

This is an efficient way to process the output from a parallelized query because it does not require a merge step at the end. However, order of execution is non-deterministic.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or action is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::GroupBy Method

ParallelEnumerable Class  See Also  Send Feedback
# Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupBy&lt;(Of &lt;(TSource, TKey)&gt;)(ParallelQuery&lt;(Of &lt;(TSource)&gt;)), Func&lt;(Of &lt;(TSource, TKey)&gt;)&gt;</td>
<td>Groups in parallel the elements of a sequence according to a specified key selector function.</td>
</tr>
<tr>
<td>GroupBy&lt;(Of &lt;(TSource, TKey)&gt;)(ParallelQuery&lt;(Of &lt;(TSource)&gt;)), Func&lt;(Of &lt;(TSource, TKey)&gt;)&gt;, IEqualityComparer&lt;Of &lt;(TKey)&gt;&gt;</td>
<td>Groups in parallel the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.</td>
</tr>
<tr>
<td>GroupBy&lt;(Of &lt;(TSource, TKey, TElement)&gt;)(ParallelQuery&lt;(Of &lt;(TSource)&gt;)), Func&lt;(Of &lt;(TSource, TKey, TElement)&gt;)&gt;</td>
<td>Groups in parallel the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.</td>
</tr>
<tr>
<td>GroupBy&lt;(Of &lt;(TSource, TKey, TResult)&gt;)&lt;TResult&gt;(ParallelQuery&lt;(Of &lt;(TSource)&gt;)), Func&lt;(Of &lt;(TSource, TKey, TResult)&gt;)&gt;, Func&lt;(Of &lt;(TSource, TKey, TResult)&gt;)&gt;, IEqualityComparer&lt;Of &lt;(TKey)&gt;&gt;</td>
<td>Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.</td>
</tr>
<tr>
<td>GroupBy&lt;(Of &lt;(TSource, TKey, TElement)&gt;)&lt;TResult&gt;(ParallelQuery&lt;(Of &lt;(TSource)&gt;)), Func&lt;(Of &lt;(TSource, TKey, TElement)&gt;)&gt;, Func&lt;(Of &lt;(TSource, TKey, TElement)&gt;)&gt;, IEqualityComparer&lt;Of &lt;(TKey)&gt;&gt;</td>
<td>Groups in parallel the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.</td>
</tr>
</tbody>
</table>
| GroupBy<(Of <(TSource, TKey, TElement, TResult)>)<TResult>(ParallelQuery<(Of <(TSource)>)), Func<(Of <(TSource, TKey, TElement, TResult)>)> | Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and
Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Groups in parallel the elements of a sequence according to a specified key selector function.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function GroupBy(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As ParallelQuery(Of IGrouping(Of TKey, TSource))

C#

public static ParallelQuery<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of >(Of )
    An OrderedParallelQuery{TSource} than contains elements to sort.

keySelector
    Type: System(Func(Of )
    A function to extract a key from an element.
Type Parameters

TSource
The type of elements of source.

TKey
The type of the key returned by keySelector.

Return Value

An OrderedParallelQuery{TSource} whose elements are sorted descending according to a key.
See Also

ParallelEnumerable Class
GroupBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Groups in parallel the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function GroupBy(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    comparer As IEquivalenceComparer(Of TKey) _
) As ParallelQuery(Of IGrouping(Of TKey, TSource))

C#

public static ParallelQuery<IGrouping<TKey, TSource>> GroupBy<TSource>
(ParallelQuery<TSource> source,
 Func<TSource, TKey> keySelector,
 IEquivalenceComparer<TKey> comparer
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)
    An OrderedParallelQuery{TSource} than contains elements to sort.

keySelector
    Type: System.Func(Of TSource, TKey)
    A function to extract a key from an element.

comparer
    Type: System.Collections.Generic.IEqualityComparer<TKey>
    An IComparer{TSource} to compare keys.
Type Parameters

TSource
The type of elements of source.

TKey
The type of the key returned by keySelector>.

Return Value

An OrderedParallelQuery{TSource} whose elements are sorted descending according to a key.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Groups in parallel the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.

**Namespace:**  System.Linq
**Assembly:**  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function GroupBy(Of TSource, TKey, TElement) ( _
   source As ParallelQuery(Of TSource), _
   keySelector As Func(Of TSource, TKey), _
   elementSelector As Func(Of TSource, TElement) _
) As ParallelQuery(Of IGrouping(Of TKey, TElement))
```

### C#

```csharp
public static ParallelQuery<IGrouping<TKey, TElement>> GroupBy<TSource, TKey, TElement>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector
)
```

## Parameters

### source

Type: `System.Linq.ParallelQuery<Of <(TSource)>)`
An OrderedParallelQuery<Of <(TElement)>>) than contains elements to sort.

### keySelector

Type: `System.Func<Of <(TSource, TKey)>>)`
A function to extract a key from an element.

### elementSelector

Type: `System.Func<Of <(TSource, TElement)>>)`
A function to map each source element to an element in an IGrouping.
**Type Parameters**

**TSource**
  The type of elements of source.

**TKey**
  The type of the key returned by keySelector.

**TElement**
  The type of the elements in the IGrouping

**Return Value**

A ParallelQuery<IGrouping<TKey, TElement>> in C# or ParallelQuery(Of IGrouping(Of TKey, TElement)) in Visual Basic where each IGrouping generic object contains a collection of objects of type TElement and a key.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector or elementSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function GroupBy(Of TSource, TKey, TResult) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    resultSelector As Func(Of TKey, IEnumerable(Of TSource), TResult) ) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> GroupBy<TSource, TKey, TResult>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TKey, IEnumerable<TSource>, TResult> resultSelector
)

Parameters

source
Type: System.Linq.ParallelQuery(Of (Of (TSource)>)
A sequence whose elements to group.

keySelector
Type: System.Func(Of (Of (TSource, TKey)>)
A function to extract the key for each element.

resultSelector
Type: System.Func(Of (Of (TKey, IEnumerable(Of (Of (TSource)>)>, TResult>)
A function to create a result value from each group.
Type Parameters

TSource
   The type of the elements of source.
TKey
   The type of the key returned by keySelector.
TResult
   The type of the result value returned by resultSelector.

Return Value

A collection of elements of type TResult where each element represents a projection over a group and its key.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or keySelector or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Groups in parallel the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function GroupBy(Of TSource, TKey, TElement) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement), _
    comparer As IEqualityComparer(Of TKey) _
) As ParallelQuery(Of IGrouping(Of TKey, TElement))

C#

public static ParallelQuery<IGrouping<TKey, TElement>> GroupBy<TSource>
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector,
    IEqualityComparer<TKey> comparer

Parameters

source
    Type: System.Linq.::;ParallelQuery<(Of <(TSource)>)>)
    An OrderedParallelQuery{TSource} than contains elements to sort.

keySelector
    Type: System.::.Func<(Of <(TSource, TKey)>)>)
    A function to extract a key from an element.

elementSelector
    Type: System.::.Func<(Of <(TSource, TElement)>)>)
    A function to map each source element to an element in an IGrouping.

comparer
    Type: System.Collections.Generic.::.IEqualityComparer<(Of <(TKey)>)>)
    An IComparer{TSource} to compare keys.
**Type Parameters**

**TSource**
The type of elements of source.

**TKey**
The type of the key returned by keySelector.

**TElement**
The type of the elements in the IGrouping

**Return Value**

A ParallelQuery{IGrouping{TKey, TElement}} in C# or ParallelQuery(Of IGrouping(Of TKey, TElement)) in Visual Basic where each IGrouping generic object contains a collection of objects of type TElement and a key.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..ArgumentNullException</td>
<td>source or keySelector or elementSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function GroupBy(Of TSource, TKey, TElement, TResult) source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement), _
    resultSelector As Func(Of TKey, IEnumerable(Of TElement), TResult) _
) As ParallelQuery(Of TResult)
```

**C#**

```csharp
public static ParallelQuery<TResult> GroupBy<TSource, TKey, TElement>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector,
    Func<TKey, IEnumerable<TElement>, TResult> resultSelector
)
```

### Parameters

**source**
- Type: System.Linq::ParallelQuery<(Of <(TSource)>)>)
- A sequence whose elements to group.

**keySelector**
- Type: System::Func<(Of <(TSource, TKey)>)>
- A function to extract the key for each element.

**elementSelector**
- Type: System::Func<(Of <(TSource, TElement)>)>
- A function to map each source element to an element in an IGrouping<TKey, TElement>.

**resultSelector**
- Type: System::Func<(Of <(TKey, IEnumerable<Of <(TElement)>)>, TResult)>)
A function to create a result value from each group.
Type Parameters

TSource
The type of the elements of source.

TKey
The type of the key returned by keySelector.

TElement
The type of the elements in each IGrouping{TKey, TElement}.

TResult
The type of the result value returned by resultSelector.

Return Value

A collection of elements of type TElement where each element represents a projection over a group and its key.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgumentNullException</td>
<td>source or keySelector or elementSelector or System..::.ArgumentNullException resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.GroupBy(Of TSource, TKey, TResult)> Method (ParallelQuery(Of TSource>>, Func(Of (TSource, TKey)>), Func(Of (TKey, IEnumerable(Of TSource)>), TResult>), IEqualityComparer(Of TKey>>) Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer. 

Syntax

Visual Basic (Declaration)

Public Shared Function GroupBy(Of TSource, TKey, TResult) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    resultSelector As Func(Of TKey, IEnumerable(Of TSource), TResult), _
    comparer As IEqualityComparer(Of TKey) _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> GroupBy<TSource, TKey, TResult>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TKey, IEnumerable<TSource>, TResult> resultSelector,
    IEqualityComparer<TKey> comparer
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of (TSource)>))
    A sequence whose elements to group.

keySelector
    Type: System.Func(Of (Of (TSource, TKey)>))
    A function to extract the key for each element.

resultSelector
    Type: System.Func(Of (Of (TKey, IEnumerable(Of (Of (TSource)>),
                                  TResult>))>
    A function to create a result value from each group.

comparer
    Type: System.Collections.Generic.IEqualityComparer(Of TKey)
    An IEqualityComparer{TKey} to compare keys.
Type Parameters

TSource
  The type of the elements of source.
TKey
  The type of the key returned by keySelector.
TResult
  The type of the result value returned by resultSelector.

Return Value

An ParallelQuery<IGrouping<TKey, TResult>> in C# or ParallelQuery(Of IGrouping(Of TKey, TResult)) in Visual Basic where each IGrouping<Of <(TKey, TResult)>>) object contains a collection of objects of type TResult and a key.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:ArgumentNullException</td>
<td>source or keySelector or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function GroupBy(Of TSource, TKey, TElement, TResult)  
    source As ParallelQuery(Of TSource), _  
    keySelector As Func(Of TSource, TKey), _  
    elementSelector As Func(Of TSource, TElement), _  
    resultSelector As Func(Of TKey, IEnumerable(Of TElement), TResult), _  
    comparer As IEqualityComparer(Of TKey) _  
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> GroupBy<TSource, TKey, TElement>( 
    ParallelQuery<TSource> source, 
    Func<TSource, TKey> keySelector, 
    Func<TSource, TElement> elementSelector, 
    Func<TKey, IEnumerable<TElement>, TResult> resultSelector, 
    IEqualityComparer<TKey> comparer
)

Parameters

source
Type: System.Linq.:::ParallelQuery(Of <(TSource)>)
A sequence whose elements to group.

keySelector
Type: System.:::Func(Of <(TSource, TKey)>)
A function to extract the key for each element.

elementSelector
Type: System.:::Func(Of <(TSource, TElement)>)
A function to map each source element to an element in an IGrouping{Key, TElement}.

resultSelector
Type: System.:::Func(Of <(TKey, IEnumerable<Of <(TElement)>)>,
TResult>)>
A function to create a result value from each group.

comparer
Type: System.Collections.Generic.IEqualityComparer(Of TKey)
An IEqualityComparer{TKey} to compare keys.
Type Parameters

TSource
   The type of the elements of source.
TKey
   The type of the key returned by keySelector.
TElement
   The type of the elements in each IGrouping{TKey, TElement}.
TResult
   The type of the result value returned by resultSelector.

Return Value

A collection of elements of type TResult where each element represents a projection over a group and its key.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgumentNullException</td>
<td>source or keySelector or elementSelector or System..::.ArgumentNullException resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:::GroupJoin Method

ParallelEnumerable Class  See Also  Send Feedback
Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| GroupJoin<(Of <(TOuter, TInner, TKey, TResult)>=)>(ParallelQuery<(Of <(TOuter)>), IEnumerable<(Of <(TInner)>), Func<(Of <(TOuter, TKey)>), Func<(Of <(TInner, TKey)>), Func<(Of <(TOuter, IEnumerable<(Of <(TInner)>), TKey, TResult)>))>

Correlates in parallel the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.

Obsoletes.
This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

GroupJoin<(Of <(TOuter, TInner, TKey, TResult)>=)>(ParallelQuery<(Of <(TOuter)>), IEnumerable<(Of <(TInner)>), Func<(Of <(TOuter, TKey)>), Func<(Of <(TInner, TKey)>), Func<(Of <(TOuter, IEnumerable<(Of <(TInner)>), TKey, TResult)>)), IEqualityComparer<(Of <(TKey)>))>

Correlates in parallel the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer{T} is used to compare keys.

Obsoletes.
This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#

.NET Framework Class Library

ParallelEnumerable...:::GroupJoin(Of <(TOuter, TInner, TKey, TResult)>)
Method (ParallelQuery(Of <(TOuter)>), IEnumerable(Of <(TInner)>), Func(Of <(TOuter, TKey)>), Func(Of <(TInner, TKey)>), Func(Of <(TOuter, IEnumerable(Of <(TInner)>), TResult)>))

ParallelEnumerable Class  See Also  Send Feedback

This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
## Syntax

```csharp

Public Shared Function GroupJoin(Of TOuter, TInner, TKey, TResult) (outer As ParallelQuery(Of TOuter), _
inner As IEnumerable(Of TInner), _
outerKeySelector As Func(Of TOuter, TKey), _
innerKeySelector As Func(Of TInner, TKey), _
resultSelector As Func(Of TOuter, IEnumerable(Of TInner), TResult)) As ParallelQuery(Of TResult)

public static ParallelQuery<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(
    ParallelQuery<TOuter> outer,
    IEnumerable<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, IEnumerable<TInner>, TResult> resultSelector
)
```

### Parameters

**outer**
- Type: `System.Linq.ParallelQuery(Of TOuter)`
- This parameter is not used.

**inner**
- Type: `System.Collections.Generic.IEnumerable(Of TInner)`
- This parameter is not used.

**outerKeySelector**
- Type: `System.Func(Of TOuter, TKey)`
- This parameter is not used.

**innerKeySelector**
- Type: `System.Func(Of TInner, TKey)`
- This parameter is not used.
Type: System:::Func<(Of<(TInner, TKey)>)>  
This parameter is not used.

resultSelector
Type: System:::Func<(Of<(TOuter, IEnumerable<(Of<(TInner)>)), TResult)>)>  
This parameter is not used.
Type Parameters

TOuter
   This type parameter is not used.
TInner
   This type parameter is not used.
TKey
   This type parameter is not used.
TResult
   This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of GroupJoin with a left data source of type `ParallelQuery<Of <(TSource)>>` and a right data source of type `IEnumerable<Of <(T)>>`. Otherwise, the GroupJoin operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

- ParallelEnumerable Class
- GroupJoin Overload
- System.Linq Namespace

Send feedback on this topic to Microsoft.
Correlates in parallel the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function GroupJoin(Of TOuter, TInner, TKey, TResult) (outer As ParallelQuery(Of TOuter), _
inner As ParallelQuery(Of TInner), _
outerKeySelector As Func(Of TOuter, TKey), _
innerKeySelector As Func(Of TInner, TKey), _
resultSelector As Func(Of TOuter, IEnumerable(Of TInner), TResult)) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(
    ParallelQuery<TOuter> outer,
    ParallelQuery<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, IEnumerable<TInner>, TResult> resultSelector
)

Parameters

outer
Type: System.Linq.::.ParallelQuery<(Of (Of(TOuter)>))
The first sequence to join.

inner
Type: System.Linq.::.ParallelQuery<(Of (Of(TInner)>))
The sequence to join to the first sequence.

outerKeySelector
Type: System.::.Func<(Of (Of(TOuter, TKey)>))
A function to extract the join key from each element of the first sequence.

innerKeySelector
Type: System.::.Func<(Of (Of(TInner, TKey)>))
A function to extract the join key from each element of the second
sequence.

resultSelector
Type: System::Func<(Of <(TOuter, IEnumerable<(Of <(TInner)>)), TResult)>)
A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.
Type Parameters

TOuter
   The type of the elements of the first sequence.
TInner
   The type of the elements of the second sequence.
TKey
   The type of the keys returned by the key selector functions.
TResult
   The type of the result elements.

Return Value

A sequence that has elements of type TResult that are obtained by performing a grouped join on two sequences.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgumentNullException</td>
<td>outer or inner or outerKeySelector or System..::.ArgumentException innerKeySelector or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupJoin Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

**Namespace:**  System.Linq
**Assembly:**  System.Threading (in System.Threading.dll)
# Syntax

```csharp
<ObsoleteAttribute("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Linq.Query<T>.")>

Public Shared Function GroupJoin(Of TOuter, TInner, TKey, TResult) (outer As ParallelQuery(Of TOuter),
    inner As IEnumerable(Of TInner),
    outerKeySelector As Func(Of TOuter, TKey),
    innerKeySelector As Func(Of TInner, TKey),
    resultSelector As Func(Of TOuter, IEnumerable(Of TInner), TResult),
    comparer As IEqualityComparer(Of TKey))
As ParallelQuery(Of TResult)
```


```csharp
public static ParallelQuery<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(
    ParallelQuery<TOuter> outer,
    IEnumerable<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, IEnumerable<TInner>, TResult> resultSelector,
    IEqualityComparer<TKey> comparer
)
```

## Parameters

**outer**
- Type: `System.Linq.ParallelQuery(OfOf(TOuter))`
- This parameter is not used.

**inner**
- Type: `System.Collections.Generic.IEnumerable(OfOf(TInner))`
- This parameter is not used.

**outerKeySelector**
- Type: `System.Func(OfOf(TOuter, TKey))`
- This parameter is not used.
innerKeySelector
   Type: System.Func(Of (TInner, TKey))
   This parameter is not used.

resultSelector
   Type: System.Func(Of (TOuter, IEnumerable(Of (TInner))>, TResult)
   This parameter is not used.

comparer
   Type: System.Collections.Generic.IEqualityComparer(Of<TKey>)
   This parameter is not used.
**Type Parameters**

TOuter
   This type parameter is not used.
TInner
   This type parameter is not used.
TKey
   This type parameter is not used.
TResult
   This type parameter is not used.

**ReturnValue**

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of GroupJoin with a left data source of type \texttt{ParallelQuery\langle Of <(TSource)>\rangle} and a right data source of type \texttt{IEnumerable\langle Of <(T)>\rangle}. Otherwise, the GroupJoin operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System....::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
GroupJoin Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Correlates in parallel the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer{T} is used to compare keys.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function GroupJoin(Of TOuter, TInner, TKey, TResult) (outer As ParallelQuery(Of TOuter), _
inner As ParallelQuery(Of TInner), _
outerKeySelector As Func(Of TOuter, TKey), _
innerKeySelector As Func(Of TInner, TKey), _
resultSelector As Func(Of TOuter, IEnumerable(Of TInner), TResult), comparer As IEqualityComparer(Of TKey) _) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> GroupJoin<TOuter, TInner, TKey, 
ParallelQuery<TOuter> outer,
ParallelQuery<TInner> inner,
Func<TOuter, TKey> outerKeySelector,
Func<TInner, TKey> innerKeySelector,
Func<TOuter, IEnumerable<TInner>, TResult> resultSelector,
IEqualityComparer<TKey> comparer
)

Parameters

outer
Type: System.Linq.::.ParallelQuery<(Of <(TOuter)>))
The first sequence to join.

inner
Type: System.Linq.::.ParallelQuery<(Of <(TInner)>))
The sequence to join to the first sequence.

outerKeySelector
Type: System.::.Func<(Of <(TOuter, TKey)>))
A function to extract the join key from each element of the first sequence.

innerKeySelector
Type: System:::Func<(Of<br>(Of<br>(TInner, TKey)>)><br>A function to extract the join key from each element of the second sequence.

resultSelector<br>Type: System:::Func<(Of<br>(Of<br>(TOuter, IEnumerable(Of<br>(TInner)>),<br>TResult>)>)<br>A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

comparer<br>Type: System.Collections.Generic:::IEqualityComparer<(Of<br>(Of<br>(TKey)>)><br>An IEqualityComparer<(Of<br>(T)>)> to hash and compare keys.
Type Parameters

TOuter
   The type of the elements of the first sequence.
TInner
   The type of the elements of the second sequence.
TKey
   The type of the keys returned by the key selector functions.
TResult
   The type of the result elements.

Return Value

A sequence that has elements of type TResult that are obtained by performing a grouped join on two sequences.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgumentNullException</td>
<td>outer or inner or outerKeySelector or System..::.ArgumentNullException innerKeySelector or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

- ParallelEnumerable Class
- GroupJoin Overload
- System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Intersect Method

ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Overload List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>Intersect&lt;Of (Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;Of &lt;(TSource)&gt;&gt;, IEnumerable&lt;Of &lt;(TSource)&gt;&gt;)</td>
</tr>
<tr>
<td>Intersect&lt;Of &lt;(TSource)&gt;&gt; (ParallelQuery&lt;Of &lt;(TSource)&gt;&gt;, ParallelQuery&lt;Of &lt;(TSource)&gt;&gt;)</td>
</tr>
<tr>
<td>Intersect&lt;Of &lt;(TSource)&gt;&gt; (ParallelQuery&lt;Of &lt;(TSource)&gt;&gt;, IEnumerable&lt;Of &lt;(TSource)&gt;&gt;, IEqualityComparer&lt;Of &lt;(TSource)&gt;&gt;)</td>
</tr>
<tr>
<td>Intersect&lt;Of &lt;(TSource)&gt;&gt; (ParallelQuery&lt;Of &lt;(TSource)&gt;&gt;, ParallelQuery&lt;Of &lt;(TSource)&gt;&gt;, IEqualityComparer&lt;Of &lt;(TSource)&gt;&gt;)</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

```csharp
[ObsoleteAttribute("The second data source of a binary operator must
must
must
Linq.ParallelQuery<T>.")] Public Shared Function Intersect(Of TSource) ( first As ParallelQuery(Of TSource), second As IEnumerable(Of TSource) ) As ParallelQuery(Of TSource)

[ObsoleteAttribute("The second data source of a binary operator must
must
must
Linq.ParallelQuery<T>.")] public static ParallelQuery<TSource> Intersect<TSource>( ParallelQuery<TSource> first, IEnumerable<TSource> second )

Parameters

first
Type: System.Linq..::.ParallelQuery<(Of <(TSource)>)> This parameter is not used.

second
Type: System.Collections.Generic..::.IEnumerable<(Of <(TSource)>)> This parameter is not used.
**Type Parameters**

**TSource**

This type parameter is not used.

**Return Value**

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Intersect with a left data source of type `ParallelQuery<(Of<(TSource)>)>` and a right data source of type `IEnumerable<(Of<(T)>)>`. Otherwise, the Intersect operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Intersect Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Intersect\((TSource)\) Method
(ParallelQuery\((TSource)\), ParallelQuery\((TSource)\))

ParallelEnumerable Class  See Also  Send Feedback

Produces the set intersection of two parallel sequences by using the default
equality comparer to compare values.

**Namespace:**  System.Linq
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Intersect(Of TSource) ( _
    first As ParallelQuery(Of TSource), _
    second As ParallelQuery(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Intersect<TSource>(
    ParallelQuery<TSource> first,
    ParallelQuery<TSource> second
)

Parameters

first
    Type: System.Linq.Enumerable.ParallelQuery(Of (Of TSource))
    A sequence whose distinct elements that also appear in second will be returned.

second
    Type: System.Linq.Enumerable.ParallelQuery(Of (Of TSource))
    A sequence whose distinct elements that also appear in the first sequence will be returned.
Type Parameters

TSource
   The type of the elements of the input sequences.

Return Value

A sequence that contains the elements that form the set intersection of two sequences.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Intersect Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedExceptio when called.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)

Public Shared Function Intersect(Of TSource) ( _
    first As ParallelQuery(Of TSource), _
    second As IEnumerable(Of TSource), _
    comparer As IEqualityComparer(Of TSource) _
) As ParallelQuery(Of TSource)


public static ParallelQuery<TSource> Intersect<TSource>(
    ParallelQuery<TSource> first,
    IEnumerable<TSource> second,
    IEqualityComparer<TSource> comparer
)

Parameters

first
    Type: System.Linq...::ParallelQuery<(Of <(TSource)>))
    This parameter is not used.

second
    Type: System.Collections.Generic...::IEnumerable<(Of <(TSource)>))
    This parameter is not used.

comparer
    Type: System.Collections.Generic...::IEqualityComparer<(Of <(TSource)>))
    This parameter is not used.
Type Parameters

TSource
   This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Intersect with a left data source of type `ParallelQuery<Of <(TSource)>>>` and a right data source of type `IEnumerable<Of <(T)>>`. Otherwise, the Intersect operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.NotSupportedException</code></td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Intersect Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
ParallelEnumerable::Intersect(Of TSource>) Method
(ParallelQuery(Of TSource>), ParallelQuery(Of TSource>),
IEqualityComparer(Of TSource>))

ParallelEnumerable Class  See Also  Send Feedback

Produces the set intersection of two parallel sequences by using the specified
IEqualityComparer{T} to compare values.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Intersect(Of TSource) ( _
    first As ParallelQuery(Of TSource), _
    second As ParallelQuery(Of TSource), _
    comparer As IEqualityComparer(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery&lt;TSource&gt; Intersect&lt;TSource&gt;(ParallelQuery&lt;TSource&gt; first,
    ParallelQuery&lt;TSource&gt; second,
    IEqualityComparer&lt;TSource&gt; comparer

Parameters

first
    Type: System.Linq::.ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;
    A sequence whose distinct elements that also appear in second will be returned.

second
    Type: System.Linq::.ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;
    A sequence whose distinct elements that also appear in the first sequence will be returned.

comparer
    Type: System.Collections.Generic::.IEqualityComparer&lt;(Of &lt;(T)&gt;)&gt;
    An IEqualityComparer&lt;(Of &lt;(T)&gt;)&gt; to compare values.
**Type Parameters**

TSource

The type of the elements of the input sequences.

**Return Value**

A sequence that contains the elements that form the set intersection of two sequences.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Intersect Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable.Join Method
ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join&lt;(Of &lt;(TOuter, TInner, TKey, TResult)&gt;)(ParallelQuery&lt;(Of &lt;(TOuter)&gt;), IEnumerable&lt;(Of &lt;(TInner)&gt;), Func&lt;(Of &lt;(TOuter, TKey)&gt;), Func&lt;(Of &lt;(TInner, TKey)&gt;), Func&lt;(Of &lt;(TOuter, TInner, TResult)&gt;)&gt;))</td>
<td><strong>Obsolete.</strong> This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked. Correlates in parallel the elements of two sequences based on matching keys. A specified IEqualityComparer{T} is used to compare keys.</td>
</tr>
<tr>
<td>Join&lt;(Of &lt;(TOuter, TInner, TKey, TResult)&gt;)(ParallelQuery&lt;(Of &lt;(TOuter)&gt;), ParallelQuery&lt;(Of &lt;(TInner)&gt;), Func&lt;(Of &lt;(TOuter, TKey)&gt;), Func&lt;(Of &lt;(TInner, TKey)&gt;), Func&lt;(Of &lt;(TOuter, TInner, TResult)&gt;), IEqualityComparer&lt;(Of &lt;(TKey)&gt;)&gt;))</td>
<td><strong>Obsolete.</strong> This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked. Correlates in parallel the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.</td>
</tr>
<tr>
<td>Join&lt;(Of &lt;(TOuter, TInner, TKey, TResult)&gt;)(ParallelQuery&lt;(Of &lt;(TOuter)&gt;), IEnumerable&lt;(Of &lt;(TInner)&gt;), Func&lt;(Of &lt;(TOuter, TKey)&gt;), Func&lt;(Of &lt;(TInner, TKey)&gt;), Func&lt;(Of &lt;(TOuter, TInner, TResult)&gt;), IEqualityComparerComparer&lt;(Of &lt;(TKey)&gt;)&gt;))</td>
<td><strong>Obsolete.</strong> This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked. Correlates in parallel the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Join<(Of <(TOuter, TInner, TKey, TResult)>)> Method
(ParallelQuery<(Of <(TOuter)>), IEnumerable<(Of <(TInner)>)), Func<(Of <(TOuter, TKey)>), Func<(Of <(TInner, TKey)>), Func<(Of <(TOuter, TInner, TResult)>))

ParallelEnumerable Class  See Also  Send Feedback

This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

```csharp

public static ParallelQuery<TResult> Join<TOuter, TInner, TKey, TResult>(
    ParallelQuery<TOuter> outer,
    IEnumerable<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, TInner, TResult> resultSelector)
```

#### Parameters

- **outer**
  - Type: `System.Linq.Enumerable.ParallelQuery<TOuter>(Of (TOuter)>
  - This parameter is not used.

- **inner**
  - Type: `System.Collections.Generic.Enumerable.IEnumerable<TInner>(Of (TInner)>
  - This parameter is not used.

- **outerKeySelector**
  - Type: `System.Enumerable.Func<TOuter, TKey>(Of ((TOuter, TKey)>
  - This parameter is not used.

- **innerKeySelector**
  - Type: `System.Enumerable.Func<TInner, TKey>(Of ((TInner, TKey)>
  - This parameter is not used.
Type: System:::Func<(Of <(TInner, TKey)>)>  
This parameter is not used.

resultSelector
Type: System:::Func<(Of <(TOuter, TInner, TResult)>)>  
This parameter is not used.
Type Parameters

TOuter
   This type parameter is not used.
TInner
   This type parameter is not used.
TKey
   This type parameter is not used.
TResult
   This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage Join with a left data source of type `ParallelQuery<Of <(TSource)<>>` and a right data source of type `IEnumerable<Of <(T)>>`.

Otherwise, the Join operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Join Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Correlates in parallel the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Join(Of TOuter, TInner, TKey, TResult) ( _
    outer As ParallelQuery(Of TOuter), _
    inner As ParallelQuery(Of TInner), _
    outerKeySelector As Func(Of TOuter, TKey), _
    innerKeySelector As Func(Of TInner, TKey), _
    resultSelector As Func(Of TOuter, TInner, TResult) _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> Join<TOuter, TInner, TKey, TResult>(
    ParallelQuery<TOuter> outer,
    ParallelQuery<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, TInner, TResult> resultSelector
)

Parameters

outer
    Type: System.Linq.ParallelQuery(Of <(TOuter)>)
    The first sequence to join.

inner
    Type: System.Linq.ParallelQuery(Of <(TInner)>)
    The sequence to join to the first sequence.

outerKeySelector
    Type: System.Func(Of <(TOuter, TKey)>)
    A function to extract the join key from each element of the first sequence.

innerKeySelector
    Type: System.Func(Of <(TInner, TKey)>)
    A function to extract the join key from each element of the second
resultSelector

Type: System::Func<(Of <(TOuter, TInner, TResult)>)>

A function to create a result element from two matching elements.
Type Parameters

TOuter
The type of the elements of the first sequence.

TInner
The type of the elements of the second sequence.

TKey
The type of the keys returned by the key selector functions.

TResult
The type of the result elements.

Return Value

A sequence that has elements of type TResult that are obtained by performing an inner join on two sequences.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>outer or inner or outerKeySelector or System..::.ArgumentNullException innerKeySelector or resultSelector</td>
<td>is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Join Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.Join(Of (TOuter, TInner, TKey, TResult)> Method
(ParallelQuery(Of (TOuter)>), IEnumerable(Of (TInner)>), Func(Of (TOuter, TKey)>), Func(Of (TInner, TKey)>), Func(Of (TOuter, TInner, TResult)>), IEqualityComparer(Of (TKey)>))

This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

<ObsoleteAttribute("The second data source of a binary operator must
Public Shared Function Join(Of TOuter, TInner, TKey, TResult) ( _
    outer As ParallelQuery(Of TOuter), _
    inner As IEnumerable(Of TInner), _
    outerKeySelector As Func(Of TOuter, TKey), _
    innerKeySelector As Func(Of TInner, TKey), _
    resultSelector As Func(Of TOuter, TInner, TResult), _
    comparer As IEqualityComparer(Of TKey) _
) As ParallelQuery(Of TResult)

[ObsoleteAttribute("The second data source of a binary operator must
public static ParallelQuery<TResult> Join<TOuter, TInner, TKey, TRes:
    ParallelQuery<TOuter> outer,
    IEnumerable<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, TInner, TResult> resultSelector,
    IEqualityComparer<TKey> comparer
)

Parameters

outer
    Type: System.Linq.ParallelQuery(Of (Of (TOuter)))
    This parameter is not used.

inner
    Type: System.Collections.Generic.IEnumerable(Of (Of (TInner)))
    This parameter is not used.

outerKeySelector
    Type: System.Func(Of (Of (TOuter, TKey)))
    This parameter is not used.
innerKeySelector
   Type: System::Func<(Of <(TInner, TKey)>)
   This parameter is not used.

resultSelector
   Type: System::Func<(Of <(TOuter, TInner, TResult)>)
   This parameter is not used.

comparer
   Type: System.Collections.Generic::IEqualityComparer<(Of <(TKey)>)
   This parameter is not used.
### Type Parameters

TOuter
   This type parameter is not used.
TInner
   This type parameter is not used.
TKey
   This type parameter is not used.
TResult
   This type parameter is not used.

### Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Join with a left data source of type `ParallelQuery<Of <(TSource)>)>` and a right data source of type `IEnumerable<Of <(T)>)>`. Otherwise, the Join operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System....::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Join Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Join(Of TOuter, TInner, TKey, TResult) Method (ParallelQuery(Of TOuter), ParallelQuery(Of TInner), Func(Of TOuter, TKey, TResult), Func(Of TInner, TKey, TResult), Func(Of TOuter, TInner, TResult), IEqualityComparer<TKey>)

Correlates in parallel the elements of two sequences based on matching keys. A specified IEqualityComparer{T} is used to compare keys.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Join(Of TOuter, TInner, TKey, TResult) ( _
outer As ParallelQuery(Of TOuter), _
inner As ParallelQuery(Of TInner), _
outerKeySelector As Func(Of TOuter, TKey), _
innerKeySelector As Func(Of TInner, TKey), _
resultSelector As Func(Of TOuter, TInner, TResult), _
comparer As IEqualityComparer(Of TKey) _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> Join<TOuter, TInner, TKey, TResult>(
    ParallelQuery<TOuter> outer,
    ParallelQuery<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, TInner, TResult> resultSelector,
    IEqualityComparer<TKey> comparer
)

Parameters

outer
Type: System.Linq.ParallelQuery(Of (Of (Of TOuter))>)
The first sequence to join.

inner
Type: System.Linq.ParallelQuery(Of (Of (Of TInner))>)
The sequence to join to the first sequence.

outerKeySelector
Type: System::.Func(Of (Of (Of TOuter, TKey))>)
A function to extract the join key from each element of the first sequence.

innerKeySelector
Type: System..::.Func<Of<(TInner, TKey)>>
A function to extract the join key from each element of the second sequence.

resultSelector
Type: System..::.Func<Of<(TOuter, TInner, TResult)>>
A function to create a result element from two matching elements.

comparer
Type: System.Collections.Generic..::.IEqualityComparer<Of<(TKey)>>
An IEqualityComparer<Of<(T)>> to hash and compare keys.
Type Parameters

TOuter
    The type of the elements of the first sequence.
TInner
    The type of the elements of the second sequence.
TKey
    The type of the keys returned by the key selector functions.
TResult
    The type of the result elements.

Return Value

A sequence that has elements of type TResult that are obtained by performing an inner join on two sequences.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>outer or inner or outerKeySelector or innerKeySelector or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Join Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable Class
# Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;), Last&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TSource, Boolean)&gt;)&gt;))</td>
<td>Returns the last element of a parallel sequence. Returns the last element of a parallel sequence that satisfies a specified condition.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the last element of a parallel sequence.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Last(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As TSource

C#

public static TSource Last<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<TSource>
    The sequence to return the last element from.
**Type Parameters**

**TSource**

The type of the elements of source.

**Return Value**

The value at the last position in the source sequence.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Last Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the last element of a parallel sequence that satisfies a specified condition.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Last(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As TSource

C#

public static TSource Last<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
    Type: System.Linq.:::ParallelQuery<(Of <(TSource)>))
    The sequence to return an element from.

predicate
    Type: System.:::Func<(Of <(TSource, Boolean)>))
    A function to test each element for a condition.
Type Parameters

TSource
  The type of the elements of source.

Return Value

The last element in the sequence that passes the test in the specified predicate function.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>No element in source satisfies the condition in predicate.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Last Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LastOrDefault&lt;TSource&gt;()(ParallelQuery&lt;TSource&gt;())</code></td>
<td>Returns the last element of a parallel sequence, or a default value if the sequence contains no elements.</td>
</tr>
<tr>
<td><code>LastOrDefault&lt;TSource&gt;()(ParallelQuery&lt;TSource&gt;(), Func&lt;TSource, Boolean&gt;())</code></td>
<td>Returns the last element of a parallel sequence that satisfies a condition, or a default value if no such element is found.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.LastOrDefault(Of TSource) Method (ParallelQuery(Of TSource))

ParallelEnumerable Class  See Also  Send Feedback

Returns the last element of a parallel sequence, or a default value if the sequence contains no elements.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function LastOrDefault(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As TSource

C#

public static TSource LastOrDefault<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of T<TSource>)
The sequence to return an element from.
Type Parameters

TSource

The type of the elements of source.

Return Value

default(TSource) if the source sequence is empty; otherwise, the last element in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
LastOrDefault Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the last element of a parallel sequence that satisfies a condition, or a default value if no such element is found.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function LastOrDefault(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As TSource

C#

public static TSource LastOrDefault<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of(Of(Of(TSource)>))>
    The sequence to return an element from.

predicate
    Type: System::.Func(Of(Of(TSource, Boolean)>)>)
    A function to test each element for a condition.
### Type Parameters

**TSource**
The type of the elements of source.

### Return Value

default(TSource) if the sequence is empty or if no elements pass the test in the predicate function; otherwise, the last element that passes the test in the predicate function.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
LastOrDefault Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.LongCount Method

ParallelEnumerable Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LongCount&lt;Of &lt;(TSource)&gt;&gt;</code> (ParallelQuery&lt;Of &lt;(TSource)&gt;&gt;)</td>
<td>Returns an Int64 that represents the total number of elements in a parallel sequence.</td>
</tr>
<tr>
<td><code>LongCount&lt;Of &lt;(TSource)&gt;&gt;</code> (ParallelQuery&lt;Of &lt;(TSource)&gt;&gt;, Func&lt;Of &lt;(TSource, Boolean)&gt;&gt;)</td>
<td>Returns an Int64 that represents how many elements in a parallel sequence satisfy a condition.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns an Int64 that represents the total number of elements in a parallel sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function LongCount(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As Long
```

#### C#

```csharp
public static long LongCount<TSource>(
    ParallelQuery<TSource> source
)
```

### Parameters

- **source**
  - Type: `System.Linq.ParallelQuery(Of (Of (TSource)>)`
  - A sequence that contains elements to be counted.
Type Parameters

TSource
   The type of the elements of source.

Return Value

The number of elements in the input sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>The number of elements in source is larger than <code>MaxValue()</code>(). -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
LongCount Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.LongCount(Of (TSource)) Method
(ParallelQuery(Of (TSource)), Func(Of (TSource, Boolean)))

ParallelEnumerable Class  See Also  Send Feedback

Returns an Int64 that represents how many elements in a parallel sequence satisfy a condition.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function LongCount(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As Long

C#

public static long LongCount<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
    Type: System.Linq...::ParallelQuery<(Of <(TSource)>>)
    A sequence that contains elements to be counted.

predicate
    Type: System...::Func<(Of <(TSource, Boolean)>>)
    A function to test each element for a condition.
Type Parameters

TSource
   The type of the elements of source.

Return Value

A number that represents how many elements in the sequence satisfy the condition in the predicate function.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>The number of elements in source is larger than MaxValue(). -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
LongCount Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:Max Method
ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Decimal} \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Double} \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Int32} \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Int64} \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Nullable} &lt; \langle \text{Decimal} \rangle \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Nullable} &lt; \langle \text{Double} \rangle \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Nullable} &lt; \langle \text{Int32} \rangle \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Nullable} &lt; \langle \text{Int64} \rangle \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Nullable} &lt; \langle \text{Single} \rangle \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max(ParallelQuery&lt;Of } \langle \text{Single} \rangle) )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max&lt;Of } \langle \text{TSource} \rangle ) ( \langle \text{ParallelQuery&lt;Of } \langle \text{TSource} \rangle \rangle )</td>
<td>Returns the maximum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>( \text{Max&lt;Of } \langle \text{TSource} \rangle ) ( \langle \text{ParallelQuery&lt;Of } \langle \text{TSource} \rangle \rangle, \ Func&lt;Of } \langle \text{TSource, Decimal} \rangle \rangle )</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the maximum value.</td>
</tr>
<tr>
<td>( \text{Max&lt;Of } \langle \text{TSource} \rangle ) ( \langle \text{ParallelQuery&lt;Of } \langle \text{TSource} \rangle \rangle, \ Func&lt;Of } \langle \text{TSource, Double} \rangle \rangle )</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the maximum value.</td>
</tr>
</tbody>
</table>
Max(Of<TSource>)(ParallelQuery(Of<TSource>), Func(Of<TSource, Int32>))

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

Max(Of<TSource>)(ParallelQuery(Of<TSource>), Func(Of<TSource, Int64>))

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

Max(Of<TSource>)(ParallelQuery(Of<TSource>), Func(Of<TSource, Nullable(Of<Decimal>>)))

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

Max(Of<TSource>)(ParallelQuery(Of<TSource>), Func(Of<TSource, Nullable(Of<Double>>)))

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

Max(Of<TSource>)(ParallelQuery(Of<TSource>), Func(Of<TSource, Nullable(Of<Int32>>)))

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

Max(Of<TSource>)(ParallelQuery(Of<TSource>), Func(Of<TSource, Nullable(Of<Int64>>)))

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

Max(Of<TSource>)(ParallelQuery(Of<TSource>), Func(Of<TSource, Nullable(Of<Single>>)))

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

Max(Of<TSource, TResult>)(ParallelQuery(Of<TSource>), Func(Of<TSource, TResult>))

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the maximum value in a parallel sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function Max ( 
    source As ParallelQuery(Of Decimal) 
) As Decimal
```

### C#

```csharp
public static decimal Max(
    ParallelQuery<decimal> source 
)
```

### Parameters

source  
Type: `System.Linq.:::ParallelQuery<Of <(Decimal)>>)`  
A sequence of values to determine the maximum value of.

### Return Value

The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the maximum value in a parallel sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max ( _
    source As ParallelQuery(Of Double) _
) As Double

C#

public static double Max(
    ParallelQuery<double> source
)

Parameters

source
Type: System.Linq.ParallelQuery(Of (Double))
A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the maximum value in a parallel sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max ( _
    source As ParallelQuery(Of Integer) _
) As Integer

C#

public static int Max(
    ParallelQuery<int> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Int32))
    A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System:::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the maximum value in a parallel sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max ( _
    source As ParallelQuery(Of Long) _
) As Long

C#

public static long Max(
    ParallelQuery<long> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Int64))
    A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
ParallelEnumerable...::Max Method (ParallelQuery<(Of <(Nullable<(Of <(Decimal>>)>)>)>)

ParallelEnumerable Class  See Also  Send Feedback

Returns the maximum value in a parallel sequence of values.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max ( _
             source As ParallelQuery(Of Nullable(Of Decimal)) _) _
) As Nullable(Of Decimal)

C#

public static Nullable<decimal> Max(  
    ParallelQuery<Nullable<decimal>> source  
)

Parameters

source
    Type: System.Linq.Enumerable.ParallelQuery(Of Nullable(Of Nullable(Of Decimal)))
    A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...::Max Method (ParallelQuery<Of Nullable<Double>>)

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)

Returns the maximum value in a parallel sequence of values.
Syntax

Visual Basic (Declaration)

Public Shared Function Max ( _
    source As ParallelQuery(Of Nullable(Of Double)) _
) As Nullable(Of Double)

C#

public static Nullable<double> Max(
    ParallelQuery<Nullable<double>> source
)

Parameters

source
    Type: System.Linq.::: ParallelQuery<((Nullable<((Nullable<((Nullable<Double>>)>)>)>))>
    A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the maximum value in a parallel sequence of values.

**Namespace:**  System.Linq

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max ( _
    source As ParallelQuery(Of Nullable(Of Integer)) _
) As Nullable(Of Integer)

C#

public static Nullable<int> Max(
    ParallelQuery<Nullable<int>> source
)

Parameters

source
    Type: System.Linq.:::ParallelQuery<Of <(Nullable<Of <(Int32)>>)>>
    A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Max Method (ParallelQuery<Nullable<Int64>>)

Returns the maximum value in a parallel sequence of values.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max (  _
    source As ParallelQuery(Of Nullable(Of Long))  _
) As Nullable(Of Long)

C#

public static Nullable<long> Max(
    ParallelQuery<Nullable<long>> source
)

Parameters

source
    Type: `System.Linq.Enumerable.ParallelQuery<Nullable<long>>`<Of＜(Nullable<Of＜(Int64)＞)＞)＞)
A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System..::.AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System..::.OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the maximum value in a parallel sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max ( _
    source As ParallelQuery(Of Nullable(Of Single)) _
) As Nullable(Of Single)

C#

public static Nullable<float> Max(
    ParallelQuery<Nullable<float>> source
)

Parameters

source
    Type: System.Linq.:::ParallelQuery<Of (Nullable<Of (Single)>)>>
    A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System..::.AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System..::.OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Max Method (ParallelQuery<Of <(Single)>>)

ParallelEnumerable Class  See Also  Send Feedback

Returns the maximum value in a parallel sequence of values.

**Namespace:**  System.Linq

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max ( _
    source As ParallelQuery(Of Single) _
) As Single

C#

public static float Max(
    ParallelQuery<float> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<float>
    A sequence of values to determine the maximum value of.

Return Value

The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>InvalidOperationException</code></td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><code>AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the maximum value in a parallel sequence of values.

**Namespace:**  [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As TSource

C#

public static TSource Max<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq::ParallelQuery(Of (OfType))>
    A sequence of values to determine the maximum value of.
- **Type Parameters**

  TSource

- **Return Value**

  The maximum value in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements and is a non-nullable value type.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Decimal) _
) As Decimal

C#

public static decimal Max<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, decimal> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource>)
    A sequence of values to determine the maximum value of.

selector
    Type: System.Func(Of TSource, Decimal)>)
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value

The maximum value in the sequence.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
   source As ParallelQuery(Of TSource), _
   selector As Func(Of TSource, Double) _
) As Double

C#

public static double Max<TSource>(
   ParallelQuery<TSource> source,
   Func<TSource, double> selector
)

Parameters

source
   Type: System.Linq.ParallelQuery<TSource>
   A sequence of values to determine the maximum value of.

selector
   Type: System.Func<TSource, double>
   A transform function to apply to each element.
### Type Parameters

**TSource**
- The type of elements of source.

### Return Value

The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Integer) _
) As Integer

**C#**

public static int Max<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, int> selector
)

**Parameters**

source
- Type: `System.Linq.ParallelQuery(Of <TSource>)`
- A sequence of values to determine the maximum value of.

selector
- Type: `System.Func<Of<(TSource, Int32)>)`
- A transform function to apply to each element.
Type Parameters

TSource
   The type of elements of source.

Return Value

The maximum value in the sequence.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System::: AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System::: OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Long) _
) As Long

C#

public static long Max<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, long> selector
)

Parameters

source
    Type: System.Linq<pclrnamespace: ParallelQuery>(Of (Of TSource))>
    A sequence of values to determine the maximum value of.

selector
    Type: System::Func<(Of (Of TSource, Int64))>
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Max<
(OF <(TSource)>)>
Method
(ParallelQuery<
(OF <(TSource)>)>,
Func<
(OF <(TSource, Nullable<
(OF <(Decimal)>)>>)>>)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Decimal)) ) _
) As Nullable(Of Decimal)

C#

public static Nullable<decimal> Max<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<decimal>> selector
)

Parameters

source
Type: System.Linq::<ParallelQuery<TSource>>
A sequence of values to determine the maximum value of.

selector
Type: System::<Func<TSource, Nullable<decimal>>>)
A transform function to apply to each element.
**Type Parameters**

**TSource**

The type of elements of source.

**Return Value**

The maximum value in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Double)) _
) As Nullable(Of Double)

C#

public static Nullable<double> Max<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<double>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)
    A sequence of values to determine the maximum value of.

selector
    Type: System.Func(Of Nullable(Of Double), Nullable(Of Double))
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
   source As ParallelQuery(Of TSource), _
   selector As Func(Of TSource, Nullable(Of Integer)) ) _
) As Nullable(Of Integer)

C#

public static Nullable<int> Max<TSource>(
   ParallelQuery<TSource> source,
   Func<TSource, Nullable<int>> selector
)

Parameters

source
   Type: System.Linq.ParallelQuery(Of (Of TSource))>
   A sequence of values to determine the maximum value of.

selector
   Type: System.Func(Of (Of Nullable(Of (Of Int32))>)>)
   A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The maximum value in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Long)) _
) As Nullable(Of Long)

C#

public static Nullable<long> Max<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<long>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery<Of <(TSource)>>
    A sequence of values to determine the maximum value of.

selector
    Type: System.Func<Of <(TSource, Nullable<Of <(Int64)>>)>>
    A transform function to apply to each element.
**Type Parameters**

**TSource**
The type of elements of source.

**Return Value**
The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Single)) _
) As Nullable(Of Single)

C#

public static Nullable<float> Max<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<float>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of (Of Single))>)
    A sequence of values to determine the maximum value of.

selector
    Type: System.Func(Of (Of Nullable(Of (Of Single))>)>)
    A transform function to apply to each element.
**Type Parameters**

TSource
   
The type of elements of source.

**Return Value**

The maximum value in the sequence.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Public Shared Function Max(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Single) _
) As Single

**C#**

public static float Max<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, float> selector
)

**Parameters**

source
    Type: System.LinqgetStatusParallelQuery<TSource>(Of TSource>)
    A sequence of values to determine the maximum value of.

selector
    Type: System..::.Func(Of TSource, Single>)
    A transform function to apply to each element.
Type Parameters

TSource
   The type of elements of source.

Return Value

The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgumentException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

**Namespace:**  [System.Linq](/System.Linq)

**Assembly:**  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function Max(Of TSource, TResult) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, TResult) _
) As TResult
```

#### C#

```csharp
public static TResult Max<TSource, TResult>(
    ParallelQuery<TSource> source,
    Func<TSource, TResult> selector
)
```

### Parameters

**source**
- Type: `System.Linq.ParallelQuery(Of `(TSource)`)`
- A sequence of values to determine the maximum value of.

**selector**
- Type: `System.Func(Of `(TSource, TResult)`)`
- A transform function to apply to each element.
**Type Parameters**

**TSource**
The type of elements of source.

**TResult**
The type of the value returned by selector.

**Return Value**
The maximum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements and TResult is a non-nullable value type.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Max Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:::Min Method
ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min(ParallelQuery(Of Decimal))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(ParallelQuery(Of Double))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(ParallelQuery(Of Int32))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(ParallelQuery(Of Int64))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(ParallelQuery(Of Nullable(Of Decimal)))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(ParallelQuery(Of Nullable(Of Double)))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(ParallelQuery(Of Nullable(Of Int32)))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(ParallelQuery(Of Nullable(Of Int64)))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(ParallelQuery(Of Nullable(Of Single)))</td>
<td>Returns the minimum value in a parallel sequence of values.</td>
</tr>
<tr>
<td>Min(Of(Of(Of TSource))) (ParallelQuery(Of(Of(Of TSource))))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td>Min(Of(Of(Of TSource))) (ParallelQuery(Of(Of(Of TSource))), Func(Of(Of(Of TSource, Decimal))))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>Min&lt;Of (TSource)&gt;()</code> (ParallelQuery&lt;Of (TSource)&gt;, Func&lt;Of (TSource, Int32)&gt;))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td><code>Min&lt;Of (TSource)&gt;()</code> (ParallelQuery&lt;Of (TSource)&gt;, Func&lt;Of (TSource, Int64)&gt;))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td><code>Min&lt;Of (TSource)&gt;()</code> (ParallelQuery&lt;Of (TSource)&gt;, Func&lt;Of (TSource, Nullable&lt;Of (Decimal)&gt;)&gt;))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td><code>Min&lt;Of (TSource)&gt;()</code> (ParallelQuery&lt;Of (TSource)&gt;, Func&lt;Of (TSource, Nullable&lt;Of (Double)&gt;)&gt;))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td><code>Min&lt;Of (TSource)&gt;()</code> (ParallelQuery&lt;Of (TSource)&gt;, Func&lt;Of (TSource, Nullable&lt;Of (Int32)&gt;)&gt;))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td><code>Min&lt;Of (TSource)&gt;()</code> (ParallelQuery&lt;Of (TSource)&gt;, Func&lt;Of (TSource, Nullable&lt;Of (Int64)&gt;)&gt;))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td><code>Min&lt;Of (TSource)&gt;()</code> (ParallelQuery&lt;Of (TSource)&gt;, Func&lt;Of (TSource, Nullable&lt;Of (Single)&gt;)&gt;))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
<tr>
<td><code>Min&lt;Of (TSource, TResult)&gt;()</code> (ParallelQuery&lt;Of (TSource)&gt;, Func&lt;Of (TSource, Single)&gt;))</td>
<td>Invokes in parallel a transform function on each element of a sequence and returns the minimum value.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the minimum value in a parallel sequence of values.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( 
    source As ParallelQuery(Of Decimal) 
) As Decimal

C#

public static decimal Min(
    ParallelQuery<decimal> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Decimal))
    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the minimum value in a parallel sequence of values.

**Namespace:**  [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( _
    source As ParallelQuery(Of Double) _
) As Double

C#

public static double Min(
    ParallelQuery<double> source
)

Parameters

source
    Type: System.Linq::ParallelQuery(Of (Double))
    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the minimum value in a parallel sequence of values.

**Namespace:**  [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( _
    source As ParallelQuery(Of Integer) _
) As Integer

C#

public static int Min(
    ParallelQuery<int> source
)

Parameters

source
    Type: System.Linq::ParallelQuery(Of (Int32))
    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the minimum value in a parallel sequence of values.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function Min ( source As ParallelQuery(Of Long) ) As Long
```

### C#

```csharp
public static long Min( ParallelQuery<long> source )
```

## Parameters

- **source**
  - Type: `System.Linq.ParallelQuery<Of <(Int64)>>`
  - A sequence of values to determine the minimum value of.

## Return Value

- The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the minimum value in a parallel sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( _
    source As ParallelQuery(Of Nullable(Of Decimal)) _
) As Nullable(Of Decimal)

C#

public static Nullable<decimal> Min(
    ParallelQuery<Nullable<decimal>> source
)

Parameters

source
    Type: System.Linq.Enumerable.ParallelQuery(Of Nullable(Of Nullable<Decimal>>)

    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...::Min Method (ParallelQuery<Of <(Nullable<Of <(Double)>)>)>))

**ParallelEnumerable Class**  See Also  Send Feedback

Returns the minimum value in a parallel sequence of values.

**Namespace:**  System.Linq

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( _
    source As ParallelQuery(Of Nullable(Of Double)) _
) As Nullable(Of Double)

C#

public static Nullable<double> Min(
    ParallelQuery<Nullable<double>> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<Nullable<
        Nullable<
        Nullable<double>>
    >>
    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the minimum value in a parallel sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( _
    source As ParallelQuery(Of Nullable(Of Integer)) _
) As Nullable(Of Integer)

C#

public static Nullable<int> Min(
    ParallelQuery<Nullable<int>> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<Nullable<int>>
    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.Min Method (ParallelQuery(Of Nullable(Of Int64)))

Returns the minimum value in a parallel sequence of values.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( _
    source As ParallelQuery(Of Nullable(Of Long)) _
) As Nullable(Of Long)

C#

public static Nullable<long> Min(
    ParallelQuery<Nullable<long>> source
)

Parameters

source
    Type: System.Linq.
        ParallelQuery<Nullable<Of <(Nullable<Of <(Int64)>>)>>>
    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...::Min Method (ParallelQuery<(Of <(Nullable<(Of <(Single>>)>>)>>)>

ParallelEnumerable Class  See Also  Send Feedback

Returns the minimum value in a parallel sequence of values.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( _
    source As ParallelQuery(Of Nullable(Of Single)) _
) As Nullable(Of Single)

C#

public static Nullable<float> Min(
    ParallelQuery<Nullable<float>> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<Of Nullable<Of Nullable<Of Nullable<Single>>>>
    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the minimum value in a parallel sequence of values.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min ( _
    source As ParallelQuery(Of Single) _
) As Single

C#

public static float Min(
    ParallelQuery<float> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Single))
    A sequence of values to determine the minimum value of.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the minimum value in a parallel sequence of values.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As TSource

C#

public static TSource Min<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq.:::ParallelQuery(Of (Of TSource)>)
    A sequence of values to determine the minimum value of.
Type Parameters

TSource
   The type of elements of source.

Return Value

The minimum value in the sequence.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements and TSource is a non-nullable value type.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) (_
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Decimal) _
) As Decimal

C#

public static decimal Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, decimal> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery<(Of T<(TSource)>)>
    A sequence of values to determine the minimum value of.

selector
    Type: System.Func<(Of T<(TSource, Decimal)>)>
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System...:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Double) _
) As Double

C#

public static double Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, double> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource>)
    A sequence of values to determine the minimum value of.

selector
    Type: System.Func(Of TSource, Double)>)
    A transform function to apply to each element.
**Type Parameters**

**TSource**
The type of elements of source.

**Return Value**

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Integer) _
) As Integer
```

**C#**

```csharp
public static int Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, int> selector
)
```

**Parameters**

**source**
- Type: `System.Linq.ParallelQuery<TSource>`
- A sequence of values to determine the minimum value of.

**selector**
- Type: `System.Func<TSource, int>`
- A transform function to apply to each element.
Type Parameters

TSourse
   The type of elements of source.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:** [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Long) _) As Long

C#

public static long Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, long> selector
)

Parameters

source
    Type: System.Linq:::ParallelQuery<(Of <(TSource)>))
    A sequence of values to determine the minimum value of.

selector
    Type: System:::Func<(Of <(TSource, Int64)>))
    A transform function to apply to each element.
Type Parameters

TSource
   The type of elements of source.

Return Value

The minimum value in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Decimal)) _
) As Nullable(Of Decimal)

C#

public static Nullable<decimal> Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<decimal>> selector
)

Parameters

source
    Type: System.Linq.Enumerable.ParallelQuery(Of TSource)
    A sequence of values to determine the minimum value of.

selector
    Type: System.Func(Of TSource, Nullable<decimal>)
    A transform function to apply to each element.
**Type Parameters**

TSource
   The type of elements of source.

**Return Value**

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Min<TSource>() Method (ParallelQuery<TSource>, Func<TSource, Nullable<Double>>>)

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Double)) _
) As Nullable(Of Double)

C#

public static Nullable<double> Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<double>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery<(Of <(TSource)>)
    A sequence of values to determine the minimum value of.

selector
    Type: System.Func<(Of <(TSource, Nullable<(Of <(Double)>>>))>
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic C#

.NET Framework Class Library

ParallelEnumerable...:::Min(Of (Of TSource)> ) Method (ParallelQuery(Of (Of TSource)> ), Func(Of (Of TSource, Nullable(Of (Of Int32)> ) ) )

ParallelEnumerable Class  See Also  Send Feedback

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Integer)) ) _
) As Nullable(Of Integer)

C#

public static Nullable<int> Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<int>> selector
)

Parameters

source
Type: System.Linq..::.ParallelQuery(Of (Of (TSource)>)
A sequence of values to determine the minimum value of.

selector
Type: System..::.Func(Of (Of (TSource, Nullable(Of (Of (Int32)>)>)>)
A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value

The minimum value in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:**  [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)

**Assembly:**  [System.Threading](https://docs.microsoft.com/en-us/dotnet/api/system.threading) (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Long)) _
) As Nullable(Of Long)

C#

public static Nullable<long> Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<long>> selector
)

Parameters

source
    Type: System.Linq.:::ParallelQuery<(Of <(TSource)>))
    A sequence of values to determine the minimum value of.

selector
    Type: System.:::Func<(Of <(TSource, Nullable<(Of <(Int64)>)>)>))
    A transform function to apply to each element.
Type Parameters

TSource

The type of elements of source.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...::.OperationCANCELEDException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Single))) _
) As Nullable(Of Single)

C#

public static Nullable<float> Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<float>> selector
)

Parameters

source
Type: System.Linq.ParallelQuery<TSource>
A sequence of values to determine the minimum value of.

selector
Type: System.Func<TSource, Nullable<float>>
A transform function to apply to each element.
**Type Parameters**

**TSOURCE**
The type of elements of source.

**Return Value**

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Min(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Single) _
) As Single

C#

public static float Min<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, float> selector
)

Parameters

source
    Type: System.Linq:::ParallelQuery<(Of <(TSource)>)
    A sequence of values to determine the minimum value of.

selector
    Type: System:::Func<(Of <(TSource, Single)>)
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>source contains no elements.</td>
</tr>
<tr>
<td>System::: AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System::: OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```
Public Shared Function Min(Of TSource, TResult) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, TResult) _
) As TResult
```

**C#**

```
public static TResult Min<TSource, TResult>(
    ParallelQuery<TSource> source,
    Func<TSource, TResult> selector
)
```

**Parameters**

source
Type: `System.Linq.ParallelQuery<TSource>`
A sequence of values to determine the minimum value of.

selector
Type: `System.Func<TSource, TResult>`
A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

TResult
The type of the value returned by selector.

Return Value

The minimum value in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>source contains no elements and TResult is a non-nullable value type.</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Min Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Filters the elements of a ParallelQuery based on a specified type.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function OfType(Of TResult)( _
    source As ParallelQuery _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> OfType<TResult>(
    ParallelQuery source
)

Parameters

source
    Type: System.Linq.ParallelQuery
    The sequence whose elements to filter.
**Type Parameters**

TResult

The type to filter the elements of the sequence on.

**Return Value**

A sequence that contains elements from the input sequence of type TResult.
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...::OrderBy Method
ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OrderBy&lt;(Of &lt;(TSource, TKey)&gt;&gt;)</code> (ParallelQuery&lt;(Of &lt;(TSource)&gt;&gt;), Func&lt;(Of &lt;(TSource, TKey)&gt;&gt;))</td>
<td>Sorts in parallel the elements of a sequence in ascending order according to a key.</td>
</tr>
<tr>
<td><code>OrderBy&lt;(Of &lt;(TSource, TKey)&gt;&gt;)</code> (ParallelQuery&lt;(Of &lt;(TSource)&gt;&gt;), Func&lt;(Of &lt;(TSource, TKey)&gt;&gt;), IComparer&lt;(Of &lt;(TKey)&gt;&gt;))</td>
<td>Sorts in parallel the elements of a sequence in ascending order by using a specified comparer.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Sorts in parallel the elements of a sequence in ascending order according to a key.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

Public Shared Function OrderBy(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As OrderedParallelQuery(Of TSource)

#### C#

public static OrderedParallelQuery<TSource> OrderBy<TSource, TKey>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector
)

### Parameters

**source**

Type: System.Linq.Enumerable.ParallelQuery(Of (Of TSource)>)

A sequence of values to order.

**keySelector**

Type: System.Linq.Enumerable.Func(Of (Of TSource, TKey)>)

A function to extract a key from an element.
Type Parameters

TSource
   The type of elements of source.
TKey
   The type of the key returned by keySelector.

Return Value

An OrderedParallelQuery{TSource} whose elements are sorted according to a key.
Remarks

In contrast to the sequential implementation, this is not a stable sort. To achieve a stable sort, change a query of the form:

```csharp
var ordered = source.OrderBy((e) => e.k);
```

to instead be formed as:

```csharp
var ordered = source.Select((e,i) => new { E=e, I=i }).OrderBy((v) =>
```
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
OrderBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Sorts in parallel the elements of a sequence in ascending order by using a specified comparer.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function OrderBy(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    comparer As IComparer(Of TKey) _
) As OrderedParallelQuery(Of TSource)

C#

public static OrderedParallelQuery<TSource> OrderBy<TSource, TKey>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    IComparer<TKey> comparer
)

Parameters

source
Type: System.Linq...::ParallelQuery<(Of <(TSource)>))
A sequence of values to order.

keySelector
Type: System...::Func<(Of <(TSource, TKey)>))
A function to extract a key from an element.

comparer
Type: System.Collections.Generic...::IComparer<(Of <(TKey)>))
An IComparer{TKey} to compare keys.
**Type Parameters**

**TSource**
The type of elements of source.

**TKey**
The type of the key returned by keySelector.

**Return Value**
An `OrderedParallelQuery{TSource}` whose elements are sorted according to a key.
Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for `OrderBy(ParallelQuery<TSource>, Func<TSource,TKey>)` for an approach to implementing a stable sort.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
OrderBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.OrderByDescending Method

ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OrderByDescending&lt;Of &lt;(TSource, TKey)&gt;&gt;(ParallelQuery&lt;Of &lt;(TSource)&gt;, Func&lt;Of &lt;(TSource, TKey)&gt;&gt;)</code></td>
<td>Sorts in parallel the elements of a sequence in descending order according to a key.</td>
</tr>
<tr>
<td><code>OrderByDescending&lt;Of &lt;(TSource, TKey)&gt;&gt;(ParallelQuery&lt;Of &lt;(TSource)&gt;, Func&lt;Of &lt;(TSource, TKey)&gt;&gt;, IComparer&lt;Of &lt;(TKey)&gt;))</code></td>
<td>Sorts the elements of a sequence in descending order by using a specified comparer.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#

.NET Framework Class Library
ParallelEnumerable...::OrderByDescending<(Of <(TSource, TKey)>)> Method
(ParallelQuery<(Of <(TSource)>), Func<(Of <(TSource, TKey)>))

ParallelEnumerable Class  See Also  Send Feedback

Sorts in parallel the elements of a sequence in descending order according to a key.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Shared Function OrderByDescending(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey) _) As OrderedParallelQuery(Of TSource)
```

**C#**

```csharp
public static OrderedParallelQuery<TSource> OrderByDescending<TSource, TKey>(TSource source,
    Func<TSource, TKey> keySelector)
```

**Parameters**

**source**
- Type: `System.Linq.ParallelQuery<TSource>`
- A sequence of values to order.

**keySelector**
- Type: `System.Func<TSource, TKey>`
- A function to extract a key from an element.
**Type Parameters**

**TSource**  
The type of elements of source.

**TKey**  
The type of the key returned by keySelector.

**Return Value**

An OrderedParallelQuery{TSource} whose elements are sorted descending according to a key.
Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy(ParallelQuery{TSource}, Func{TSource,TKey}) for an approach to implementing a stable sort.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
OrderByDescending Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable::OrderByDescending(Of (TSource, TKey)) Method

Sorts the elements of a sequence in descending order by using a specified comparer.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function OrderByDescending(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    comparer As IComparer(Of TKey) _
) As OrderedParallelQuery(Of TSource)

C#

public static OrderedParallelQuery<TSource> OrderByDescending<TSource, TKey>
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    IComparer<TKey> comparer

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource)>)
    A sequence of values to order.

keySelector
    Type: System.Func(Of (Of TSource, TKey)>)
    A function to extract a key from an element.

comparer
    Type: System.Collections.Generic.IComparer(Of TKey)>
    An IComparer{TKey} to compare keys.
Type Parameters

TSource
The type of elements of source.

TKey
The type of the key returned by keySelector.

Return Value

An OrderedParallelQuery{TSource} whose elements are sorted descending according to a key.
Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for `OrderBy(ParallelQuery<TSource>, Func<TSource,TKey>)` for an approach to implementing a stable sort.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
OrderByDescending Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Generates a parallel sequence of integral numbers within a specified range.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

Public Shared Function Range (  
    start As Integer,  
    count As Integer  
) As ParallelQuery(Of Integer)

**C#**

public static ParallelQuery<int> Range(
    int start,
    int count
)

### Parameters

**start**
Type: System:::Int32
The value of the first integer in the sequence.

**count**
Type: System:::Int32
The number of sequential integers to generate.

### Return Value

An `IEnumerable<Int32>` in C# or `IEnumerable(Of Int32)` in Visual Basic that contains a range of sequential integral numbers.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgumentOutOfRangeException</td>
<td>count is less than 0 -or- start + count - 1 is larger than MaxValue()</td>
</tr>
<tr>
<td>System::ArgumentOutOfRangeException</td>
<td></td>
</tr>
</tbody>
</table>

00.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Generates a parallel sequence that contains one repeated value.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function Repeat(Of TResult) ( _
    element As TResult, _
    count As Integer _
) As ParallelQuery(Of TResult)
```

**C#**

```csharp
public static ParallelQuery<TResult> Repeat<TResult>(
    TResult element,
    int count
)
```

### Parameters

- **element**
  - Type: TResult
  - The value to be repeated.

- **count**
  - Type: System..::.Int32
  - The number of times to repeat the value in the generated sequence.
**Type Parameters**

**TResult**
The type of the value to be repeated in the result sequence.

**Return Value**

A sequence that contains a repeated value.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentOutOfRangeException</td>
<td>count is less than 0.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Inverts the order of the elements in a parallel sequence.

**Namespace:**  [System.Linq](/System.Linq)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Reverse(Of TSource) ( _
   source As ParallelQuery(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Reverse<TSource>(
   ParallelQuery<TSource> source
)

Parameters

source
   Type: System.Linq.ParallelQuery<Of ((Of (TSource)>)>
A sequence of values to reverse.
Type Parameters

TSource
   The type of the elements of source.

Return Value

A sequence whose elements correspond to those of the input sequence in reverse order.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.ExceptionSource</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::Select Method

ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select&lt;(Of &lt;(TSOURCE, TRESULT)&gt;), ParallelQuery&lt;(Of &lt;(TSOURCE)&gt;), Func&lt;(Of &lt;(TSOURCE, TRESULT)&gt;))&gt;</td>
<td>Projects in parallel each element of a sequence into a new form.</td>
</tr>
<tr>
<td>Select&lt;(Of &lt;(TSOURCE, TRESULT)&gt;), ParallelQuery&lt;(Of &lt;(TSOURCE)&gt;), Func&lt;(Of &lt;(TSOURCE, INT32, TRESULT)&gt;))&gt;</td>
<td>Projects in parallel each element of a sequence into a new form by incorporating the element's index.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Projects in parallel each element of a sequence into a new form.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Select(Of TSource, TResult) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, TResult) _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> Select<TSource, TResult>(
    ParallelQuery<TSource> source,
    Func<TSource, TResult> selector
)

Parameters

source
    Type: System.Linq:::ParallelQuery<(Of <(TSource)>))
    A sequence of values to invoke a transform function on.

selector
    Type: System:::Func<(Of <(TSource, TResult)>))
    A transform function to apply to each element.
Type Parameters

TSource
   The type of the elements of source.
 TResult
   The type of elements resturned by selector.

Return Value

A sequence whose elements are the result of invoking the transform function on each element of source.
<table>
<thead>
<tr>
<th><strong>Exception</strong></th>
<th><strong>Condition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SystemArgumentNullException</code></td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Select Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Projects in parallel each element of a sequence into a new form by incorporating the element's index.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Select(Of TSource, TResult) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Integer, TResult) _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> Select<TSource, TResult>(
    ParallelQuery<TSource> source,
    Func<TSource, int, TResult> selector
)

Parameters

source
    Type: System.Linq..:::ParallelQuery<(Of <(TSource)>))
    A sequence of values to invoke a transform function on.

selector
    Type: System..:::.Func<(Of <(TSource, Int32, TResult)>))
    A transform function to apply to each element.
Type Parameters

TSource
The type of the elements of source.

TResult
The type of elements returned by selector.

Return Value

A sequence whose elements are the result of invoking the transform function on each element of source.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference</td>
</tr>
<tr>
<td></td>
<td>(Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Select Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:::SelectMany Method
ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| SelectMany<
(TSource,
TResult>><(ParallelQuery<
(Of
<TSource>)),
Func<
(Of
<TSource, IEnumerable<(Of
<(TResult)>)>))> |
Projects in parallel each element of a sequence to an IEnumerable{T} and flattens the resulting sequences into one sequence. |
| SelectMany<
(Of
<TSource,
TResult>)(ParallelQuery<
(Of
<(TSource>)),
Func<
(Of
<(TSource, Int32,
IEnumerable<(Of
<(TResult)>)>))>),
Func<
(Of
<(TSource, IList, IEnumerables<
(Of
<(TCollection, TResult)>)>))>))> |
Projects in parallel each element of a sequence to an IEnumerable{T}, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element. |
| SelectMany<
(Of
<(TSource,
TCollection, TResult)>)(ParallelQuery<
(Of
<(TSource>)),
Func<
(Of
<(TSource, Int32,
IEnumerable<(Of
<(TCollection, TResult)>)>)),
Func<
(Of
<(TSource, IList, IEnumerables<
(Of
<(TCollection, TResult)>)>))>))> |
Projects each element of a sequence to an IEnumerable{T}, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. |
| SelectMany<
(Of
<(TSource,
TCollection, TResult)>)(ParallelQuery<
(Of
<(TSource>)),
Func<
(Of
<(TSource, Int32,
IEnumerable<(Of
<(TCollection, TResult)>)>)),
Func<
(Of
<(TSource, IList, IEnumerables<
(Of
<(TCollection, TResult)>)>))>))> |
Projects each element of a sequence to an IEnumerable{T}, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Projects in parallel each element of a sequence to an IEnumerable{T} and flattens the resulting sequences into one sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

Public Shared Function SelectMany(Of TSource, TResult) ( _
  source As ParallelQuery(Of TSource), _
  selector As Func(Of TSource, IEnumerable(Of TResult)) _
) As ParallelQuery(Of TResult)

#### C#

public static ParallelQuery<TResult> SelectMany<TSource, TResult>(
  ParallelQuery<TSource> source,
  Func<TSource, IEnumerable<TResult>> selector
)

### Parameters

**source**
- Type: System.Linq:::ParallelQuery<(Of <(TSource)>))
  - A sequence of values to project.

**selector**
- Type: System:::Func<(Of <(TSource, IEnumerable<(Of <(TResult)>))>))
  - A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

TResult
The type of the elements of the sequence returned by selector.

Return Value

A sequence whose elements are the result of invoking the one-to-many transform function on each element of the input sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SelectMany Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Projects in parallel each element of a sequence to an IEnumerable{T}, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function SelectMany(Of TSource, TResult) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Integer, IEnumerable(Of TResult))
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> SelectMany<TSource, TResult>(
    ParallelQuery<TSource> source,
    Func<TSource, int, IEnumerable<TResult>> selector
)

Parameters

source
   Type: System.Linq.ParallelQuery(Of (Of (Ts.Source)>)
   A sequence of values to project.

selector
   Type: System.Func(Of (Of (Ts.Source, Int32, IEnumerable(Of (Of (TResult)>)>)>)
   A transform function to apply to each element.
**Type Parameters**

TSource
   The type of elements of source.

TResult
   The type of the elements of the sequence returned by selector.

**Return Value**

A sequence whose elements are the result of invoking the one-to-many transform function on each element of the input sequence.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference</td>
</tr>
<tr>
<td></td>
<td>(Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SelectMany Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.::SelectMany<(Of <(TSource, TCollection, TResult)>)> Method (ParallelQuery<(Of <(TSource)>), Func<(Of <(TSource, IEnumerable<(Of <(TCollection)>)>)>), Func<(Of <(TSource, TCollection, TResult)>)>))

Projects each element of a sequence to an IEnumerable{T}, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function SelectMany(Of TSource, TCollection, TResult) source As ParallelQuery(Of TSource), _
collectionSelector As Func(Of TSource, IEnumerable(Of TCollection)), _
resultSelector As Func(Of TSource, TCollection, TResult)) _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> SelectMany<TSource, TCollection>
ParallelQuery<TSource> source,
Func<TSource, IEnumerable<TCollection>> collectionSelector,
Func<TSource, TCollection, TResult> resultSelector

Parameters

source
Type: System.Linq.ParallelQuery(Of Of (TSource)>)
A sequence of values to project.

collectionSelector
Type: System.Func(Of Of (TSource, IEnumerable(Of {(TCollection)}))>
A transform function to apply to each source element; the second parameter of the function represents the index of the source element.

resultSelector
Type: System.Func(Of Of (TSource, TCollection, TResult)>)
A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.
- **Type Parameters**

  **TSource**
  The type of elements of source.

  **TCollection**
  The type of the intermediate elements collected by `collectionSelector`.

  **TResult**

- **Return Value**

  A sequence whose elements are the result of invoking the one-to-many transform function `collectionSelector` on each element of source and then mapping each of those sequence elements and their corresponding source element to a result element.
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
</table>
| System..::.ArgumentNullException | source or collectionSelector or
resultSelector is a null reference (Nothing in Visual Basic). |
See Also

ParallelEnumerable Class
SelectMany Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Projects each element of a sequence to an IEnumerable{T}, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function SelectMany(Of TSource, TCollection, TResult) 
  source As ParallelQuery(Of TSource), _
  collectionSelector As Func(Of TSource, Integer, IEnumerable(TCollection))
  resultSelector As Func(Of TSource, TCollection, TResult) _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> SelectMany<TSource, TCollection>
  ParallelQuery<TSource> source,
  Func<TSource, int, IEnumerable<TCollection>> collectionSelector,
  Func<TSource, TCollection, TResult> resultSelector

Parameters

source
  Type: System.Linq.ParallelQuery(Of (Of TSource)>)
  A sequence of values to project.

collectionSelector
  Type: System.Func(Of (Of TSource, Int32, IEnumerable(Of (Of TCollection)>)>)
  A transform function to apply to each source element; the second parameter of the function represents the index of the source element.

resultSelector
  Type: System.Func(Of (Of TSource, TCollection, TResult)>)
  A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.
**Type Parameters**

TSource
   The type of elements of source.
TCollection
   The type of the intermediate elements collected by collectionSelector.
TResult
   The type of elements to return.

**Return Value**

A sequence whose elements are the result of invoking the one-to-many transform function collectionSelector on each element of source and then mapping each of those sequence elements and their corresponding source element to a result element.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or collectionSelector or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SelectMany Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:::SequenceEqual Method
ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| SequenceEqual<TSource>
  (ParallelQuery<TSource>,
   IEnumerable<TSource>)<br>SequenceEqual<TSource>
  (ParallelQuery<TSource>,
   ParallelQuery<TSource>)<br>SequenceEqual<TSource>
  (ParallelQuery<TSource>,
   IEnumerable<TSource>,
   IEquatable<T>)<br>SequenceEqual<TSource>
  (ParallelQuery<TSource>,
   ParallelQuery<TSource>,
   IEquatable<T>)<br>SequenceEqual<TSource>
  (ParallelQuery<TSource>,
   ParallelQuery<TSource>,
   IEquatable<T>) | **Obsolete.**
This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called. Determines whether two parallel sequences are equal by comparing their elements by using the default equality comparer for their type. |
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

```csharp
<ObsoleteAttribute("The second data source of a binary operator must
Public Shared Function SequenceEqual(Of TSource) ( _
    first As ParallelQuery(Of TSource), _
    second As IEnumerable(Of TSource) _
) As Boolean

[ObsoleteAttribute("The second data source of a binary operator must
public static bool SequenceEqual<TSource>(
    ParallelQuery<TSource> first,
    IEnumerable<TSource> second
)

Parameters

first
    Type: System.Linq.:::ParallelQuery<((TSource))>
    This parameter is not used.

second
    Type: System.Collections.Generic.:::IEnumerable<((TSource))>
    This parameter is not used.
Type Parameters

TSource
   This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of SequenceEqual with a left data source of type `ParallelQuery<Of<TSource>>` and a right data source of type `IEnumerable<Of<T>>`. Otherwise, the SequenceEqual operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>Thrown every time this method is called.</td>
</tr>
</tbody>
</table>
See Also

- ParallelEnumerable Class
- SequenceEqual Overload
- System.Linq Namespace

Send feedback on this topic to Microsoft.
Determines whether two parallel sequences are equal by comparing the elements by using the default equality comparer for their type.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function SequenceEqual(Of TSource) ( _
first As ParallelQuery(Of TSource), _
second As ParallelQuery(Of TSource)) _
) As Boolean
```

### C#

```csharp
public static bool SequenceEqual<TSource>(
ParallelQuery<TSource> first,
ParallelQuery<TSource> second
)
```

### Parameters

**first**

Type: `System.Linq.ParallelQuery(Of TSource)`
A sequence to compare to **second**.

**second**

Type: `System.Linq.ParallelQuery(Of TSource)`
A sequence to compare to the first input sequence.
Type Parameters

TSource
   The type of the elements of the input sequences.

Return Value

true if the two source sequences are of equal length and their corresponding elements are equal according to the default equality comparer for their type; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><strong>System...:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SequenceEqual Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.SequenceEqual(Of TSource) Method (ParallelQuery(Of TSource), IEnumerable(Of TSource), IEqualityComparer(Of TSource))

This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

```csharp
<ObsoleteAttribute("The second data source of a binary operator must
public Shared Function SequenceEqual(Of TSource) ( _
    first As ParallelQuery(Of TSource), _
    second As IEnumerable(Of TSource), _
    comparer As IEqualityComparer(Of TSource) _
) As Boolean
```

```csharp
[ObsoleteAttribute("The second data source of a binary operator must
public static bool SequenceEqual<TSource>(
    ParallelQuery<TSource> first,
    IEnumerable<TSource> second,
    IEqualityComparer<TSource> comparer
)
```

**Parameters**

**first**
Type: `System.Linq.ParallelQuery<Of (Of TSource)>`
This parameter is not used.

**second**
Type: `System.Collections.Generic.IEnumerable<Of (Of TSource)>`
This parameter is not used.

**comparer**
Type: `System.Collections.Generic.IEqualityComparer<Of (Of TSource)>`
This parameter is not used.
Type Parameters

TSource
   This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of SequenceEqual with a left data source of type `ParallelQuery<Of <(TSource)>>` and a right data source of type `IEnumerable<Of <(T)> >`. Otherwise, the SequenceEqual operator would appear to be binding to the parallel implementation, but would in reality bind to sequential implementation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>Thrown every time this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SequenceEqual Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Determines whether two parallel sequences are equal by comparing their elements by using a specified IEqualityComparer{T}.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function SequenceEqual(Of TSource) ( _  
    first As ParallelQuery(Of TSource), _  
    second As ParallelQuery(Of TSource), _  
    comparer As IEqualityComparer(Of TSource) _  
) As Boolean

C#

public static bool SequenceEqual<TSource>(
    ParallelQuery<TSource> first,
    ParallelQuery<TSource> second,
    IEqualityComparer<TSource> comparer
)

Parameters

first
    Type: System.Linq.ParallelQuery(Of (Of TSource)>)
    A sequence to compare to second.

second
    Type: System.Linq.ParallelQuery(Of (Of TSource)>)
    A sequence to compare to the first input sequence.

comparer
    Type: System.Collections.Generic.IEqualityComparer(Of (Of T>)>
    An IEqualityComparer(Of (T)> to use to compare elements.
Type Parameters

TSource
The type of the elements of the input sequences.

Return Value
true if the two source sequences are of equal length and their corresponding elements are equal according to the default equality comparer for their type; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SequenceEqual Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:::Single Method
ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;)&gt;))</td>
<td>Returns the only element of a parallel sequence, and throws an exception if there is not exactly one element in the sequence.</td>
</tr>
<tr>
<td>Single&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;&gt;), Func&lt;(Of &lt;(TSource, Boolean)&gt;)&gt;))</td>
<td>Returns the only element of a parallel sequence that satisfies a specified condition, and throws an exception if more than one such element exists.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the only element of a parallel sequence, and throws an exception if there is not exactly one element in the sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Shared Function Single(Of TSource) ( _
    source As ParallelQuery(Of TSource) _) As TSource
```

**C#**

```csharp
public static TSource Single<TSource>(
    ParallelQuery<TSource> source
)
```

**Parameters**

source
Type: `System.Linq.:::ParallelQuery<(<TSource>)>`
The sequence to return the single element of.
**Type Parameters**

**TSource**

The type of the elements of source.

**Return Value**

The single element of the input sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The input sequence contains more than one element. -or- The input sequence is empty.</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Single Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the only element of a parallel sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Single(Of TSource) ( _
source As ParallelQuery(Of TSource), _
predicate As Func(Of TSource, Boolean) _
) As TSource

C#

public static TSource Single<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
Type: System.Linq:::ParallelQuery<(Of <(TSource)>)
The sequence to return the single element of.

predicate
Type: System...::Func<(Of <(TSource, Boolean)>)
A function to test an element for a condition.
Type Parameters

TSource
   The type of the elements of source.

Return Value

The single element of the input sequence that satisfies a condition.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>No element satisfies the condition in predicate. -or- More than one element satisfies the condition in predicate.</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Single Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable....SingleOrDefault Method
ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SingleOrDefault&lt;TSource&gt;(ParallelQuery&lt;TSource&gt;)</code></td>
<td>Returns the only element of a parallel sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.</td>
</tr>
<tr>
<td><code>SingleOrDefault&lt;TSource&gt;(ParallelQuery&lt;TSource&gt;, Func&lt;TSource, Boolean&gt;)</code></td>
<td>Returns the only element of a parallel sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the only element of a parallel sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function SingleOrDefault(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As TSource

C#

public static TSource SingleOrDefault<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of <(TSource)>))
    The sequence to return the single element of.
Type Parameters

TSource
   The type of the elements of source.

Return Value

The single element of the input sequence, or default(TSource) if the sequence contains no elements.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SingleOrDefault Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns the only element of a parallel sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function SingleOrDefault(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _) As TSource
```

### C#

```csharp
public static TSource SingleOrDefault<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)
```

## Parameters

**source**
- Type: `System.Linq.ParallelQuery<TSource>`
- The sequence to return the single element of.

**predicate**
- Type: `System.Func<TSource, Boolean>`
- A function to test an element for a condition.
**Type Parameters**

**TSource**
The type of the elements of source.

**Return Value**

The single element of the input sequence that satisfies the condition, or default(TSource) if no such element is found.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SingleOrDefault Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Bypasses a specified number of elements in a parallel sequence and then returns the remaining elements.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Skip(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    count As Integer _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Skip<TSource>(
    ParallelQuery<TSource> source,
    int count
)

Parameters

source
Type: System.Linq.ParallelQuery(Of (Of (TSource)>)
The sequence to return elements from.

count
Type: System.Int32
The number of elements to skip before returning the remaining elements.
Type Parameters

TSource

The type of elements of source.

Return Value

A sequence that contains the elements that occur after the specified index in the input sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:::SkipWhile Method
ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SkipWhile&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TSource, Boolean)&gt;)&gt;))</code></td>
<td>Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements.</td>
</tr>
<tr>
<td><code>SkipWhile&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TSource, Int32, Boolean)&gt;)&gt;))</code></td>
<td>Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.SkipWhile(Of (Of TSource)>)) Method
(ParallelQuery(Of (Of TSource)>), Func(Of (Of TSource, Boolean)>))

Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function SkipWhile(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As ParallelQuery(Of TSource)
```

**C#**

```csharp
public static ParallelQuery<TSource> SkipWhile<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)
```

### Parameters

- **source**
  Type: `System.Linq.ParallelQuery<(Of <(TSource)>)>`
  The sequence to return elements from.

- **predicate**
  Type: `System.Func<(Of<(TSource, Boolean)>)>`
  A function to test each element for a condition.
Type Parameters

TSource
   The type of elements of source.

Return Value

A sequence that contains the elements from the input sequence starting at the first element in the linear series that does not pass the test specified by predicate.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::$.ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SkipWhile Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function SkipWhile(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Integer, Boolean) _) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> SkipWhile<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, int, bool> predicate
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (TSource)>)
    The sequence to return elements from.

predicate
    Type: System.Func(Of (TSource, Int32, Boolean)>)
    A function to test each source element for a condition; the second parameter of the function represents the index of the source element.
**Type Parameters**

TSource

The type of elements of source.

**Return Value**

A sequence that contains the elements from the input sequence starting at the first element in the linear series that does not pass the test specified by predicate.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
SkipWhile Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.Sum Method

ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Decimal&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Double&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Int32&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Int64&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Decimal)&gt;)&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Double)&gt;)&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Int32)&gt;)&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Int64)&gt;)&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum(ParallelQuery&lt;Of &lt;(Nullable&lt;Of &lt;(Single)&gt;)&gt;)&gt;)</code></td>
<td>Computes in parallel the sum of a sequence of values.</td>
</tr>
<tr>
<td><code>Sum&lt;Of &lt;(TSource)&gt;) (ParallelQuery&lt;Of &lt;(TSource)&gt;)&gt;, Func&lt;Of &lt;(TSource, Decimal)&gt;)&gt;</code></td>
<td>Computes in parallel the sum of values that are obtained by invoking a transform function on each element of the input sequence.</td>
</tr>
<tr>
<td><code>Sum&lt;Of &lt;(TSource)&gt;) (ParallelQuery&lt;Of &lt;(TSource)&gt;) , Func&lt;Of &lt;(TSource, Double)&gt;)&gt;</code></td>
<td>Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.</td>
</tr>
</tbody>
</table>
(ParallelQuery<Of (TSource)>, Func<Of (TSource, Int32)>)
Sum<Of (TSource)>
(ParallelQuery<Of (TSource)>, Func<Of (TSource, Int64)>)
Sum<Of (TSource)>
(ParallelQuery<Of (TSource)>, Func<Of (TSource, Nullable(Of (Decimal)>>)>)
Sum<Of (TSource)>
(ParallelQuery<Of (TSource)>, Func<Of (TSource, Nullable(Of (Double)>>)>)
Sum<Of (TSource)>
(ParallelQuery<Of (TSource)>, Func<Of (TSource, Nullable(Of (Int32)>>)>)
Sum<Of (TSource)>
(ParallelQuery<Of (TSource)>, Func<Of (TSource, Nullable(Of (Int64)>>)>)
Sum<Of (TSource)>
(ParallelQuery<Of (TSource)>, Func<Of (TSource, Nullable(Of (Single)>>)>)
Sum<Of (TSource)>
(ParallelQuery<Of (TSource)>, Func<Of (TSource, Single)>)

sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:** [System.Linq](https://docs.microsoft.com)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum ( _
    source As ParallelQuery(Of Decimal) _
) As Decimal

C#

public static decimal Sum(
    ParallelQuery<decimal> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Decimal))
    A sequence of values to calculate the sum of.

Return Value

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>The sum is larger than MaxValue(). -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum ( source As ParallelQuery(Of Double) ) As Double

C#

public static double Sum( ParallelQuery<double> source )

Parameters

source

Type: System.Linq.ParallelQuery(Of Double)

A sequence of values to calculate the sum of.

Return Value

The sum of the values in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum ( _
    source As ParallelQuery(Of Integer) _
) As Integer

C#

public static int Sum(
    ParallelQuery<int> source
)

Parameters

source
    Type: System.Linq:: ParallelQuery(Of (Int32)>)
    A sequence of values to calculate the sum of.

Return Value

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The sum is larger than MaxValue(). -or- One or more exceptions occurred</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum ( _
    source As ParallelQuery(Of Long) _
) As Long

C#

public static long Sum(  
    ParallelQuery<long> source
)

Parameters

source
	Type: System.Linq.:::ParallelQuery<Of <(Int64)>>
	A sequence of values to calculate the sum of.

Return Value

The sum of the values in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>The sum is larger than <code>MaxValue()()</code>.</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.\::\::Sum Method (ParallelQuery<(Of <(Nullable<(Of <(Decimal>))>)>)>)

ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the sum of a sequence of values.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Shared Function Sum ( _  
    source As ParallelQuery(Of Nullable(Of Decimal)) _  
) As Nullable(Of Decimal)

**C#**

public static Nullable<decimal> Sum(
    ParallelQuery<Nullable<decimal>> source
)

**Parameters**

source

Type: System.Linq.:::ParallelQuery<((Nullable<Of Nullable<Of Nullable<Decimal>>))>)

A sequence of values to calculate the sum of.

**Return Value**

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td></td>
<td>The sum is larger than MaxValue()().</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The query was canceled.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum ( _
    source As ParallelQuery(Of Nullable(Of Double)) _) As Nullable(Of Double)

C#

public static Nullable<double> Sum(
    ParallelQuery<Nullable<double>> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of Nullable(Of Double))
    A sequence of values to calculate the sum of.

Return Value

The sum of the values in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum (_
    source As ParallelQuery(Of Nullable(Of Integer)) _
) As Nullable(Of Integer)

C#

public static Nullable<int> Sum(
    ParallelQuery<Nullable<int>> source
)

Parameters

source
    Type: System.Linq.:::ParallelQuery(Of <(Nullable(Of <(Int32)>))>)
    A sequence of values to calculate the sum of.

Return Value

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td></td>
<td>The sum is larger than <code>MaxValue()</code>.</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>AggregateException</code></td>
<td></td>
</tr>
<tr>
<td><code>OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:**  System.Linq
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function Sum (_
    source As ParallelQuery(Of Nullable(Of Long)) _
) As Nullable(Of Long)
```

**C#**

```csharp
public static Nullable<long> Sum(
    ParallelQuery<Nullable<long>> source
)
```

**Parameters**

- `source`: Type: `System.Linq.Enumerable.ParallelQuery<Nullable<long>>` (Of Nullable(Of Int64>)
  A sequence of values to calculate the sum of.

**Return Value**

The sum of the values in the sequence.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System::.AggregateException</td>
<td>The sum is larger than MaxValue().</td>
</tr>
<tr>
<td>-or-</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum ( _
    source As ParallelQuery(Of Nullable(Of Single)) _
) As Nullable(Of Single)

C#

public static Nullable<float> Sum(
    ParallelQuery<Nullable<float>> source
)

Parameters

source
    Type: System.Linq::ParallelQuery<(Of <(Nullable<(Of <(Single)>)>))>
    A sequence of values to calculate the sum of.

Return Value

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of a sequence of values.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum ( _
    source As ParallelQuery(Of Single) _
) As Single

C#

public static float Sum(
    ParallelQuery<float> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of Single)
    A sequence of values to calculate the sum of.

Return Value

The sum of the values in the sequence.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:..ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System...:..AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System...:..OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable::Sum(Of (Of TSource)) Method (ParallelQuery(Of (Of TSource)), Func(Of (Of TSource, Decimal)))

ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Decimal) _
) As Decimal

C#

public static decimal Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, decimal> selector
)

Parameters

source
  Type: System.Linq::$::ParallelQuery(Of (TSource)>)
  A sequence of values to calculate the sum of.

selector
  Type: System...::Func(Of (TSource, Decimal)>)
  A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td></td>
<td>The sum is larger than <code>MaxValue()()</code>.</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System..::.AggregateException</code></td>
<td></td>
</tr>
<tr>
<td><code>System..::.OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Double) _
) As Double
```

### C#

```csharp
public static double Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, double> selector
)
```

### Parameters

**source**
- Type: `System.Linq.ParallelQuery<TSource>`
- A sequence of values to calculate the sum of.

**selector**
- Type: `System.Func<TSource, Double>`
- A transform function to apply to each element.
Type Parameters

TSource
   The type of elements of source.

Return Value

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:**  System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```
Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Integer) _
) As Integer
```

**C#**

```
public static int Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, int> selector
)
```

**Parameters**

source
Type: System.Linq.ParallelQuery<TSource>()
A sequence of values to calculate the sum of.

selector
Type: System.Func<TSource, int>()
A transform function to apply to each element.
Type Parameters

TSource

Return Value

The sum of the values in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The sum is larger than MaxValue(). -or- One or more exceptions occurred</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>during the evaluation of the query.</td>
</tr>
<tr>
<td></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Long) _
) As Long

C#

public static long Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, long> selector
)

Parameters

source
   Type: System.Linq.ParallelQuery(Of T<TSource>)
   A sequence of values to calculate the sum of.

selector
   Type: System.Func(Of T<TSource, Int64>)
   A transform function to apply to each element.
**Type Parameters**

TSource  
The type of elements of source.

**Return Value**

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>The sum is larger than <code>MaxValue()</code>(). -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Decimal)) _
) As Nullable(Of Decimal)

C#

public static Nullable<decimal> Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<decimal>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery<TSource>
    A sequence of values to calculate the sum of.

selector
    Type: System.Func<TSource, Nullable<decimal>>
    A transform function to apply to each element.
## Type Parameters

TSource  
The type of elements of source.

## Return Value

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The sum is larger than MaxValue(). -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::Sum<Of <(TSource>> Method (ParallelQuery<Of <(TSource>>>, Func<Of <(TSource, Nullable<Of <(Double)>>)>>))

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Double)) _) As Nullable(Of Double)

C#

public static Nullable<double> Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<double>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)
    A sequence of values to calculate the sum of.

selector
    Type: System.Func(Of TSource, Nullable(Of Double))
    A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The sum of the values in the sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference</td>
</tr>
<tr>
<td></td>
<td>(Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred</td>
</tr>
<tr>
<td></td>
<td>during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable::Sum<
(OfType<TSource>)>
Method
(ParallelQuery<
(OfType<TSource>)>,
Func<
(OfType<TSource,
Nullable<
(OfType<Int32>)>)>)

ParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the sum of the sequence of values that are obtained by
invoking a transform function on each element of the input sequence.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Integer)) _
) As Nullable(Of Integer)

C#

public static Nullable<int> Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<int>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource>)
    A sequence of values to calculate the sum of.

selector
    Type: System.Func(Of TSource, Nullable(Of Int32)>)
    A transform function to apply to each element.
- **Type Parameters**

  *TSource*
  
  The type of elements of source.

- **Return Value**

  The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The sum is larger than MaxValue()().</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Long)) ) As Nullable(Of Long)

C#

public static Nullable<long> Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<long>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource)
    A sequence of values to calculate the sum of.

selector
    Type: System.Func(Of TSource, Nullable(Of Long))
    A transform function to apply to each element.
**Type Parameters**

TSource

The type of elements of source.

**Return Value**

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The sum is larger than MaxValue(). -or- One or more exceptions occurred</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>during the evaluation of the query.</td>
</tr>
<tr>
<td></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable::Sum<TSource>(ParallelQuery<TSource>, Func<TSource, Nullable<Single>>)) Method (ParallelQuery<TSource>, Func<TSource, Nullable<Single>>)\n\nParallelEnumerable Class  See Also  Send Feedback

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Nullable(Of Single)) _) _
) As Nullable(Of Single)

C#

public static Nullable<float> Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, Nullable<float>> selector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource))>
    A sequence of values to calculate the sum of.

selector
    Type: System.Func(Of (Of TSource, Nullable(Of Single))>)>
    A transform function to apply to each element.
**Type Parameters**

TSource
   The type of elements of source.

**Return Value**

The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

**Namespace:**  [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)

**Assembly:**  [System.Threading](https://docs.microsoft.com/en-us/dotnet/api/system.threading) (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function Sum(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    selector As Func(Of TSource, Single) _) As Single
```

### C#

```csharp
public static float Sum<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, float> selector
)
```

## Parameters

**source**
- Type: `System.Linq.ParallelQuery<TSource>`
- A sequence of values to calculate the sum of.

**selector**
- Type: `System.Func<TSource, Single>`
- A transform function to apply to each element.
Type Parameters

TSource
The type of elements of source.

Return Value
The sum of the values in the sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or selector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><strong>System:::AggregateException</strong></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Sum Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns a specified number of contiguous elements from the start of a parallel sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Take(Of TSource) ( _
           source As ParallelQuery(Of TSource), _
           count As Integer _
    ) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Take<TSource>(
    ParallelQuery<TSource> source,
    int count
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource>)
    The sequence to return elements from.

count
    Type: System.Int32
    The number of elements to return.
Type Parameters

TSource
   The type of elements of source.

Return Value

A sequence that contains the specified number of elements from the start of the input sequence.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable::TakeWhile Method
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TakeWhile&lt;Of (Of&lt;TSource&gt;)&gt;(ParallelQuery&lt;Of&lt;TSource&gt;&gt;, Func&lt;Of&lt;TSource, Boolean&gt;))</code></td>
<td>Returns elements from a parallel sequence as long as a specified condition is true.</td>
</tr>
<tr>
<td><code>TakeWhile&lt;Of (Of&lt;TSource&gt;)&gt;(ParallelQuery&lt;Of&lt;TSource&gt;), Func&lt;Of&lt;TSource, Int32, Boolean&gt;))</code></td>
<td>Returns elements from a parallel sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns elements from a parallel sequence as long as a specified condition is true.

**Namespace:** [System.Linq](/System.Linq)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TakeWhile(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> TakeWhile<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
    Type: System.Linq.ParallelQuery<Of <(TSource)>>
    The sequence to return elements from.

predicate
    Type: System.Func<Of <(TSource, Boolean)>>
    A function to test each element for a condition.
Type Parameters

TSource
   The type of elements of source.

Return Value

A sequence that contains the elements from the input sequence that occur before the element at which the test no longer passes.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
TakeWhile Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns elements from a parallel sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function TakeWhile(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Integer, Boolean) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> TakeWhile<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, int, bool> predicate
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of TSource>)
    The sequence to return elements from.

predicate
    Type: System.Func(Of TSource, Int32, Boolean>)
    A function to test each source element for a condition; the second parameter of the function represents the index of the source element.
Type Parameters

TSource
The type of elements of source.

Return Value

A sequence that contains elements from the input sequence that occur before the element at which the test no longer passes.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..:..ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
TakeWhile Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ThenBy&lt;(Of &lt;(TSource, TKey)&gt;)(OrderedParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TSource, TKey)&gt;))&gt;</code></td>
<td>Performs in parallel a subsequent ordering of the elements in a sequence in ascending order according to a key.</td>
</tr>
<tr>
<td><code>ThenBy&lt;(Of &lt;(TSource, TKey)&gt;)(OrderedParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TSource, TKey)&gt;), IComparer&lt;(Of &lt;( TKey)&gt;))&gt;</code></td>
<td>Performs in parallel a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Performs in parallel a subsequent ordering of the elements in a sequence in ascending order according to a key.

**Namespace:**  [System.Linq](/system.linq)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ThenBy(Of TSource, TKey) ( _
    source As OrderedParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As OrderedParallelQuery(Of TSource)

C#

public static OrderedParallelQuery<TSource> ThenBy<TSource, TKey>(
    OrderedParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector
)

Parameters

source
    Type: System.Linq..::.OrderedParallelQuery<(Of <>Nullable)(Of TSource)>)
    An OrderedParallelQuery<TSource> than contains elements to sort.

keySelector
    Type: System..::.Func<(Of <>Nullable)(Of TSource, TKey)>)
    A function to extract a key from an element.
Type Parameters

TSource
   The type of elements of source.
TKey
   The type of the key returned by keySelector.

Return Value

An OrderedParallelQuery{TSource} whose elements are sorted according to a key.
Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for `OrderBy(ParallelQuery<TSource>, Func<TSource,TKey>)` for an approach to implementing a stable sort.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ThenBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Performs in parallel a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ThenBy(Of TSource, TKey) ( _
    source As OrderedParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    comparer As IComparer(Of TKey) _
) As OrderedParallelQuery(Of TSource)

C#

public static OrderedParallelQuery<TSource> ThenBy<TSource, TKey>(
    OrderedParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    IComparer<TKey> comparer
)

Parameters

source
Type: System.Linq:::OrderedParallelQuery<(Of <(TSource)>))
An OrderedParallelQuery{TSource} that contains elements to sort.

keySelector
Type: System:::Func<(Of <(TSource, TKey)>))
A function to extract a key from an element.

comparer
Type: System.Collections.Generic:::IComparer<(Of <(TKey)>))
An IComparer{TKey} to compare keys.
**Type Parameters**

**TSource**
- The type of elements of source.

**TKey**
- The type of the key returned by keySelector.

**Return Value**

An OrderedParallelQuery{TSource} whose elements are sorted according to a key.
Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy(ParallelQuery<TSource>, Func<TSource,TKey>) for an approach to implementing a stable sort.
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ThenBy Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...::ThenByDescending Method
ParallelEnumerable Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ThenByDescending(Of &lt;(TSource, TKey)&gt;)(OrderedParallelQuery(Of &lt;(TSource)&gt;, Func(Of &lt;(TSource, TKey)&gt;)))</code></td>
<td>Performs in parallel a subsequent ordering of the elements in a sequence in descending order, according to a key.</td>
</tr>
<tr>
<td><code>ThenByDescending&lt;(Of &lt;(TSOURCE, TKey)&gt;)&gt;(OrderedParallelQuery&lt;(Of &lt;(TSOURCE)&gt;)&gt;, Func&lt;(Of &lt;(TSOURCE, TKey)&gt;)&gt;, IComparer&lt;(Of &lt;(TKey)&gt;)&gt;)</code></td>
<td>Performs in parallel a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Performs in parallel a subsequent ordering of the elements in a sequence in descending order, according to a key.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ThenByDescending(Of TSource, TKey) ( _
   source As OrderedParallelQuery(Of TSource), _
   keySelector As Func(Of TSource, TKey) _
) As OrderedParallelQuery(Of TSource)

C#

public static OrderedParallelQuery<TSource> ThenByDescending<TSource,
   OrderedParallelQuery<TSource> source,
   Func<TSource, TKey> keySelector
)

Parameters

source
   Type: System.Linq.:::OrderedParallelQuery<(Of <(TSource)>)>)
   An OrderedParallelQuery<TSource> than contains elements to sort.

keySelector
   Type: System.:::Func<(Of <(TSource, TKey)>)>)
   A function to extract a key from an element.
Type Parameters

TSource
The type of elements of source.

TKey
The type of the key returned by keySelector.

Return Value

An OrderedParallelQuery{TSource} whose elements are sorted descending according to a key.
Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy(ParallelQuery<TSource>, Func<TSource,TKey>) for an approach to implementing a stable sort.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ThenByDescending Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Perform in parallel a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ThenByDescending(Of TSource, TKey) ( _
    source As OrderedParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    comparer As IComparer(Of TKey) _
) As OrderedParallelQuery(Of TSource)

C#

public static OrderedParallelQuery<TSource> ThenByDescending<TSource>(
    OrderedParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    IComparer<TKey> comparer
)

Parameters

source
    Type: System.Linq.:::OrderedParallelQuery<Of <(TSource)>>
    An OrderedParallelQuery{TSource} than contains elements to sort.

keySelector
    Type: System.:::Func<Of <(TSource, TKey)>>
    A function to extract a key from an element.

comparer
    Type: System.Collections.Generic.:::IComparer<Of <(TKey)>>
    An IComparer{TKey} to compare keys.
Type Parameters

TSource
- The type of elements of source.

TKey
- The type of the key returned by keySelector.

Return Value

An OrderedParallelQuery{TSource} whose elements are sorted descending according to a key.
Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for `OrderBy(ParallelQuery<TSource>, Func<TSource,TKey>)` for an approach to implementing a stable sort.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ThenByDescending Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Creates an array from a ParallelQuery{T}.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
```
Visual Basic (Declaration)

Public Shared Function ToArray(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As TSource()

C#

public static TSource[] ToArray<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq.ParallelQuery<TSource>
    A sequence to create an array from.
```
Type Parameters

TSOURCE
The type of the elements of source.

Return Value

An array that contains the elements from the input sequence.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:ToDictionary Method
ParallelEnumerable Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToDictionary(Of (Of (TSource, TKey)&gt;), (ParallelQuery(Of (TSource)&gt;), Func(Of (Of (TSource, TKey)&gt;)))</td>
<td>Creates a Dictionary{TKey,TValue} from a ParallelQuery{T} according to a specified key selector function.</td>
</tr>
<tr>
<td>ToDictionary(Of (Of (TSource, TKey)&gt;), (ParallelQuery(Of (TSource)&gt;), Func(Of (Of (TSource, TKey)&gt;), IEqualityComparer(Of (Of (TSource, TKey)&gt;)))</td>
<td>Creates a Dictionary{TKey,TValue} from a ParallelQuery{T} according to a specified key selector function and key comparer.</td>
</tr>
<tr>
<td>ToDictionary(Of (Of (TSource, TKey, TElement)&gt;), (ParallelQuery(Of (TSource)&gt;), Func(Of (Of (TSource, TKey)&gt;), Func(Of (Of (TSource, TElement)&gt;)))</td>
<td>Creates a Dictionary{TKey,TValue} from a ParallelQuery{T} according to specified key selector and element selector functions.</td>
</tr>
<tr>
<td>ToDictionary(Of (Of (TSource, TKey, TElement)&gt;), (ParallelQuery(Of (TSource)&gt;), Func(Of (Of (TSource, TKey)&gt;), Func(Of (Of (TSource, TElement)&gt;), IEqualityComparer(Of (Of (TKey)&gt;)))</td>
<td>Creates a Dictionary{TKey,TValue} from a ParallelQuery{T} according to a specified key selector function, a comparer, and an element selector function.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable::ToDictionary(Of (TSource, TKey)) Method
(ParallelQuery(Of (TSource)), Func(Of (TSource, TKey)))

ParallelEnumerable Class  See Also  Send Feedback

Creates a Dictionary{TKey,TValue} from a ParallelQuery{T} according to a specified key selector function.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ToDictionary(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As Dictionary(Of TKey, TSource)

C#

public static Dictionary<TKey, TSource> ToDictionary<TSource, TKey>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of<TSource>)
    A sequence to create a Dictionary(Of<(TKey, TValue)>) from.

keySelector
    Type: System.Func(Of<(TSourse, TKey)>)
    A function to extract a key from each element.
**Type Parameters**

TSource
   The type of the elements of source.
TKey
   The type of the key returned by keySelector.

**Return Value**

A Dictionary<(Of <(TKey, TValue)>)> that contains keys and values.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td></td>
<td>keySelector produces a key that is a null reference (Nothing in Visual Basic). -or- keySelector produces duplicate keys for two elements. -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td></td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ToDictionary Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::ToDictionary(Of (TSource, TKey)>) Method (ParallelQuery(Of (TSource)<>), Func(Of (TSource, TKey)<>), IEqualityComparer(Of (TKey)<>))

ParallelEnumerable Class  See Also  Send Feedback

Creates a Dictionary{TKey,TValue} from a ParallelQuery{T} according to a specified key selector function and key comparer.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ToDictionary(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    comparer As IEqualityComparer(Of TKey) _
) As Dictionary(Of TKey, TSource)

C#

public static Dictionary<TKey, TSource> ToDictionary<TSource, TKey>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    IEqualityComparer<TKey> comparer
)

Parameters

source
   Type: System.Linq.Enumerable.ParallelQuery(Of ((TSource)>)
   A sequence to create a Dictionary(Of ((TKey, TValue)>) from.

keySelector
   Type: System.Enumerable.Enumerable.Func(Of ((TSource, TKey)>)
   A function to extract a key from each element.

comparer
   An IEqualityComparer(Of ((T)>) to compare keys.
**Type Parameters**

**TSource**
The type of the elements of source.

**TKey**
The type of the key returned by keySelector.

**Return Value**

A Dictionary<(Of <(TKey, TValue)>)> that contains keys and values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::.ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td></td>
<td>keySelector produces a key that is a null reference (Nothing in Visual</td>
</tr>
<tr>
<td></td>
<td>Basic). -or- keySelector produces duplicate keys for two elements. -or-</td>
</tr>
<tr>
<td></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System:::.AggregateException</td>
<td></td>
</tr>
<tr>
<td>System:::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ToDictionary Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.ToDictionary(Of (TSource, TKey, TElement)>)) Method (ParallelQuery(Of (TSource)>, Func(Of (TSource, TKey)>, Func(Of (TSource, TElement)>)>

ParallelEnumerable Class  See Also  Send Feedback

Creates a Dictionary{TKey,TValue} from a ParallelQuery{T} according to specified key selector and element selector functions.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ToDictionary(Of TSource, TKey, TElement) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement) _
) As Dictionary(Of TKey, TElement)

C#

public static Dictionary<TKey, TElement> ToDictionary<TSource, TKey, TElement>
    (ParallelQuery<TSource> source,
     Func<TSource, TKey> keySelector,
     Func<TSource, TElement> elementSelector)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource))>
    A sequence to create a Dictionary(Of (Of TKey, TValue))>) from.

keySelector
    Type: System.Func(Of (Of TSource, TKey)>)
    A function to extract a key from each element.

elementSelector
    Type: System.Func(Of (Of TSource, TElement)>)
    A transform function to produce a result element value from each element.
**Type Parameters**

**TSource**
- The type of the elements of source.

**TKey**
- The type of the key returned by keySelector.

**TElement**
- The type of the value returned by elementSelector.

**Return Value**

A Dictionary<(Of <(TKey, TValue)>)> that contains values of type TElement selected from the input sequence
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector or elementSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>keySelector produces a key that is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td></td>
<td>-or- keySelector produces duplicate keys for two elements.</td>
</tr>
<tr>
<td></td>
<td>-or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ToDictionary Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.ToDictionary(Of (TSource, TKey, TElement)>)
Method (ParallelQuery(Of (TSource)<>), Func(Of (TSource, TKey)<>), Func(Of (TSource, TElement)<>), IEqualityComparer(Of (TKey)<>))

Creates a Dictionary{TKey,TValue} from a ParallelQuery{T} according to a specified key selector function, a comparer, and an element selector function.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Visual Basic (Declaration)

Public Shared Function ToDictionary(Of TSource, TKey, TElement) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement), _
    comparer As IEqualityComparer(Of TKey) _
) As Dictionary(Of TKey, TElement)

C#

public static Dictionary<TKey, TElement> ToDictionary<TSource, TKey, TElement>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector,
    IEqualityComparer<TKey> comparer
)

Parameters

tSource
    Type: System.Linq.ParallelQuery(Of (Of TSource)>)
    A sequence to create a Dictionary(Of (Of TKey, TValue)>) from.

d KeySelector
    Type: System.Func(Of (Of TSource, TKey)>)
    A function to extract a key from each element.

e ElementSelector
    Type: System.Func(Of (Of TSource, TElement)>)
    A transform function to produce a result element value from each element.

c Comparer
    Type: System.Collections.Generic.IEqualityComparer(Of TKey)>
    An IEqualityComparer(Of (Of T)> to compare keys.
**Type Parameters**

**TSource**
- The type of the elements of source.

**TKey**
- The type of the key returned by keySelector.

**TElement**
- The type of the value returned by elementSelector.

**Return Value**

A Dictionary<(Of<(TKey, TValue)>)> that contains values of type TElement selected from the input sequence
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source or keySelector or elementSelector is a null reference (Nothing in Visual Basic). keySelector produces a key that is a null reference (Nothing in Visual Basic). -or- keySelector produces duplicate keys for two elements. -or- One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>The query was canceled.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td></td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ToDictionary Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.ToList(Of TSource) Method

Creates a List{T} from a ParallelQuery{T}.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ToList(Of TSource) ( _
    source As ParallelQuery(Of TSource) _
) As List(Of TSource)

C#

public static List<TSource> ToList<TSource>(
    ParallelQuery<TSource> source
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource)>)
    A sequence to create a List(Of ((T)>)) from.
Type Parameters

TSource
   The type of the elements of source.

Return Value

A List<(Of <(T)>)> that contains elements from the input sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...:::ToLookup Method

ParallelEnumerable Class  See Also  Send Feedback
# Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ToLookup(Of &lt;(TSource, TKey)&gt;)</code>&lt;br&gt;<code>(ParallelQuery(Of &lt;(TSource)&gt;)</code>,&lt;br&gt;<code>Func(Of &lt;(TSource, TKey)&gt;)</code>)`</td>
<td>Creates an ILookup{TKey,T} from a ParallelQuery{T} according to a specified key selector function. Creates an ILookup{TKey,T} from a ParallelQuery{T} according to a specified key selector function and key comparer. \n\nCreates an ILookup{TKey,TElement} from a ParallelQuery{T} according to specified key selector and element selector functions. \n\nCreates an ILookup{TKey,TElement} from a ParallelQuery{T} according to a specified key selector function, a comparer and an element selector function.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.ToLookup(Of (TSource, TKey)> Method
(ParallelQuery(Of (TSource)>), Func(Of (TSource, TKey)>))

ParallelEnumerable Class  See Also  Send Feedback

Creates an ILookup{TKey,T} from a ParallelQuery{T} according to a specified key selector function.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ToLookup(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As ILookup(Of TKey, TSource)

C#

public static ILookup<TKey, TSource> ToLookup<TSource, TKey>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource))>
    The sequence to create a Lookup(Of (Of TKey, TElement)> from.

keySelector
    Type: System.Func(Of (Of TSource, TKey)>)
    A function to extract a key from each element.
Type Parameters

TSource
    The type of elements of source.
TKey
    The type of the key returned by keySelector.

Return Value

A Lookup<(Of <(TKey, TElement)>)) that contains keys and values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ToLookup Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.ToLookup(Of (TSource, TKey)) Method
(ParallelEnumerable(Of TSource), Func(Of (TSource, TKey) => TKey),
IEqualityComparer(Of TKey))

ParallelEnumerable Class  See Also  Send Feedback

Creates an ILookup{TKey,T} from a ParallelEnumerable{T} according to a specified
key selector function and key comparer.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ToLookup(Of TSource, TKey) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    comparer As IEqualityComparer(Of TKey) _
) As ILookup(Of TKey, TSource)

C#

public static ILookup<TKey, TSource> ToLookup<TSource, TKey>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    IEqualityComparer<TKey> comparer
)

Parameters

source
    Type: System.Linq.:::ParallelQuery(Of (TSource)>)
    The sequence to create a Lookup(Of (TKey, TElement)>) from.

keySelector
    Type: System.:::Func(Of (TSource, TKey)>)
    A function to extract a key from each element.

comparer
    Type: System.Collections.Generic.:::IEqualityComparer(Of (TKey)>)
    An IEqualityComparer(Of (T)>) to compare keys.
Type Parameters

**TSource**
The type of elements of source.

**TKey**
The type of the key returned by keySelector.

Return Value

A Lookup<(Of <(TKey, TElement)>)> that contains keys and values.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentNullException</code></td>
<td>source or keySelector or is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System..::.AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System..::.OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ToLookup Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable..::.ToLookup(Of (TSource, TKey, TElement)>)) Method (ParallelQuery(Of (TSource)>), Func(Of (TSource, TKey)>), Func(Of (TSource, TElement)>))

ParallelEnumerable Class  See Also  Send Feedback

Creates an ILookup{TKey,TElement} from a ParallelQuery{T} according to specified key selector and element selector functions.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Shared Function ToLookup(Of TSource, TKey, TElement) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement) _
) As ILookup(Of TKey, TElement)
```

**C#**

```csharp
public static ILookup<TKey, TElement> ToLookup<TSource, TKey, TElement>
    (ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector)
```

**Parameters**

**source**
- Type: `System.Linq.ParallelQuery<TSource>`
- The sequence to create a Lookup<(Of <(TKey, TElement)>)> from.

**keySelector**
- Type: `System.Func<TSource, TKey>`
- A function to extract a key from each element.

**elementSelector**
- Type: `System.Func<TSource, TElement>`
- A transform function to produce a result element value from each element.
Type Parameters

TSource
The type of elements of source.

TKey
The type of the key returned by keySelector.

TElement
The type of the value returned by elementSelector.

Return Value

A Lookup<(Of <(TKey, TElement)>)) that contains values of type TElement selected from the input sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>source or keySelector or elementSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ToLookup Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
ParallelEnumerable.ToLookup(Of (TSource, TKey, TElement)>) Method
(ParallelQuery(Of TSource), Func(Of (TSource, TKey) -> TElement), Func(Of (TSource, TElement) -> TElement), IEqualityComparer(Of TKey))
ParallelEnumerable Class  See Also  Send Feedback

Creates an ILookup{TKey,TElement} from a ParallelQuery{T} according to a specified key selector function, a comparer and an element selector function.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ToLookup(Of TSource, TKey, TElement) ( _
    source As ParallelQuery(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement), _
    comparer As IEqualityComparer(Of TKey) _
) As ILookup(Of TKey, TElement)

C#

public static ILookup<TKey, TElement> ToLookup<TSource, TKey, TElement>(
    ParallelQuery<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector,
    IEqualityComparer<TKey> comparer
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of <TSource>))
    The sequence to create a Lookup(Of <(TKey, TElement)>)) from.

keySelector
    Type: System.Func(Of (Of <TSource, TKey>))
    A function to extract a key from each element.

elementSelector
    Type: System.Func(Of (Of <TSource, TElement>))
    A transform function to produce a result element value from each element.

comparer
    Type: System.Collections.Generic.IEqualityComparer(Of TKey)
    An IEqualityComparer(Of (Of <T>)) to compare keys.
Type Parameters

TSource
    The type of elements of source.
TKey
    The type of the key returned by keySelector.
TElement
    The type of the value returned by elementSelector.

Return Value

A Lookup<(Of <(TKey, TElement)>)> that contains values of type TElement selected from the input sequence.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or keySelector or elementSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>One or more exceptions occurred during the evaluation of the query.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The query was canceled.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
ToLookup Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union&lt;(Of,&lt;TSource&gt;)&gt; (ParallelQuery&lt;(Of,&lt;TSource&gt;)&gt;),</td>
<td><strong>Obsolete.</strong> This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.</td>
</tr>
<tr>
<td>IEnumerable&lt;(Of,&lt;TSource&gt;))</td>
<td></td>
</tr>
<tr>
<td>Union&lt;(Of,&lt;TSource&gt;)&gt; (ParallelQuery&lt;(Of,&lt;TSource&gt;)&gt;),</td>
<td>Produces the set union of two parallel sequences by using the default equality comparer.</td>
</tr>
<tr>
<td>ParallelQuery&lt;(Of,&lt;TSource&gt;)) (ParallelQuery&lt;(Of,&lt;TSource&gt;)&gt;),</td>
<td><strong>Obsolete.</strong> This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.</td>
</tr>
<tr>
<td>IEqualityComparer&lt;(Of,&lt;TSource&gt;))</td>
<td></td>
</tr>
<tr>
<td>Union&lt;(Of,&lt;TSource&gt;)&gt; (ParallelQuery&lt;(Of,&lt;TSource&gt;)&gt;),</td>
<td>Produces the set union of two parallel sequences by using a specified IEqualityComparer{T}.</td>
</tr>
<tr>
<td>ParallelQuery&lt;(Of,&lt;TSource&gt;)&gt; (ParallelQuery&lt;(Of,&lt;TSource&gt;)&gt;),</td>
<td></td>
</tr>
<tr>
<td>IEqualityComparer&lt;(Of,&lt;TSource&gt;))</td>
<td></td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

```csharp

Public Shared Function Union(Of TSource) ( _
    first As ParallelQuery(Of TSource), _
    second As IEnumerable(Of TSource) _
) As ParallelQuery(Of TSource)

    ParallelQuery<TSource> first,
    IEnumerable<TSource> second
)
```

### Parameters

**first**

Type: `System.Linq.ParallelQuery<Of <(TSource)>>`
This parameter is not used.

**second**

Type: `System.Collections.Generic.IEnumerable<Of <(TSource)>>`
This parameter is not used.
Type Parameters

TSource
    This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Union with a left data source of type `ParallelQuery<Of <(TSource)>>` and a right data source of type `IEnumerable<Of <(T)>>`. Otherwise, the Union operator would appear to be binding to the parallel implementation, but would in reality bind to sequential implementation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Union Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable::Union<TSource> Method (ParallelQuery<TSource>, ParallelQuery<TSource>)

Produces the set union of two parallel sequences by using the default equality comparer.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

Public Shared Function Union(Of TSource) ( _
    first As ParallelQuery(Of TSource), _
    second As ParallelQuery(Of TSource) _
) As ParallelQuery(Of TSource)

**C#**

public static ParallelQuery<TSource> Union<TSource>(
    ParallelQuery<TSource> first,
    ParallelQuery<TSource> second
)

### Parameters

**first**

Type: System.Linq.:::ParallelQuery<Of (Of(TSource)>)
A sequence whose distinct elements form the first set for the union.

**second**

Type: System.Linq.:::ParallelQuery<Of (Of(TSource)>)
A sequence whose distinct elements form the second set for the union.
**Type Parameters**

TSource

The type of the elements of the input sequences.

**Return Value**

A sequence that contains the elements from both input sequences, excluding duplicates.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Union Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable...::Union(Of (Of (TSource)>) Method (ParallelQuery(Of (Of (TSource)>)>, IEnumerable(Of (Of (TSource)>)>, IEqualityComparer(Of (Of (TSource)>)>)

ParallelEnumerable Class  See Also  Send Feedback

This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

```csharp
<ObsoleteAttribute("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Linq.ParallelQuery<T>.")>

Public Shared Function Union(Of TSource) ( _
  first As ParallelQuery(Of TSource), _
  second As IEnumerable(Of TSource), _
  comparer As IEqualityComparer(Of TSource) _
) As ParallelQuery(Of TSource)
```

```csharp
[ObsoleteAttribute("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Linq.ParallelQuery<T>.")]
public static ParallelQuery<TSource> Union<TSource>(
  ParallelQuery<TSource> first,
  IEnumerable<TSource> second,
  IEqualityComparer<TSource> comparer
)
```

### Parameters

**first**
Type: `System.Linq.ParallelQuery<Of (Of TSource)>`  
This parameter is not used.

**second**
Type: `System.Collections.Generic.IEnumerable<Of (Of TSource)>`  
This parameter is not used.

**comparer**
Type: `System.Collections.Generic.IEqualityComparer<Of (Of TSource)>`  
This parameter is not used.
Type Parameters

TSource
  This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Union with a left data source of type `ParallelQuery<Of <(TSource)>>` and a right data source of type `IEnumerable<Of <(T)>>`. Otherwise, the Union operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System....NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Union Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelEnumerable.:::Union(Of (TSource)) Method (ParallelQuery(Of (TSource)), ParallelQuery(Of (TSource)), IEqualityComparer(Of (TSource)))

ParallelEnumerable Class  See Also  Send Feedback

Produces the set union of two parallel sequences by using a specified IEqualityComparer{T}.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Union(Of TSource) ( _
    first As ParallelQuery(Of TSource), _
    second As ParallelQuery(Of TSource), _
    comparer As IEqualityComparer(Of TSource) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Union<TSource>(
    ParallelQuery<TSource> first,
    ParallelQuery<TSource> second,
    IEqualityComparer<TSource> comparer
)

Parameters

first
    Type: System.Linq.:::ParallelQuery<(Of <(TSource)>))
    A sequence whose distinct elements form the first set for the union.

second
    Type: System.Linq.:::ParallelQuery<(Of <(TSource)>))
    A sequence whose distinct elements form the second set for the union.

comparer
    Type: System.Collections.Generic.:::IEqualityComparer<(Of <(T)>))
    An IEqualityComparer<(Of <(T)>)) to compare values.
Type Parameters

TSource
   The type of the elements of the input sequences.

Return Value

A sequence that contains the elements from both input sequences, excluding duplicates.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>first or second is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Union Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ParallelEnumerable...:::Where Method
ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TSource, Boolean)&gt;)&gt;))</td>
<td>Filters in parallel a sequence of values based on a predicate.</td>
</tr>
<tr>
<td>Where&lt;(Of &lt;(TSource)&gt;)&gt; (ParallelQuery&lt;(Of &lt;(TSource)&gt;), Func&lt;(Of &lt;(TSource, Int32, Boolean)&gt;)&gt;))</td>
<td>Filters in parallel a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Filters in parallel a sequence of values based on a predicate.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Where(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Where<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, bool> predicate
)

Parameters

source
Type: System.Linq.Enumerable.ParallelQuery(Of (Of (TSource)>)
A sequence to filter.

predicate
Type: System.Func(Of (Of (TSource, Boolean)>)
A function to test each element for a condition.
Type Parameters

TSource
   The type of the elements of source.

Return Value

A sequence that contains elements from the input sequence that satisfy the condition.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Where Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Filters in parallel a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Where(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    predicate As Func(Of TSource, Integer, Boolean) _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> Where<TSource>(
    ParallelQuery<TSource> source,
    Func<TSource, int, bool> predicate
)

Parameters

source
    Type: System.Linq.ParallelQuery(Of (Of TSource))>
    A sequence to filter.

predicate
    Type: System.Func(Of (Of TSource, Int32, Boolean)>)
    A function to test each element for a condition.
Type Parameters

TSource
   The type of the elements of source.

Return Value

A sequence that contains elements from the input sequence that satisfy the condition.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::ArgumentException</td>
<td>source or predicate is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Where Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Sets the `CancellationToken` to associate with the query.

**Namespace:**  `System.Linq`

**Assembly:**  `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Public Shared Function WithCancellation(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    cancellationToken As CancellationToken _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> WithCancellation<TSource>(
    ParallelQuery<TSource> source,
    CancellationToken cancellationToken
}

Parameters

source
    Type: System.Linq.:::ParallelQuery<(Of <(TSource)>))
    A ParallelQuery on which to set the option.

cancellationToken
    Type: System.Threading.:::CancellationToken
    A cancellation token.
Type Parameters

TSource
The type of elements of source.

Return Value

ParallelQuery representing the same query as source, but with the registered.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>WithCancellation is used multiple times in the query.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The <a href="#">CancellationTokenSource</a> associated with the cancellationToken has been disposed.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace
System.Threading.CancellationToken

Send feedback on this topic to Microsoft.
Sets the degree of parallelism to use in a query. Degree of parallelism is the maximum number of concurrently executing tasks that will be used to process the query.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function WithDegreeOfParallelism(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    degreeOfParallelism As Integer _
) As ParallelQuery(Of TSource)
```

**C#**

```csharp
public static ParallelQuery<TSource> WithDegreeOfParallelism<TSource>(
    ParallelQuery<TSource> source,
    int degreeOfParallelism
)
```

### Parameters

**source**
- Type: `System.Linq.ParallelQuery<TSource>`
- A `ParallelQuery` on which to set the limit on the degrees of parallelism.

**degreeOfParallelism**
- Type: `System.Int32`
- The degree of parallelism for the query.
Type Parameters

TSource
The type of elements of source.

Return Value

ParallelQuery representing the same query as source, with the limit on the degrees of parallelism set.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>WithDegreeOfParallelism is used multiple times in the query.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>degreeOfParallelism is less than 1 or greater than 63.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Sets the execution mode of the query.

**Namespace:** System.Linq

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function WithExecutionMode(Of TSource) ( _
  source As ParallelQuery(Of TSource), _
  executionMode As ParallelExecutionMode )
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> WithExecutionMode<TSource>(
  ParallelQuery<TSource> source,
  ParallelExecutionMode executionMode
)

Parameters

source
  Type: System.Linq.:::ParallelQuery<(Of (Of<TSource>))>
  A ParallelQuery on which to set the option.

executionMode
  Type: System.Linq.:::ParallelExecutionMode
  The mode in which to execute the query.
Type Parameters

TSource
   The type of elements of source.

Return Value

ParallelQuery representing the same query as source, but with the registered.
<table>
<thead>
<tr>
<th><strong>Exception</strong></th>
<th><strong>Condition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>executionMode is not a valid ParallelExecutionMode value.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>WithExecutionMode is used multiple times in the query.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace
System.Linq... ParallelExecutionMode

Send feedback on this topic to Microsoft.
Sets the merge options for this query, which specify how the query will buffer output.

Namespace:  System.Linq
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function WithMergeOptions(Of TSource) ( _
    source As ParallelQuery(Of TSource), _
    mergeOptions As ParallelMergeOptions _
) As ParallelQuery(Of TSource)

C#

public static ParallelQuery<TSource> WithMergeOptions<TSource>(
    ParallelQuery<TSource> source,
    ParallelMergeOptions mergeOptions
)

Parameters

source
    Type: System.Linq.ParallelQuery<TSource><Of <(TSource)>>
    A ParallelQuery on which to set the option.

mergeOptions
    Type: System.Linq.ParallelMergeOptions
    The merge options to set for this query.
**Type Parameters**

**TSource**

The type of elements of source.

**Return Value**

ParallelQuery representing the same query as source, but with the registered.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>source is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>mergeOptions is not a valid ParallelMergeOptions value.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>WithMergeOptions is used multiple times in the query.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class  
System.Linq Namespace  
System.Linq.ParallelMergeOptions

Send feedback on this topic to Microsoft.
ParallelEnumerable.Zip Method

ParallelEnumerable Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zip&lt;(Of &lt;(TFirst, TSecond, TResult)&gt;)(ParallelQuery&lt;(Of &lt;(TFirst)&gt;), IEnumerable&lt;(Of &lt;(TSecond)&gt;), Func&lt;(Of &lt;(TFirst, TSecond, TResult)&gt;)) &gt;))</td>
<td><strong>Obsolete.</strong> This Zip overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.</td>
</tr>
<tr>
<td>Zip&lt;(Of &lt;(TFirst, TSecond, TResult)&gt;)(ParallelQuery&lt;(Of &lt;(TFirst)&gt;), ParallelQuery&lt;(Of &lt;(TSecond)&gt;), Func&lt;(Of &lt;(TFirst, TSecond, TResult)&gt;)) &gt;))</td>
<td>Merges in parallel two sequences by using the specified predicate function.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
This Zip overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)
### Syntax

```csharp
<ObsoleteAttribute("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>.", use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")>

Public Shared Function Zip(Of TFirst, TSecond, TResult) ( _
    first As ParallelQuery(Of TFirst), _
    second As IEnumerable(Of TSecond), _
    resultSelector As Func(Of TFirst, TSecond, TResult) _
) As ParallelQuery(Of TResult)
```

```csharp
[ObsoleteAttribute("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>.", use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static ParallelQuery<TResult> Zip<TFirst, TSecond, TResult>(
    ParallelQuery<TFirst> first,
    IEnumerable<TSecond> second,
    Func<TFirst, TSecond, TResult> resultSelector
)
```

### Parameters

**first**
- Type: `System.Linq.ParallelQuery(Of (Of(TFirst)>)`
  - This parameter is not used.

**second**
- Type: `System.Collections.Generic.IEnumerable(Of (Of(TSecond)>)`
  - This parameter is not used.

**resultSelector**
- Type: `System.Func(Of (Of(TFirst, TSecond, TResult)>)`
  - This parameter is not used.
Type Parameters

TFirst
  This type parameter is not used.
TSecond
  This type parameter is not used.
TResult
  This type parameter is not used.

Return Value

This overload always throws a NotSupportedException.
Remarks

This overload exists to disallow usage of Zip with a left data source of type `ParallelQuery<TSource>` and a right data source of type `IEnumerable<T>`. Otherwise, the Zip operator would appear to be bind to the parallel implementation, but would in reality bind to the sequential implementation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NotSupportedException</td>
<td>The exception that occurs when this method is called.</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Zip Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
Merges in parallel two sequences by using the specified predicate function.

**Namespace:**  [System.Linq](#)  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Zip(Of TFirst, TSecond, TResult) ( _
    first As ParallelQuery(Of TFirst), _
    second As ParallelQuery(Of TSecond), _
    resultSelector As Func(Of TFirst, TSecond, TResult) _
) As ParallelQuery(Of TResult)

C#

public static ParallelQuery<TResult> Zip<TFirst, TSecond, TResult>(
    ParallelQuery<TFirst> first,
    ParallelQuery<TSecond> second,
    Func<TFirst, TSecond, TResult> resultSelector
)

Parameters

first
    Type: System.Linq.ParallelQuery(Of (Of TFirst))>
    The first sequence to zip.

second
    Type: System.Linq.ParallelQuery(Of (Of TSecond))>
    The second sequence to zip.

resultSelector
    Type: System.Func(Of (Of TFirst, TSecond, TResult))>
    A function to create a result element from two matching elements.
Type Parameters

TFirst
The type of the elements of the first sequence.
TSecond
The type of the elements of the second sequence.
TResult
The type of the return elements.

Return Value

A sequence that has elements of type TResult that are obtained by performing resultSelector pairwise on two sequences. If the sequence lengths are unequal, this truncates to the length of the shorter sequence.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::..ArgumentNullException</td>
<td>first or second or resultSelector is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ParallelEnumerable Class
Zip Overload
System.Linq Namespace

Send feedback on this topic to Microsoft.
The query execution mode is a hint that specifies how the system should handle performance trade-offs when parallelizing queries.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Enumeration ParallelExecutionMode

C#

public enum ParallelExecutionMode
```
### Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>By default, the system will use algorithms for queries that are ripe for parallelism and will avoid algorithms with high overheads that will likely result in slowdowns for parallel execution.</td>
</tr>
<tr>
<td>ForceParallelism</td>
<td>Parallelize the entire query, even if that means using high-overhead algorithms.</td>
</tr>
</tbody>
</table>
See Also

System.Linq Namespace

Send feedback on this topic to Microsoft.
Specifies the preferred type of output merge to use in a query. This is a hint only, and may not be respected by the system when parallelizing all queries.

**Namespace:** System.Linq
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Enumeration ParallelMergeOptions

C#

public enum ParallelMergeOptions
### Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Use the default merge type, which is AutoBuffered. Use a merge without output buffers. As soon as result elements have been computed, make that element available to the consumer of the query.</td>
</tr>
<tr>
<td>NotBuffered</td>
<td>Use a merge with output buffers of a size chosen by the system. Results will accumulate into an output buffer before they are available to the consumer of the query. Use a merge with full output buffers. The system will accumulate all of the results before making any of them available to the consumer of the query.</td>
</tr>
<tr>
<td>AutoBuffered</td>
<td></td>
</tr>
<tr>
<td>FullyBuffered</td>
<td></td>
</tr>
</tbody>
</table>
Remarks

Use **NotBuffered** for queries that will be consumed and output as streams, this has the lowest latency between beginning query execution and elements being yielded. For some queries, such as those involving a sort (OrderBy, OrderByDescending), buffering is essential and a hint of NotBuffered or AutoBuffered will be ignored.

Use **AutoBuffered** for most cases; this is the default. It strikes a balance between latency and overall performance.

Use **FullyBuffered** for queries when the entire output can be processed before the information is needed. This option offers the best performance when all of the output can be accumulated before yielding any information, though it is not suitable for stream processing or showing partial results mid-query.
See Also

System.Linq Namespace

Send feedback on this topic to Microsoft.
Represents a parallel sequence.

**Namespace:** System.Linq  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Class ParallelQuery
    Implements IEnumerable

C#

public class ParallelQuery : IEnumerable
Inheritance Hierarchy

System...Object
System.Linq...ParallelQuery
  System.Linq...ParallelQuery<Of <(TSource)>>()
See Also

System.Linq Namespace

Send feedback on this topic to Microsoft.
The `ParallelQuery` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IEnumerable::&lt;&gt;GetEnumerator</code></td>
<td>Returns an enumerator that iterates through the sequence.</td>
</tr>
</tbody>
</table>
See Also

ParallelQuery Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Returns an enumerator that iterates through the sequence.

**Namespace:**  System.Linq  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private Function GetEnumerator As IEnumerator
    Implements IEnumerable.GetEnumerator

C#

IEnumerator IEnumerable.GetEnumerator()

Return Value

An enumerator that iterates through the sequence.

Implements

IEnumerable...:.GetEnumerator()()}
See Also

ParallelQuery Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
Represents a parallel sequence.

**Namespace:**  [System.Linq](https://docs.microsoft.com/en-us/dotnet/api/system.linq)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Class ParallelQuery(Of TSource) _
Inherits ParallelQuery _
Implements IEnumerable(Of T), IEnumerable

C#

public class ParallelQuery<TSource> : ParallelQuery, 
IEnumerable<T>, IEnumerable
Type Parameters

TSource
Inheritance Hierarchy

System..::.Object
   System.Linq..::.ParallelQuery
      System.Linq..::.ParallelQuery<(Of <(TSource)>))
         System.Linq..::.OrderedParallelQuery<(Of <(TSource)>))
See Also

System.Linq Namespace

Send feedback on this topic to Microsoft.
The `ParallelQuery<(Of <(TSource)>)>` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetEnumerator</td>
<td>Returns an enumerator that iterates through the sequence.</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
# Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IEnumerable::GetEnumerator</code></td>
<td>Returns an enumerator that iterates through the sequence. (Inherited from <code>ParallelQuery</code>.)</td>
</tr>
</tbody>
</table>
See Also

ParallelQuery<(Of <(TSource)>)> Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
ParallelQuery(Of (Of TSource)>).GetEnumerator Method

ParallelQuery(Of (Of TSource)> Class

Namespace: System.Linq
Assembly: System.Threading (in System.Threading.dll)

Returns an enumerator that iterates through the sequence.
Syntax

Visual Basic (Declaration)

Public Overridable Function GetEnumerator As IEnumerator(Of TSource)

C#

public virtual IEnumerator<TSource> GetEnumerator()

Return Value

An enumerator that iterates through the sequence.
See Also

ParallelQuery(Of (TSource)>) Class
System.Linq Namespace

Send feedback on this topic to Microsoft.
# Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier</td>
<td>Enables multiple tasks to cooperatively work on an algorithm in parallel through multiple phases.</td>
</tr>
<tr>
<td>BarrierPostPhaseException</td>
<td>The exception that is thrown when the post-phase action of a <strong>Barrier</strong> fails.</td>
</tr>
<tr>
<td>CancellationTokenSource</td>
<td>Signals to a <strong>CancellationToken</strong> that it should be canceled.</td>
</tr>
<tr>
<td>CountdownEvent</td>
<td>Represents a synchronization primitive that is signaled when its count reaches zero.</td>
</tr>
<tr>
<td>LazyInitializer</td>
<td>Provides lazy initialization routines.</td>
</tr>
<tr>
<td>SemaphoreSlim</td>
<td>Limits the number of threads that can access a resource or pool of resources concurrently.</td>
</tr>
<tr>
<td>ThreadLocal&lt;Of &lt;(T)&gt;&gt;</td>
<td>Provides thread-local storage of data.</td>
</tr>
</tbody>
</table>
## Structures

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancellationToken</td>
<td>Propogates notification that operations should be canceled.</td>
</tr>
<tr>
<td>CancellationTokenRegistration</td>
<td>Represents a callback delegate that has been registered with a CancellationToken.</td>
</tr>
<tr>
<td>SpinLock</td>
<td>Provides a mutual exclusion lock primitive where a thread trying to acquire the lock waits in a loop repeatedly checking until the lock becomes available.</td>
</tr>
<tr>
<td>SpinWait</td>
<td>Provides support for spin-based waiting.</td>
</tr>
</tbody>
</table>
## Enumerations

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LazyThreadSafetyMode</code></td>
<td>Specifies how a <code>Lazy&lt;Of &lt;(T)&gt;)</code> instance should synchronize access among multiple threads.</td>
</tr>
</tbody>
</table>

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Enables multiple tasks to cooperatively work on an algorithm in parallel through multiple phases.

**Namespace:** [System.Threading](https://docs.microsoft.com/en-us/dotnet/api/system.threading)  
**Assembly:** System.Threading (in System.Threading.dll)
Visual Basic (Declaration)

<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading := True)> _
Public Class Barrier _
   Implements IDisposable

C#

[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading = true)]
public class Barrier : IDisposable
Remarks

A group of tasks cooperate by moving through a series of phases, where each in the group signals it has arrived at the Barrier in a given phase and implicitly waits for all others to arrive. The same Barrier can be used for multiple phases.

All public and protected members of Barrier are thread-safe and may be used concurrently from multiple threads, with the exception of Dispose, which must only be used when all other operations on the Barrier have completed.
Inheritance Hierarchy

System..::..Object
System.Threading..::..Barrier
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier(Int32)</td>
<td>Initializes a new instance of the Barrier class.</td>
</tr>
<tr>
<td>Barrier(Int32, Action&lt;Of (Barrier)&gt;))</td>
<td>Initializes a new instance of the Barrier class.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the **Barrier** class.

**Namespace:**  [System.Threading](#)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    participantCount As Integer _
)

C#

public Barrier(
    int participantCount
)

Parameters

participantCount
    Type: System..::.Int32
    The number of participating threads.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>participantCount is less than 0 or greater than MaxValue.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
Barrier Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Barrier Constructor (Int32, Action<Of <(Barrier)>>)  

Barrier Class  See Also  Send Feedback

Initializes a new instance of the Barrier class.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    participantCount As Integer, _
    postPhaseAction As Action(Of Barrier) _
)

C#

public Barrier(
    int participantCount,
    Action<Barrier> postPhaseAction
)

Parameters

participantCount
Type: System.Int32
The number of participating threads.

postPhaseAction
Type: System.Action(Of Barrier)
The Action(Of (T)->) to be executed after each phase.
Remarks

The postPhaseAction delegate will be executed after all participants have arrived at the barrier in one phase. The participants will not be released to the next phase until the postPhaseAction delegate has completed execution.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException participantCount</td>
<td>participantCount is less than 0 or greater than MaxValue.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
Barrier Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The **Barrier** type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddParticipant</td>
<td>Notifies the <a href="#">Barrier</a> that there will be an additional participant.</td>
</tr>
<tr>
<td>AddParticipants</td>
<td>Notifies the <a href="#">Barrier</a> that there will be additional participants.</td>
</tr>
<tr>
<td>Dispose</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>RemoveParticipant</td>
<td>Notifies the <a href="#">Barrier</a> that there will be one less participant.</td>
</tr>
<tr>
<td>RemoveParticipants</td>
<td>Notifies the <a href="#">Barrier</a> that there will be fewer participants.</td>
</tr>
<tr>
<td>SignalAndWait</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Notifies the `Barrier` that there will be an additional participant.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Function AddParticipant As Long

C#

public long AddParticipant()

Return Value

The phase number of the barrier in which the new participants will first participate.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.InvalidOperationException</td>
<td>Adding a participant would cause the barrier's participant count to exceed MaxValue.</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>The method was invoked from within a post-phase action.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Notifies the `Barrier` that there will be additional participants.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function AddParticipants (_
    participantCount As Integer _
) As Long
```

**C#**

```csharp
public long AddParticipants(
    int participantCount
)
```

### Parameters

`participantCount`
- Type: `System..::.Int32`
- The number of additional participants to add to the barrier.

### Return Value

The phase number of the barrier in which the new participants will first participate.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>participantCount is less than 0.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>Adding participantCount participants would cause the barrier's participant count to exceed MaxValue.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The method was invoked from within a post-phase action.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
Barrier... Dispose Method
Barrier Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose()()</td>
<td>Releases all resources used by the current instance of Barrier.</td>
</tr>
<tr>
<td>Dispose(Boolean)</td>
<td>When overridden in a derived class, releases the unmanaged resources used by the Barrier, and optionally releases the managed resources.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Barrier::Dispose Method

Barrier Class  See Also  Send Feedback

Releases all resources used by the current instance of Barrier.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Sub Dispose

C#

public void Dispose()

Implements

IDisposable...:::Dispose()()()
Remarks

Unlike most of the members of Barrier, Dispose is not thread-safe and may not be used concurrently with other members of this instance.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::InvalidOperationException</td>
<td>The method was invoked from within a post-phase action.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
When overridden in a derived class, releases the unmanaged resources used by the `Barrier`, and optionally releases the managed resources.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Protected Overridable Sub Dispose ( _
    disposing As Boolean _
)

C#

protected virtual void Dispose(  
    bool disposing
)

Parameters

disposing
    Type: System..::.Boolean
    true to release both managed and unmanaged resources; false to release
    only unmanaged resources.
Remarks

Unlike most of the members of Barrier, Dispose is not thread-safe and may not be used concurrently with other members of this instance.
See Also

Barrier Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
 Removes one participant from the Barrier.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub RemoveParticipant

C#

public void RemoveParticipant()
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::InvalidOperationException</td>
<td>The barrier already has 0 participants.</td>
</tr>
<tr>
<td>System...:::InvalidOperationException</td>
<td>The method was invoked from within a post-phase action.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Barrier Class

See Also
Send Feedback

Notifies the Barrier that there will be fewer participants.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub RemoveParticipants ( _
    participantCount As Integer _
)  

C#

public void RemoveParticipants(
    int participantCount
)  

Parameters

participantCount
    Type: System..::.Int32
    The number of additional participants to remove from the barrier.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::.ArgumentOutOfRangeException participantCount is less than 0.</td>
<td>The barrier already has 0 participants.</td>
</tr>
<tr>
<td>System::.InvalidOperationException</td>
<td>The method was invoked from within a post-phase action.</td>
</tr>
<tr>
<td>System::.InvalidOperationException</td>
<td>The current instance has already been disposed.</td>
</tr>
<tr>
<td>System::.ObjectDisposedException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
Barrier...:SignalAndWait Method
Barrier Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SignalAndWait()()</code></td>
<td>Signals that a participant has reached the <code>Barrier</code> and waits for all other participants to reach the barrier as well.</td>
</tr>
<tr>
<td><code>SignalAndWait(Int32)</code></td>
<td>Signals that a participant has reached the <code>Barrier</code> and waits for all other participants to reach the barrier, using a 32-bit signed integer to measure the time interval.</td>
</tr>
<tr>
<td><code>SignalAndWait(CancellationToken)</code></td>
<td>Signals that a participant has reached the <code>Barrier</code> and waits for all other participants to reach the barrier, while observing a <code>CancellationToken</code>.</td>
</tr>
<tr>
<td><code>SignalAndWait(TimeSpan)</code></td>
<td>Signals that a participant has reached the <code>Barrier</code> and waits for all other participants to reach the barrier as well, using a <code>TimeSpan</code> to measure the time interval.</td>
</tr>
<tr>
<td><code>SignalAndWait(Int32, CancellationToken)</code></td>
<td>Signals that a participant has reached the <code>Barrier</code> and waits for all other participants to reach the barrier as well, using a 32-bit signed integer to measure the time interval, while observing a <code>CancellationToken</code>.</td>
</tr>
<tr>
<td><code>SignalAndWait(TimeSpan, CancellationToken)</code></td>
<td>Signals that a participant has reached the <code>Barrier</code> and waits for all other participants to reach the barrier as well, using a <code>TimeSpan</code> to measure the time interval, while</td>
</tr>
</tbody>
</table>
observing a CancellationToken.
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Signals that a participant has reached the `Barrier` and waits for all other participants to reach the barrier as well.

**Namespace:** `System.Threading`

**Assembly:** `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub SignalAndWait

C#

public void SignalAndWait()
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.InvalidOperationException</td>
<td>The method was invoked from within a post-phase action, the barrier currently has 0 participants, or the barrier is being used by more threads than are registered as participants.</td>
</tr>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
SignalAndWait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Signals that a participant has reached the Barrier and waits for all other participants to reach the barrier as well, using a 32-bit signed integer to measure the time interval.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function SignalAndWait ( (millisecondsTimeout As Integer  ) As Boolean

C#

public bool SignalAndWait(
    int millisecondsTimeout
)  

Parameters

millisecondsTimeout
    Type: System..::..Int32
    The number of milliseconds to wait, or Infinite(-1) to wait indefinitely.

Return Value

true if all other participants reached the barrier; otherwise, false.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The method was invoked from within a post-phase action, the barrier currently has 0 participants, or the barrier is being used by more threads than are registered as participants.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
SignalAndWait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Signals that a participant has reached the `Barrier` and waits for all other participants to reach the barrier, while observing a `CancellationToken`.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub SignalAndWait ( _
    cancellationToken As CancellationToken _
)

C#

public void SignalAndWait(
    CancellationToken cancellationToken
)

Parameters

cancellationToken
    Type: System.Threading.:.:CancellationToken
    The CancellationToken to observe.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>The method was invoked from within a post-phase action, the barrier currently has 0 participants, or the barrier is being used by more threads than are registered as participants.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>cancellationToken has been canceled.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
SignalAndWait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Signals that a participant has reached the `Barrier` and waits for all other participants to reach the barrier as well, using a `TimeSpan` to measure the time interval.

**Namespace:** `System.Threading`

**Assembly:** `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Function SignalAndWait ( _
    timeout As TimeSpan _
) As Boolean

C#

public bool SignalAndWait(
    TimeSpan timeout
)

Parameters

timeout
    Type: System::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

true if all other participants reached the barrier; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out, or it is greater than MaxValue.</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>The method was invoked from within a post-phase action, the barrier currently has 0 participants, or the barrier is being used by more threads than are registered as participants.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
SignalAndWait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
Barrier...::.SignalAndWait Method (Int32, CancellationToken)

Barrier Class  See Also  Send Feedback

Signals that a participant has reached the barrier and waits for all other participants to reach the barrier as well, using a 32-bit signed integer to measure the time interval, while observing a CancellationToken.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function SignalAndWait ( _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public bool SignalAndWait(  
    int millisecondsTimeout,  
    CancellationToken cancellationToken
)

Parameters

millisecondsTimeout
    Type: System::Int32
    The number of milliseconds to wait, or Infinite(-1) to wait indefinitely.

cancellationToken
    Type: System.Threading::CancellationToken
    The CancellationToken to observe.

Return Value

true if all other participants reached the barrier; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentOutOfRangeException</code></td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>The method was invoked from within a post-phase action, the barrier currently has 0 participants, or the barrier is being used by more threads than are registered as participants.</td>
</tr>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>cancellationToken has been canceled.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
SignalAndWait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library
Barrier...::SignalAndWait Method (TimeSpan, CancellationToken)

Barrier Class  See Also  Send Feedback

Signals that a participant has reached the Barrier and waits for all other participants to reach the barrier as well, using a TimeSpan to measure the time interval, while observing a CancellationToken.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function SignalAndWait ( _
    timeout As TimeSpan, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public bool SignalAndWait(
    TimeSpan timeout,
    CancellationToken cancellationToken
)

Parameters

timeout
    Type: System:::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken to observe.

Return Value

true if all other participants reached the barrier; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentOutOfRangeException</code></td>
<td><code>timeout</code> is a negative number other than -1 milliseconds, which represents an infinite time-out.</td>
</tr>
<tr>
<td><code>System..::.InvalidOperationException</code></td>
<td>The method was invoked from within a post-phase action, the barrier currently has 0 participants, or the barrier is being used by more threads than are registered as participants.</td>
</tr>
<tr>
<td><code>System..::.OperationCanceledException</code></td>
<td><code>cancellationToken</code> has been canceled.</td>
</tr>
<tr>
<td><code>System..::.ObjectDisposedException</code></td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
SignalAndWait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The **Barrier** type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CurrentPhaseNumber</code></td>
<td>Gets the number of the barrier's current phase.</td>
</tr>
<tr>
<td><code>ParticipantCount</code></td>
<td>Gets the total number of participants in the barrier.</td>
</tr>
<tr>
<td><code>ParticipantsRemaining</code></td>
<td>Gets the number of participants in the barrier that haven’t yet signaled in the current phase.</td>
</tr>
</tbody>
</table>
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets the number of the barrier's current phase.

**Namespace:**  [System.Threading](https://docs.microsoft.com)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property CurrentPhaseNumber As Long

C#

public long CurrentPhaseNumber { get; }
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets the total number of participants in the barrier.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property ParticipantCount As Integer

C#

public int ParticipantCount { get; }
See Also

Barrier Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets the number of participants in the barrier that haven’t yet signaled in the current phase.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property ParticipantsRemaining As Integer

C#

public int ParticipantsRemaining { get; }
Remarks

This could be 0 during a post-phase action delegate execution or if the ParticipantCount is 0.
See Also

 Barrier Class
 System.Threading Namespace

Send feedback on this topic to Microsoft.
The exception that is thrown when the post-phase action of a Barrier fails.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute>
Public Class BarrierPostPhaseException
    Inherits Exception

C#

[SerializableAttribute]
public class BarrierPostPhaseException : Exception
Inheritance Hierarchy

System...:::Object
  System...:::Exception
    System.Threading...:::BarrierPostPhaseException
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
BarrierPostPhaseException Constructor
BarrierPostPhaseException Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarrierPostPhaseException()()</td>
<td>Initializes a new instance of the <code>BarrierPostPhaseException</code> class.</td>
</tr>
<tr>
<td>BarrierPostPhaseException(_exception)</td>
<td>Initializes a new instance of the <code>BarrierPostPhaseException</code> class with the specified inner exception.</td>
</tr>
<tr>
<td>BarrierPostPhaseException(_string)</td>
<td>Initializes a new instance of the <code>BarrierPostPhaseException</code> class with a specified error message.</td>
</tr>
<tr>
<td>BarrierPostPhaseException(_serializationInfo, _streamingContext)</td>
<td>Initializes a new instance of the <code>BarrierPostPhaseException</code> class with serialized data.</td>
</tr>
<tr>
<td>BarrierPostPhaseException(_string, _exception)</td>
<td>Initializes a new instance of the <code>BarrierPostPhaseException</code> class with a specified error message and inner exception.</td>
</tr>
</tbody>
</table>
See Also

BarrierPostPhaseException Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `BarrierPostPhaseException` class.

**Namespace:** [System.Threading](https://docs.microsoft.com/en-us/dotnet/api/system.threading)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public BarrierPostPhaseException()
See Also

BarrierPostPhaseException Class
BarrierPostPhaseException Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
BarrierPostPhaseException Constructor (Exception)

Initializes a new instance of the `BarrierPostPhaseException` class with the specified inner exception.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    innerException As Exception _
)

C#

public BarrierPostPhaseException(
    Exception innerException
)

Parameters

innerException
    Type: System::Exception
    The exception that is the cause of the current exception.
See Also

BarrierPostPhaseException Class
BarrierPostPhaseException Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
BarrierPostPhaseException Constructor (String)

Initializes a new instance of the **BarrierPostPhaseException** class with a specified error message.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    message As String _
)   

C#

public BarrierPostPhaseException(
    string message
)

Parameters

message
    Type: System::String
    A string that describes the exception.
See Also

BarrierPostPhaseException Class  
BarrierPostPhaseException Overload  
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `BarrierPostPhaseException` class with serialized data.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Protected Sub New (_
    info As SerializationInfo, _
    context As StreamingContext _
)

C#

protected BarrierPostPhaseException(
    SerializationInfo info,
    StreamingContext context
)

Parameters

info
    Type: System.Runtime.Serialization.SerializationInfo
    The object that holds the serialized object data.

context
    Type: System.Runtime.Serialization.StreamingContext
    An object that describes the source or destination of the serialized data.
See Also

BarrierPostPhaseException Class
BarrierPostPhaseException Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
BarrierPostPhaseException Constructor (String, Exception)

Initializes a new instance of the `BarrierPostPhaseException` class with a specified error message and inner exception.

Namespace: `System.Threading`
Assembly: `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    message As String, _
    innerException As Exception _
)

C#

public BarrierPostPhaseException(
    string message,
    Exception innerException
)

Parameters

message
    Type: System:::String
    A string that describes the exception.

innerException
    Type: System:::Exception
    The exception that is the cause of the current exception.
See Also

BarrierPostPhaseException Class
BarrierPostPhaseException Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The **BarrierPostPhaseException** type exposes the following members.
# Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetBaseException</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetObjectData</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Exception.)</td>
</tr>
</tbody>
</table>
See Also

BarrierPostPhaseException Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
The **BarrierPostPhaseException** type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>HelpLink</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>HResult</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>InnerException</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Message</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Source</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>StackTrace</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>TargetSite</td>
<td>(Inherited from Exception.)</td>
</tr>
</tbody>
</table>
See Also

BarrierPostPhaseException Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Propogates notification that operations should be canceled.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading := True)> _
Public Structure CancellationToken
```

**C#**

```csharp
[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading = true)]
public struct CancellationToken
```
Remarks

A CancellationToken may be created directly in an unchangeable canceled or non-canceled state using the CancellationToken's constructors. However, to have a CancellationToken that can change from a non-canceled to a canceled state, CancellationTokenSource must be used. CancellationTokenSource exposes the associated CancellationToken that may be canceled by the source through its Token property.

Once canceled, a token may not transition to a non-canceled state, and a token whose CanBeCanceled is false will never change to one that can be canceled.

All members of this struct are thread-safe and may be used concurrently from multiple threads.
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes the CancellationToken.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    canceled As Boolean _
)

C#

public CancellationToken(  
    bool canceled  
)

Parameters

canceled
    Type: System...::Boolean
    The canceled state for the token.
Remarks

Tokens created with this constructor will remain in the canceled state specified by the canceled parameter. If canceled is false, both `CanBeCanceled` and `IsCancellationRequested` will be false. If canceled is true, both `CanBeCanceled` and `IsCancellationRequested` will be true.
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `CancellationToken` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Equals</code></td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
| `Finalize`            | (Inherited from Object.) Serves as a hash function for a `CancellationToken`.
| `GetHashCode`        | (Overrides `ValueType::GetHashCode`()) |
| `GetType`             | (Inherited from Object.) |
| `MemberwiseClone`     | (Inherited from Object.) |
| `Register`            | Overloaded. Throws a `OperationCanceledException` if this token has had cancellation requested. |
| `ThrowIfCancellationRequested` | |
| `ToString`            | (Inherited from `ValueType`.) |
## Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>Determines whether two <code>CancellationToken</code> instances are equal.</td>
</tr>
<tr>
<td>Inequality</td>
<td>Determines whether two <code>CancellationToken</code> instances are not equal.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether two CancellationToken instances are equal.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Operator = ( _
    left As CancellationToken, _
    right As CancellationToken _
) As Boolean

C#

public static bool operator ==(
    CancellationToken left,
    CancellationToken right
)

Parameters

left
Type: System.Threading.CancellationToken
The first instance.

right
Type: System.Threading.CancellationToken
The second instance.

Return Value

True if the instances are equal; otherwise, false.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>An associated <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtokensource">CancellationTokenSource</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
CancellationToken...::..Equals Method
CancellationToken Structure  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals(Object)</td>
<td>Determines whether the current CancellationToken instance is equal to the specified Object. (Overrides ValueType..::..Equals(Object).)</td>
</tr>
<tr>
<td>Equals(CancellationToken) CancellationToken</td>
<td>Determines whether the current CancellationToken instance is equal to the specified token.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether the current `CancellationToken` instance is equal to the specified `Object`.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overrides Function Equals ( _
    other As Object _
) As Boolean

C#

public override bool Equals( Object other
)

Parameters

other
    Type: System::Object
    The other object to which to compare this instance.

Return Value

True if other is a CancellationToken and if the two instances are equal; otherwise, false. Two tokens are equal if they are associated with the same CancellationTokenSource or if they were both constructed from public CancellationToken constructors and their IsCancellationRequested values are equal.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ObjectDisposedException</td>
<td>An associated CancellationTokenSource has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
Equals Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether the current `CancellationToken` instance is equal to the specified token.

**Namespace:**  `System.Threading`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Function Equals ( _
    other As CancellationToken _
) As Boolean

C#

public bool Equals(
    CancellationToken other
)

Parameters

other
    Type: System.Threading.CancellationToken
    The other CancellationToken to which to compare this instance.

Return Value

True if the instances are equal; otherwise, false. Two tokens are equal if they are associated with the same CancellationTokenSource or if they were both constructed from public CancellationToken constructors and their IsCancellationRequested values are equal.
See Also

CancellationToken Structure
Equals Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Serves as a hash function for a `CancellationToken`.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overrides Function GetHashCode As Integer

C#

public override int GetHashCode()

Return Value

A hash code for the current CancellationToken instance.
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether two `CancellationToken` instances are not equal.

**Namespace:**  `System.Threading`  
**Assembly:**  `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Public Shared Operator <> ( _
    left As CancellationToken, _
    right As CancellationToken _
) As Boolean

C#

public static bool operator !=(
    CancellationToken left,
    CancellationToken right
)

Parameters

left
    Type: System.Threading.CancellationToken
    The first instance.

right
    Type: System.Threading.CancellationToken
    The second instance.

Return Value

True if the instances are not equal; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>An associated <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtokensource">CancellationTokenSource</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#  Include Protected Members  Include Inherited Members

.NET Framework Class Library

CancellationToken::Register Method

CancellationToken Structure  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register(Action)</td>
<td>Registers a delegate that will be called when this <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken">CancellationToken</a> is canceled.</td>
</tr>
<tr>
<td>Register(Action, Boolean)</td>
<td>Registers a delegate that will be called when this <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken">CancellationToken</a> is canceled.</td>
</tr>
<tr>
<td>Register(Action(Of (Object)-&gt;, Object))</td>
<td>Registers a delegate that will be called when this <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken">CancellationToken</a> is canceled.</td>
</tr>
<tr>
<td>Register(Action(Of (Object)-&gt;), Object, Boolean)</td>
<td>Registers a delegate that will be called when this <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken">CancellationToken</a> is canceled.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationToken.Register Method (Action)

Registers a delegate that will be called when this CancellationToken is canceled.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Register ( _
    callback As Action _
) As CancellationTokenRegistration

C#

public CancellationTokenRegistration Register(
    Action callback
)

Parameters

callback
    Type: System::Action
    The delegate to be executed when the CancellationToken is canceled.

Return Value

The CancellationTokenRegistration instance that can be used to deregister the callback.
Remarks

If this token is already in the canceled state, the delegate will be run immediately and synchronously. Any exception the delegate generates will be propagated out of this method call.

The current ExecutionContext, if one exists, will be captured along with the delegate and will be used when executing it.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>callback is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The associated <a href="#">CancellationTokenSource</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
Register Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#  
.NET Framework Class Library
CancellationToken..::.Register Method (Action, Boolean)

Registers a delegate that will be called when this CancellationToken is canceled.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Register ( _
    callback As Action, _
    useSynchronizationContext As Boolean _
) As CancellationTokenRegistration

C#

public CancellationTokenRegistration Register(
    Action callback,
    bool useSynchronizationContext
)

Parameters

callback
    Type: System:::Action
    The delegate to be executed when the CancellationToken is canceled.

useSynchronizationContext
    Type: System:::Boolean
    A Boolean value that indicates whether to capture the current SynchronizationContext and use it when invoking the callback.

Return Value

The CancellationTokenRegistration instance that can be used to deregister the callback.
Remarks

If this token is already in the canceled state, the delegate will be run immediately and synchronously. Any exception the delegate generates will be propagated out of this method call.

The current ExecutionContext, if one exists, will be captured along with the delegate and will be used when executing it.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>callback is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The associated <a href="#">CancellationTokenSource</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
Register Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationToken..::.Register Method (Action(Of <(Object)>), Object)

Registers a delegate that will be called when this CancellationToken is canceled.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function Register ( _
    callback As Action(Of Object), _
    state As Object _
) As CancellationTokenRegistration
```

**C#**

```csharp
public CancellationTokenRegistration Register(
    Action<Object> callback,
    Object state
)
```

### Parameters

**callback**

- **Type:** System...:::Action<(Of <(Object)>)
- The delegate to be executed when the CancellationToken is canceled.

**state**

- **Type:** System...:::Object
- The state to pass to the callback when the delegate is invoked. This may be null.

### Return Value

The CancellationTokenRegistration instance that can be used to deregister the callback.
Remarks

If this token is already in the canceled state, the delegate will be run immediately and synchronously. Any exception the delegate generates will be propagated out of this method call.

The current ExecutionContext, if one exists, will be captured along with the delegate and will be used when executing it.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>callback is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The associated <a href="#">CancellationTokenSource</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
Register Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationToken..::.Register Method (Action<Of <(Object)>>, Object, Boolean)

Registers a delegate that will be called when this CancellationToken is canceled.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Register ( _
    callback As Action(Of Object), _
    state As Object, _
    useSynchronizationContext As Boolean _
) As CancellationTokenRegistration

C#

public CancellationTokenRegistration Register(    
    Action<Object> callback,
    Object state,
    bool useSynchronizationContext    
)

Parameters

callback
    Type: System...::Action(Of <(Object)>)
    The delegate to be executed when the CancellationToken is canceled.

state
    Type: System...::Object
    The state to pass to the callback when the delegate is invoked. This may be null.

useSynchronizationContext
    Type: System...::Boolean
    A Boolean value that indicates whether to capture the current SynchronizationContext and use it when invoking the callback.

Return Value

The CancellationTokenRegistration instance that can be used to deregister the
callback.
Remarks

If this token is already in the canceled state, the delegate will be run immediately and synchronously. Any exception the delegate generates will be propagated out of this method call.

The current ExecutionContext, if one exists, will be captured along with the delegate and will be used when executing it.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The callback is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The associated <code>CancellationTokenSource</code> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
Register Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationToken...::ThrowIfCancellationRequested Method

CancellationToken Structure  See Also  Send Feedback

Throws a OperationCanceledException if this token has had cancellation requested.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub ThrowIfCancellationRequested

C#

public void ThrowIfCancellationRequested()
Remarks

This method provides functionality equivalent to:

```csharp
if (token.IsCancellationRequested)
    throw new OperationCanceledException(token);
```
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The token has had cancellation requested.</td>
</tr>
<tr>
<td></td>
<td>The associated <a href="#">CancellationTokenSource</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `CancellationToken` type exposes the following members.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CanBeCanceled</code></td>
<td>Gets whether this token is capable of being in the canceled state.</td>
</tr>
<tr>
<td><code>IsCancellationRequested</code></td>
<td>Gets whether cancellation has been requested for this token.</td>
</tr>
<tr>
<td><code>None</code></td>
<td>Returns an empty CancellationToken value.</td>
</tr>
<tr>
<td><code>WaitHandle</code></td>
<td>Gets a WaitHandle that is signaled when the token is canceled.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether this token is capable of being in the canceled state.

**Namespace:**  [System.Threading](/System.Threading)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property CanBeCanceled As Boolean

C#

public bool CanBeCanceled { get; }
Remarks

If CanBeCanceled returns false, it is guaranteed that the token will never transition into a canceled state, meaning that IsCancellationRequested will never return true.
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether cancellation has been requested for this token.

**Namespace:**  System.Threading

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsCancellationRequested As Boolean

C#

public bool IsCancellationRequested { get; }

Field Value

Whether cancellation has been requested for this token.
Remarks

This property indicates whether cancellation has been requested for this token, either through the token initially being constructed in a canceled state, or through calling `Cancel` on the token's associated `CancellationTokenSource`.

If this property is true, it only guarantees that cancellation has been requested. It does not guarantee that every registered handler has finished executing, nor that cancellation requests have finished propagating to all registered handlers. Additional synchronization may be required, particularly in situations where related objects are being canceled concurrently.
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationToken..::..None Property

Returns an empty CancellationToken value.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Public Shared ReadOnly Property None As CancellationToken

**C#**

public static CancellationToken None { get; }
Remarks

The `CancellationToken` value returned by this property will be non-cancelable by default.
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets a WaitHandle that is signaled when the token is canceled.

**Namespace:**  [System.Threading]

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property WaitHandle As WaitHandle

C#

public WaitHandle WaitHandle { get; }
Remarks

Accessing this property causes a WaitHandle to be instantiated. It is preferable to only use this property when necessary, and to then dispose the associated CancellationTokenSource instance at the earliest opportunity (disposing the source will dispose of this allocated handle). The handle should not be closed or disposed directly.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The associated <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.cancellationtokensource">CancellationTokenSource</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationToken Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Represents a callback delegate that has been registered with a `CancellationToken`.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading := True)> _
Public Structure CancellationTokenRegistration _
    Implements IComparable(Of CancellationTokenRegistration), IDisposable

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading = true)]
public struct CancellationTokenRegistration : IComparable<CancellationTokenRegistration>, IDisposable
Remarks

To unregister a callback, dispose the corresponding Registration instance.
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
The `CancellationTokenRegistration` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose</td>
<td>Disposes of the registration and unregisters the target callback from the associated CancellationToken. If the target callback is currently executing this method will wait until it completes, except in the degenerate cases where a callback method deregisters itself.</td>
</tr>
<tr>
<td>Equals</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>Serves as a hash function for a CancellationTokenRegistration. (Overides ValueType..::.GetHashCode())</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from ValueType.)</td>
</tr>
</tbody>
</table>
## Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>Determines whether two <a href="#">CancellationTokenRegistration</a> instances are equal.</td>
</tr>
<tr>
<td>Inequality</td>
<td>Determines whether two <a href="#">CancellationTokenRegistration</a> instances are not equal.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenRegistration Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Disposes of the registration and unregisters the target callback from the associated `CancellationToken`. If the target callback is currently executing this method will wait until it completes, except in the degenerate cases where a callback method deregisters itself.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub Dispose

C#

public void Dispose()

Implements

IDisposable..:::Dispose()()
See Also

CancellationTokenRegistration Structure  
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether two CancellationTokenRegistration instances are equal.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Operator = ( _
    left As CancellationTokenRegistration, _
    right As CancellationTokenRegistration _
) As Boolean
```

### C#

```csharp
public static bool operator ==(
    CancellationTokenRegistration left,
    CancellationTokenRegistration right
)
```

### Parameters

- **left**
  - Type: `System.Threading.CancellationTokenRegistration`
  - The first instance.

- **right**
  - Type: `System.Threading.CancellationTokenRegistration`
  - The second instance.

### Return Value

- True if the instances are equal; otherwise, false.
See Also

CancellationTokenRegistration Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
CancellationTokenRegistration...:::Equals Method
CancellationTokenRegistration Structure  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Equals(Object)</code></td>
<td>Determines whether the current <a href="#">CancellationTokenRegistration</a> instance is equal to the specified Object. ( Overrides ValueType::::Equals(Object).)</td>
</tr>
<tr>
<td><code>Equals(CancellationTokenRegistration)</code></td>
<td>Determines whether the current <a href="#">CancellationToken</a> instance is equal to the specified Object.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenRegistration Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether the current CancellationTokenRegistration instance is equal to the specified Object.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Overrides Function Equals ( _
obj As Object _
) As Boolean

**C#**

public override bool Equals(  
Object obj
)

**Parameters**

obj  
Type: System...:::Object  
The other object to which to compare this instance.

**Return Value**

True, if both this and obj are equal. False, otherwise. Two  
[CancellationTokenRegistration](#) instances are equal if they both refer to the output of a single call to the same Register method of a [CancellationToken](#).
See Also

CancellationTokenRegistration Structure
Equals Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether the current `CancellationToken` instance is equal to the specified `Object`.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Function Equals ( _
    other As CancellationTokenRegistration _
) As Boolean

C#

public bool Equals(
    CancellationTokenRegistration other
)

Parameters

other
    Type: System.Threading..::.CancellationTokenRegistration
    The other CancellationTokenRegistration to which to compare this instance.

Return Value

True, if both this and other are equal. False, otherwise. Two
CancellationTokenRegistration instances are equal if they both refer to the
output of a single call to the same Register method of a CancellationToken.

Implements

IEquatable<(Of <(T)>)...:..Equals(T)
See Also

CancellationTokenRegistration Structure
Equals Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationTokenRegistration..::.GetHashCode Method

Serves as a hash function for a CancellationTokenRegistration.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Overrides Function GetHashCode As Integer

**C#**

public override int GetHashCode()  

**Return Value**

A hash code for the current [CancellationTokenRegistration](#) instance.
See Also

CancellationTokenRegistration Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether two `CancellationTokenRegistration` instances are not equal.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Shared Operator <> ( _
    left As CancellationTokenRegistration, _
    right As CancellationTokenRegistration _
) As Boolean

C#

public static bool operator !=(
    CancellationTokenRegistration left,
    CancellationTokenRegistration right
)

Parameters

left
    Type: System.Threading.CancellationTokenRegistration
    The first instance.

right
    Type: System.Threading.CancellationTokenRegistration
    The second instance.

Return Value

True if the instances are not equal; otherwise, false.
See Also

CancellationTokenRegistration Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Signals to a CancellationToken that it should be canceled.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading := True)> _
Public NotInheritable Class CancellationTokenSource _
    Implements IDisposable

C#

[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading = true)]
public sealed class CancellationTokenSource : IDisposable
Remarks

CancellationTokenSource is used to instantiate a CancellationToken (via the source's Token property) that can be handed to operations that wish to be notified of cancellation or that can be used to register asynchronous operations for cancellation. That token may have cancellation requested by calling to the source's Cancel method.

All members of this class, except Dispose, are thread-safe and may be used concurrently from multiple threads.
Inheritance Hierarchy

System...:::Object
System.Threading...:::CancellationTokenSource
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
CancellationTokenSource Constructor

CancellationTokenSource Class  See Also  Send Feedback

Initializes the CancellationTokenSource.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public CancellationTokenSource()
See Also

CancellationTokenSource Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `CancellationTokenSource` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cancel</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>CreateLinkedTokenSource</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Dispose</strong></td>
<td>Releases the resources used by this CancellationTokenSource.</td>
</tr>
<tr>
<td><strong>Equals</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>Finalize</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetHashCode</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetType</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>MemberwiseClone</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>ToString</strong></td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenSource Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
CancellationTokenSource...:::Cancel Method
CancellationTokenSource Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel()()</td>
<td>Communicates a request for cancellation.</td>
</tr>
<tr>
<td>Cancel(Boolean)</td>
<td>Communicates a request for cancellation.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenSource Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationTokenSource Class

Communicates a request for cancellation.

**Namespace:** [System.Threading]

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Cancel

C#

public void Cancel()
Remarks

The associated `CancellationToken` will be notified of the cancellation and will transition to a state where `IsCancellationRequested` returns true. Any callbacks or cancelable operations registered with the `CancellationToken` will be executed.

Cancelable operations and callbacks registered with the token should not throw exceptions. However, this overload of `Cancel` will aggregate any exceptions thrown into a `AggregateException`, such that one callback throwing an exception will not prevent other registered callbacks from being executed.

The `ExecutionContext` that was captured when each callback was registered will be reestablished when the callback is invoked.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::AggregateException</td>
<td>An aggregate exception containing all the exceptions thrown by the registered callbacks on the associated CancellationToken.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>This <a href="#">CancellationTokenSource</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenSource Class
Cancel Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Communicates a request for cancellation.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Cancel ( _
    throwOnFirstException As Boolean _
)

C#

public void Cancel(
    bool throwOnFirstException
)

Parameters

throwOnFirstException
    Type: System:::Boolean
    Specifies whether exceptions should immediately propagate.
Remarks

The associated CancellationToken will be notified of the cancellation and will transition to a state where IsCancellationRequested returns true. Any callbacks or cancelable operations registered with the CancellationToken will be executed.

Cancelable operations and callbacks registered with the token should not throw exceptions. If throwOnFirstException is true, an exception will immediately propagate out of the call to Cancel, preventing the remaining callbacks and cancelable operations from being processed. If throwOnFirstException is false, this overload will aggregate any exceptions thrown into a AggregateException, such that one callback throwing an exception will not prevent other registered callbacks from being executed.

The ExecutionContext that was captured when each callback was registered will be reestablished when the callback is invoked.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.AggregateException</code></td>
<td>An aggregate exception containing all the exceptions thrown by the registered callbacks on the associated <code>CancellationToken</code>.</td>
</tr>
<tr>
<td><code>System..::.ObjectDisposedException</code></td>
<td>This <code>CancellationTokenSource</code> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenSource Class
Cancel Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
CancellationTokenSource...:::CreateLinkedTokenSource Method
CancellationTokenSource Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `CreateLinkedTokenSource(array<
  CancellationToken
  >[][])` | Creates a `CancellationToken` that will be in the canceled state when any of the source tokens are in the canceled state. |
| `CreateLinkedTokenSource(CancellationToken, CancellationToken)` | Creates a `CancellationToken` that will be in the canceled state when any of the source tokens are in the canceled state. |
See Also

CancellationTokenSource Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationTokenSource..::.CreateLinkedTokenSource Method
(array<CancellationToken>[][])
### Syntax

**Visual Basic (Declaration)**

```
Public Shared Function CreateLinkedTokenSource ( _
    ParamArray tokens As CancellationToken() _
) As CancellationTokenSource
```

**C#**

```
public static CancellationTokenSource CreateLinkedTokenSource(  
    params CancellationToken[] tokens  
)
```

**Parameters**

tokens

Type: array< System.Threading.CancellationToken >[]

The CancellationToken instances to observe.

**Return Value**

A CancellationTokenSource that is linked to the source tokens.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.::.ArgumentNullException</td>
<td>tokens is null.</td>
</tr>
<tr>
<td>System.::.ObjectDisposedException</td>
<td>A <a href="https://docs.microsoft.com/en-us/dotnet/api/system.cancellationtokensource">CancellationTokenSource</a> associated with one of the source tokens has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenSource Class
CreateLinkedTokenSource Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Creates a [CancellationTokenSource](#) that will be in the canceled state when any of the source tokens are in the canceled state.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function CreateLinkedTokenSource ( _
    token1 As CancellationToken, _
    token2 As CancellationToken _
) As CancellationTokenSource

C#

public static CancellationTokenSource CreateLinkedTokenSource(
    CancellationToken token1,
    CancellationToken token2
)

Parameters

token1
    Type: System.Threading.CancellationToken
    The first CancellationToken to observe.

token2
    Type: System.Threading.CancellationToken
    The second CancellationToken to observe.

Return Value

A CancellationTokenSource that is linked to the source tokens.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectDisposedException</td>
<td>A <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtokensource">CancellationTokenSource</a> associated with one of the source tokens has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenSource Class
CreateLinkedTokenSource Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
CancellationTokenSource Class

CancellationTokenSource..::.Dispose Method

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)

Releases the resources used by this CancellationTokenSource.
Syntax

Visual Basic (Declaration)

Public Sub Dispose

C#

public void Dispose()

Implements

IDisposable...:::Dispose()()
Remarks

This method is not thread-safe for any other concurrent calls.
See Also

CancellationTokenSource Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `CancellationTokenSource` type exposes the following members.
# Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsCancellationRequested</td>
<td>Gets whether cancellation has been requested for this <a href="#">CancellationTokenSource</a>.</td>
</tr>
<tr>
<td>Token</td>
<td>Gets the <a href="#">CancellationToken</a> associated with this <a href="#">CancellationTokenSource</a>.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenSource Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether cancellation has been requested for this [CancellationTokenSource](https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtokensource).  

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Public ReadOnly Property IsCancellationRequested As Boolean

**C#**

public bool IsCancellationRequested { get; }

**Field Value**

Whether cancellation has been requested for this [CancellationTokenSource](#).
Remarks

This property indicates whether cancellation has been requested for this token source, such as due to a call to its Cancel method.

If this property returns true, it only guarantees that cancellation has been requested. It does not guarantee that every handler registered with the corresponding token has finished executing, nor that cancellation requests have finished propagating to all registered handlers. Additional synchronization may be required, particularly in situations where related objects are being canceled concurrently.
See Also

CancellationTokenSource Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets the `CancellationToken` associated with this `CancellationTokenSource`.

**Namespace:**  System.Threading
**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public ReadOnly Property Token As CancellationToken

**C#**

```csharp
public CancellationToken Token { get; }
```

**Field Value**

The [CancellationToken](#) associated with this [CancellationTokenSource](#).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The token source has been disposed.</td>
</tr>
</tbody>
</table>
See Also

CancellationTokenSource Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Represents a synchronization primitive that is signaled when its count reaches zero.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading := True)> _
Public Class CountdownEvent _
   Implements IDisposable

C#

[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading = true)]
public class CountdownEvent : IDisposable
Remarks

All public and protected members of CountdownEvent are thread-safe and may be used concurrently from multiple threads, with the exception of Dispose, which must only be used when all other operations on the CountdownEvent have completed, and Reset, which should only be used when no other threads are accessing the event.
Inheritance Hierarchy

System..::.Object
    System.Threading..::.CountdownEvent
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of `CountdownEvent` class with the specified count.

**Namespace:**  System.Threading

**Assembly:**  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

Public Sub New (_
    initialCount As Integer _
)

**C#**

public CountdownEvent(
    int initialCount
)

### Parameters

initialCount
- Type: System:::Int32
- The number of signals required to set the CountdownEvent.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>initialCount is less than 0.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `CountdownEvent` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddCount</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Dispose</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Reset</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Signal</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>TryAddCount</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Wait</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
  □  Include Protected Members
  □  Include Inherited Members
.NET Framework Class Library
CountdownEvent:::AddCount Method
CountdownEvent Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AddCount()</code>()</td>
<td>Increments the <code>CountdownEvent</code>'s current count by one.</td>
</tr>
<tr>
<td><code>AddCount(Int32)</code></td>
<td>Increments the <code>CountdownEvent</code>'s current count by a specified value.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Increments the `CountdownEvent`'s current count by one.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

Public Sub AddCount

**C#**

public void AddCount()
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The current instance is already set.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>CurrentCount is equal to MaxValue.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
AddCount Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Increments the `CountdownEvent`'s current count by a specified value.

**Namespace:** System.Threading
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub AddCount (_
    signalCount As Integer _
)

C#

public void AddCount(
    int signalCount
)

Parameters

signalCount
  Type: System..::.Int32
  The value by which to increase CurrentCount.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException signalCount</td>
<td>signalCount is less than 0.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The current instance is already set.</td>
</tr>
<tr>
<td>System:::InvalidOperationException CurrentCount</td>
<td>CurrentCount is equal to MaxValue.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
AddCount Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
CountdownEvent dispose method

See Also

Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose()()</td>
<td>Releases all resources used by the current instance of <code>CountdownEvent</code>.</td>
</tr>
<tr>
<td>Dispose(Boolean)</td>
<td>When overridden in a derived class, releases the unmanaged resources used by the <code>CountdownEvent</code>, and optionally releases the managed resources.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
CountdownEvent..::.Dispose Method

Releases all resources used by the current instance of CountdownEvent.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

Public Sub Dispose

### C#

public void Dispose()

### Implements

IDisposable...:::Dispose()()
Remarks

Unlike most of the members of `CountdownEvent`, `Dispose()` is not thread-safe and may not be used concurrently with other members of this instance.
See Also

CountdownEvent Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
When overridden in a derived class, releases the unmanaged resources used by the CountdownEvent, and optionally releases the managed resources.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Overridable Sub Dispose ( _
    disposing As Boolean _
)

C#

protected virtual void Dispose(
    bool disposing
)

Parameters

disposing
    Type: System..::.Boolean
    true to release both managed and unmanaged resources; false to release
    only unmanaged resources.
Remarks

Unlike most of the members of CountdownEvent, Dispose() is not thread-safe and may not be used concurrently with other members of this instance.
See Also

CountdownEvent Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
CountdownEvent Class
See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset()()</td>
<td>Resets the <code>CurrentCount</code> to the value of <code>InitialCount</code>.</td>
</tr>
<tr>
<td>Reset(Int32)</td>
<td>Resets the <code>CurrentCount</code> to a specified value.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
CountdownEvent...::Reset Method

Resets the **CurrentCount** to the value of **InitialCount**.

**Namespace:**  **System.Threading**
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Reset

C#

public void Reset()
Remarks

Unlike most of the members of CountdownEvent, Reset is not thread-safe and may not be used concurrently with other members of this instance.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Reset Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Resets the `CurrentCount` to a specified value.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Reset ( _
    count As Integer _
)

C#

public void Reset(
    int count
)

Parameters

count
    Type: System:::Int32
    The number of signals required to set the CountdownEvent.
Remarks

Unlike most of the members of CountdownEvent, Reset is not thread-safe and may not be used concurrently with other members of this instance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>count is less than 0.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Reset Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#  Include Protected Members  Include Inherited Members
.NET Framework Class Library  CountdownEvent:::Signal Method
CountdownEvent Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal()()</td>
<td>Registers a signal with the <a href="#">CountdownEvent</a>, decrementing its count.</td>
</tr>
<tr>
<td>Signal(Int32)</td>
<td>Registers multiple signals with the <a href="#">CountdownEvent</a>, decrementing its count by the specified amount.</td>
</tr>
</tbody>
</table>
See Also

[CountdownEvent Class](#)
[System.Threading Namespace](#)

Send [feedback](#) on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
CountdownEvent..::..Signal Method

CountdownEvent Class  See Also  Send Feedback

Registers a signal with the CountdownEvent, decrementing its count.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Signal As Boolean

C#

public bool Signal()

Return Value

true if the signal caused the count to reach zero and the event was set; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.InvalidOperationException</td>
<td>The current instance is already set.</td>
</tr>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Signal Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Registers multiple signals with the `CountdownEvent`, decrementing its count by the specified amount.

**Namespace:**  [System.Threading](https://docs.microsoft.com/en-us/dotnet/api/system.threading)

**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function Signal (_
    signalCount As Integer _
) As Boolean
```

**C#**

```csharp
public bool Signal(
    int signalCount
)
```

**Parameters**

`signalCount`
- Type: `System.Int32`
- The number of signals to register.

**Return Value**

true if the signals caused the count to reach zero and the event was set; otherwise, false.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.InvalidOperationException</td>
<td>The current instance is already set. - or- Or signalCount is greater than</td>
</tr>
<tr>
<td></td>
<td>CurrentCount</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>signalCount is less than 1.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Signal Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
CountdownEvent...::TryAddCount Method
CountdownEvent Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TryAddCount()</code></td>
<td>Attempts to increment the <em>CountdownEvent</em>’s current count by one.</td>
</tr>
<tr>
<td><code>TryAddCount(Int32)</code></td>
<td>Attempts to increment the <em>CountdownEvent</em>’s current count by a specified value.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Attempts to increment the `CountdownEvent`'s current count by one.

**Namespace:**  [System.Threading](https://msdn.microsoft.com/en-us/library/system.threading.aspx)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TryAddCount As Boolean

C#

public bool TryAddCount() 

Return Value

true if the increment succeeded; otherwise, false. If CurrentCount is already at zero. this will return false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td><code>CurrentCount</code> is equal to <code>MaxValue</code>.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
TryAddCount Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Attempts to increment the `CountdownEvent`'s current count by a specified value.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

```vbnet
Public Function TryAddCount ( signalCount As Integer ) As Boolean
```

C#

```csharp
public bool TryAddCount( int signalCount )
```

Parameters

signalCount
- Type: System.Int32
- The value by which to increase `CurrentCount`.

Return Value

true if the increment succeeded; otherwise, false. If `CurrentCount` is already at zero this will return false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentOutOfRangeException signalCount is less than 0.</td>
<td></td>
</tr>
<tr>
<td>System...::.InvalidOperationException</td>
<td>The current instance is already set.</td>
</tr>
<tr>
<td>System...::.InvalidOperationException</td>
<td><strong>CurrentCount</strong> is equal to MaxValue.</td>
</tr>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
TryAddCount Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
CountdownEvent...::Wait Method
CountdownEvent Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wait()</strong></td>
<td>Blocks the current thread until the &lt;a href=&quot;#CountdownEvent&quot;&gt;CountdownEvent&lt;/a&gt; is set.</td>
</tr>
<tr>
<td><strong>Wait(Int32)</strong></td>
<td>Blocks the current thread until the &lt;a href=&quot;#CountdownEvent&quot;&gt;CountdownEvent&lt;/a&gt; is set, using a 32-bit signed integer to measure the time interval.</td>
</tr>
<tr>
<td><strong>Wait(CancellationToken)</strong></td>
<td>Blocks the current thread until the &lt;a href=&quot;#CountdownEvent&quot;&gt;CountdownEvent&lt;/a&gt; is set, while observing a &lt;a href=&quot;#CancellationToken&quot;&gt;CancellationToken&lt;/a&gt;.</td>
</tr>
<tr>
<td><strong>Wait(TimeSpan)</strong></td>
<td>Blocks the current thread until the &lt;a href=&quot;#CountdownEvent&quot;&gt;CountdownEvent&lt;/a&gt; is set, using a &lt;a href=&quot;#TimeSpan&quot;&gt;TimeSpan&lt;/a&gt; to measure the time interval.</td>
</tr>
<tr>
<td><strong>Wait(Int32, CancellationToken)</strong></td>
<td>Blocks the current thread until the &lt;a href=&quot;#CountdownEvent&quot;&gt;CountdownEvent&lt;/a&gt; is set, using a 32-bit signed integer to measure the time interval, while observing a &lt;a href=&quot;#CancellationToken&quot;&gt;CancellationToken&lt;/a&gt;.</td>
</tr>
<tr>
<td><strong>Wait(TimeSpan, CancellationToken)</strong></td>
<td>Blocks the current thread until the &lt;a href=&quot;#CountdownEvent&quot;&gt;CountdownEvent&lt;/a&gt; is set, using a &lt;a href=&quot;#TimeSpan&quot;&gt;TimeSpan&lt;/a&gt; to measure the time interval, while observing a &lt;a href=&quot;#CancellationToken&quot;&gt;CancellationToken&lt;/a&gt;.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the `CountdownEvent` is set.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub Wait

C#

public void Wait()
Remarks

The caller of this method blocks indefinitely until the current instance is set. The caller will return immediately if the event is currently in a set state.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the CountdownEvent is set, using a 32-bit signed integer to measure the time interval.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    millisecondsTimeout As Integer _
) As Boolean

C#

public bool Wait(
    int millisecondsTimeout
)

Parameters

millisecondsTimeout

Type: System::Int32

The number of milliseconds to wait, or Infinite(-1) to wait indefinitely.

Return Value

true if the CountdownEvent was set; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class  
Wait Overload  
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the *CountdownEvent* is set, while observing a *CancellationToken*.

**Namespace:**  [System.Threading](System.Threading)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Wait ( _
    cancellationToken As CancellationToken _
)

C#

public void Wait( 
    CancellationToken cancellationToken
)

Parameters

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken to observe.
Remarks

The caller of this method blocks indefinitely until the current instance is set. The caller will return immediately if the event is currently in a set state. If the CancellationToken being observed is canceled during the wait operation, an OperationCanceledException will be thrown.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::OperationCanceledException</td>
<td>cancellationToken has been canceled.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the CountdownEvent is set, using a TimeSpan to measure the time interval.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    timeout As TimeSpan _
) As Boolean

C#

public bool Wait(
    TimeSpan timeout
)

Parameters

timeout
    Type: System::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

true if the CountdownEvent was set; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out or timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the CountdownEvent is set, using a 32-bit signed integer to measure the time interval, while observing a CancellationToken.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public bool Wait(
    int millisecondsTimeout,
    CancellationToken cancellationToken
)

Parameters

millisecondsTimeout
  Type: System:::Int32
  The number of milliseconds to wait, or Infinite(-1) to wait indefinitely.

cancellationToken
  Type: System.Threading:::CancellationToken
  The CancellationToken to observe.

Return Value

true if the CountdownEvent was set; otherwise, false.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
<td></td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>cancellationToken has been canceled.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the `CountdownEvent` is set, using a `TimeSpan` to measure the time interval, while observing a `CancellationToken`.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    timeout As TimeSpan, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public bool Wait(  
    TimeSpan timeout,  
            CancellationToken cancellationToken
)

Parameters

timeout
   Type: System...:::TimeSpan
   A TimeSpan that represents the number of milliseconds to wait, or a
   TimeSpan that represents -1 milliseconds to wait indefinitely.

cancellationToken
   Type: System.Threading...:::CancellationToken
   The CancellationToken to observe.

Return Value

true if the CountdownEvent was set; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out or timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>cancellationToken has been canceled.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `CountdownEvent` type exposes the following members.
# Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentCount</td>
<td>Gets the number of remaining signals required to set the event.</td>
</tr>
<tr>
<td>InitialCount</td>
<td>Gets the numbers of signals initially required to set the event.</td>
</tr>
<tr>
<td>IsSet</td>
<td>Determines whether the event is set.</td>
</tr>
<tr>
<td>WaitHandle</td>
<td>Gets a WaitHandle that is used to wait for the event to be set.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets the number of remaining signals required to set the event.

**Namespace:**  System.Threading

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property CurrentCount As Integer

C#

public int CurrentCount { get; }
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
CountdownEvent..::.InitialCount Property

Gets the numbers of signals initially required to set the event.

**Namespace:** [System.Threading](#)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property InitialCount As Integer

C#

public int InitialCount { get; }

Field Value

The number of signals initially required to set the event.
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Determines whether the event is set.

**Namespace:**  [System.Threading](#)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsSet As Boolean

C#

public bool IsSet { get; }

Field Value

true if the event is set; otherwise, false.
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets a WaitHandle that is used to wait for the event to be set.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property WaitHandle As WaitHandle

C#

public WaitHandle WaitHandle { get; }

Field Value

A WaitHandle that is used to wait for the event to be set.
Remarks

WaitHandle should only be used if it's needed for integration with code bases that rely on having a WaitHandle. If all that's needed is to wait for the CountdownEvent to be set, the Wait() method should be preferred.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

CountdownEvent Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Provides lazy initialization routines.

**Namespace:**  [System.Threading](https://docs.microsoft.com/en-us/dotnet/api/system.threading)

**Assembly:**  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading := True)> _
Public NotInheritable Class LazyInitializer
```

**C#**

```csharp
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading = true)]
public static class LazyInitializer
```
Remarks

These routines avoid needing to allocate a dedicated, lazy-initialization instance, instead using references to ensure targets have been initialized as they are accessed.
Inheritance Hierarchy

System...:::Object
System.Threading...:::LazyInitializer
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic

C#

Include Protected Members

Include Inherited Members

.NET Framework Class Library

LazyInitializer Methods

LazyInitializer Class  See Also  Send Feedback
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnsureInitialized</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

LazyInitializer Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
LazyInitializer Class

See Also
Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnsureInitialized&lt;(Of &lt;(T)&gt;)(T%)&gt;</td>
<td>Initializes a target reference type with the type's default constructor if the target has not already been initialized.</td>
</tr>
<tr>
<td>EnsureInitialized&lt;(Of &lt;(T)&gt;)(T%, Func&lt;(Of &lt;(T)&gt;))&gt;</td>
<td>Initializes a target reference type using the specified function if it has not already been initialized.</td>
</tr>
<tr>
<td>EnsureInitialized&lt;(Of &lt;(T)&gt;)(T%, Boolean%, Object%)&gt;</td>
<td>Initializes a target reference or value type with its default constructor if it has not already been initialized.</td>
</tr>
<tr>
<td>EnsureInitialized&lt;(Of &lt;(T)&gt;)(T%, Boolean%, Object%, Func&lt;(Of &lt;(T)&gt;))&gt;</td>
<td>Initializes a target reference or value type with a specified function if it has not already been initialized.</td>
</tr>
</tbody>
</table>
See Also

LazyInitializer Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
LazyInitializer....EnsureInitialized<(Of <(T)>)> Method (T%)

LazyInitializer Class  See Also  Send Feedback

Initializes a target reference type with the type's default constructor if the target has not already been initialized.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function EnsureInitialized(Of T As Class) ( _
    ByVal target As T _
) As T

C#

public static T EnsureInitialized<T>(
    ref T target
)
where T : class

Parameters

target
    Type: T %
    A reference of type T to initialize if it has not already been initialized.
Type Parameters

T

The reference type of the reference to be initialized.

Return Value

The initialized reference of type T.
Remarks

This method may only be used on reference types. To ensure initialization of value types, see other overloads of EnsureInitialized.

This method may be used concurrently by multiple threads to initialize target. In the event that multiple threads access this method concurrently, multiple instances of T may be created, but only one will be stored into target. In such an occurrence, this method will not dispose of the objects that were not stored. If such objects must be disposed, it is up to the caller to determine if an object was not used and to then dispose of the object appropriately.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::MissingMemberException</td>
<td>Type T does not have a default constructor.</td>
</tr>
<tr>
<td>System:::MemberAccessException</td>
<td>Permissions to access the constructor of type T were missing.</td>
</tr>
</tbody>
</table>
See Also

LazyInitializer Class
EnsureInitialized Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
LazyInitializer....EnsureInitialized<(Of <(T)>)> Method (T%, Func<(Of <(T)>)>))

LazyInitializer Class  See Also  Send Feedback

Initializes a target reference type using the specified function if it has not already been initialized.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function EnsureInitialized(Of T As Class) ( _
     ByVal target As T, _
     valueFactory As Func(Of T) _
) As T

C#

g public static T EnsureInitialized<T>(
     ref T target,
     Func<T> valueFactory
)
where T : class

Parameters

target
   Type: T %
   The reference of type T to initialize if it has not already been initialized.

valueFactory
   Type: System::Func(Of (T)>)
   The Func(Of (T)>) invoked to initialize the reference.
Type Parameters

T

The reference type of the reference to be initialized.

Return Value

The initialized reference of type T.
Remarks

This method may only be used on reference types, and valueFactory may not return a null reference (Nothing in Visual Basic). To ensure initialization of value types or to allow null reference types, see other overloads of EnsureInitialized.

This method may be used concurrently by multiple threads to initialize target. In the event that multiple threads access this method concurrently, multiple instances of T may be created, but only one will be stored into target. In such an occurrence, this method will not dispose of the objects that were not stored. If such objects must be disposed, it is up to the caller to determine if an object was not used and to then dispose of the object appropriately.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::.MissingMemberException</td>
<td>Type T does not have a default constructor.</td>
</tr>
<tr>
<td>System:::.InvalidOperationException</td>
<td>valueFactory returned null.</td>
</tr>
</tbody>
</table>
See Also

LazyInitializer Class
EnsureInitialized Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
LazyInitializer....EnsureInitialized<(Of <(T)>)> Method (T%, Boolean%, Object%)  

LazyInitializer Class  See Also  Send Feedback

Initializes a target reference or value type with its default constructor if it has not already been initialized.

Namespace:  System.Threading  
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function EnsureInitialized(Of T) ( _
    ByRef target As T, _
    ByRef initialized As Boolean, _
    ByRef syncLock As Object _
) As T

C#

public static T EnsureInitialized<T>(
    ref T target,
    ref bool initialized,
    ref Object syncLock
)

Parameters

target
    Type: T %
    A reference or value of type T to initialize if it has not already been initialized.

initialized
    Type: System:::Boolean %
    A reference to a boolean that determines whether the target has already been initialized.

syncLock
    Type: System:::Object %
    A reference to an object used as the mutually exclusive lock for initializing target.
Type Parameters

T
The type of the reference to be initialized.

Return Value

The initialized value of type T.
See Also

LazyInitializer Class
EnsureInitialized Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
LazyInitializer....EnsureInitialized<(Of <(T)>)> Method (T%, Boolean%, Object%, Func<(Of <(T)>)>)

LazyInitializer Class  See Also  Send Feedback

Initializes a target reference or value type with a specified function if it has not already been initialized.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
# Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function EnsureInitialized(Of T)
    ( _
        ByVal target As T, _
        ByVal initialized As Boolean, _
        ByVal syncLock As Object, _
        ByVal valueFactory As Func(Of T))
    As T
End Function
```

### C#

```csharp
public static T EnsureInitialized<T>(
    ref T target,
    ref bool initialized,
    ref Object syncLock,
    Func<T> valueFactory
)
```

## Parameters

**target**
- **Type:** T %
- A reference or value of type T to initialize if it has not already been initialized.

**initialized**
- **Type:** System::::Boolean %
- A reference to a boolean that determines whether the target has already been initialized.

**syncLock**
- **Type:** System::::Object %
- A reference to an object used as the mutually exclusive lock for initializing target.

**valueFactory**
Type: System::Func<(Of <(T)>)
The Func<(Of <(T)>)> invoked to initialize the reference or value.
Type Parameters

T
The type of the reference to be initialized.

Return Value

The initialized value of type T.
See Also

LazyInitializer Class
EnsureInitialized Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Specifies how a Lazy(Of T) instance should synchronize access among multiple threads.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Enumeration LazyThreadSafetyMode

C#

public enum LazyThreadSafetyMode
## Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>This mode makes no guarantees around the thread-safety of the Lazy(Of (T)&gt;) instance. If used from multiple threads, the behavior of the Lazy(Of (T)&gt;) is undefined. This mode should be used when a Lazy(Of (T)&gt;) is guaranteed to never be initialized from more than one thread simultaneously and high performance is crucial. If valueFactory throws an exception when the Lazy(Of (T)&gt;) is initialized, the exception will be cached and returned on subsequent accesses to Value. Also, if valueFactory recursively accesses Value on this Lazy(Of (T)&gt;) instance, aInvalidOperationException will be thrown. When multiple threads attempt to simultaneously initialize a Lazy(Of (T)&gt;) instance, this mode allows each thread to execute the valueFactory but only the first thread to complete initialization will be allowed to set the final value of the Lazy(Of (T)&gt;). Once initialized successfully, any future calls to Value will return the cached result. If valueFactory throws an exception on any thread, that exception will be propagated out of Value. If any thread executes valueFactory without throwing an exception and, therefore, successfully sets the value, that value will be returned on subsequent accesses to Value from any thread. If no thread succeeds in setting the value, IsValueCreated will remain false and subsequent accesses to Value will result in</td>
</tr>
<tr>
<td>PublicationOnly</td>
<td></td>
</tr>
</tbody>
</table>
the valueFactory delegate re-executing. Also, if valueFactory recursively accesses Value on this Lazy<Of <(T)>> instance, an exception will NOT be thrown.

This mode uses locks to ensure that only a single thread can initialize a Lazy<Of <(T)>> instance in a thread-safe manner. In general, taken if this mode is used in conjunction with a Lazy<Of <(T)>> valueFactory delegate that uses locks internally, a deadlock can occur if not handled carefully. If valueFactory throws an exception when the Lazy<Of <(T)>> is initialized, the exception will be cached and returned on subsequent accesses to Value. Also, if valueFactory recursively accesses Value on this Lazy<Of <(T)>> instance, a InvalidOperationException will be thrown.
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Provides a slimmed down version of ManualResetEvent.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
    ExternalThreading := True)> _
Public Class ManualResetEventSlim _
    Implements IDisposable

C#

[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
    ExternalThreading = true)]
public class ManualResetEventSlim : IDisposable
Remarks

All public and protected members of ManualResetEventSlim are thread-safe and may be used concurrently from multiple threads, with the exception of Dispose, which must only be used when all other operations on the ManualResetEventSlim have completed, and Reset, which should only be used when no other threads are accessing the event.
Inheritance Hierarchy

System..::..Object
System.Threading..::..ManualResetEventSlim
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ManualResetEventSlim Constructor
ManualResetEventSlim Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManualResetEventSlim(Boolean)</td>
<td>Initializes a new instance of the <code>ManualResetEventSlim</code> class with a <code>Boolean</code> value indicating whether to set the initial state to signaled.</td>
</tr>
<tr>
<td>ManualResetEventSlim(Boolean, Int32)</td>
<td>Initializes a new instance of the <code>ManualResetEventSlim</code> class with a <code>Boolean</code> value indicating whether to set the initial state to signaled and a specified spin count.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ManualResetEventSlim` class with an initial state of nonsignaled.

**Namespace:**  [System.Threading]

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public ManualResetEventSlim()
See Also

ManualResetEventSlim Class
ManualResetEventSlim Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ManualResetEventSlim` class with a Boolean value indicating whether to set the initial state to signaled.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    initialState As Boolean _
)  

C#

public ManualResetEventSlim(  
    bool initialState
)

Parameters

initialState
    Type: System..::.Boolean
    true to set the initial state signaled; false to set the initial state to nonsignaled.
See Also

ManualResetEventSlim Class
ManualResetEventSlim Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ManualResetEventSlim` class with a Boolean value indicating whether to set the initial state to signaled and a specified spin count.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Sub New (_
    initialState As Boolean, _
    spinCount As Integer _
)
```

#### C#

```csharp
public ManualResetEventSlim(
    bool initialState,
    int spinCount
)
```

### Parameters

- **initialState**
  - Type: System..::.Boolean
  - true to set the initial state to signaled; false to set the initial state to nonsignaled.

- **spinCount**
  - Type: System..::.Int32
  - The number of spin waits that will occur before falling back to a true wait.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>spinCount is less than 0 or greater than the maximum allowed value.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
ManualResetEventSlim Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `ManualResetEventSlim` type exposes the following members.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Reset</td>
<td>Sets the state of the event to nonsignaled, which causes threads to block.</td>
</tr>
<tr>
<td>Set</td>
<td>Sets the state of the event to signaled, which allows one or more threads waiting on the event to proceed.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Wait</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose()</td>
<td>Releases all resources used by the current instance of <code>ManualResetEventSlim</code>.</td>
</tr>
<tr>
<td>Dispose(Boolean)</td>
<td>When overridden in a derived class, releases the unmanaged resources used by the <code>ManualResetEventSlim</code>, and optionally releases the managed resources.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
ManualResetEventSlim Class

Dispose Method

Releases all resources used by the current instance of ManualResetEventSlim.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Sub Dispose

C#

public void Dispose()

Implements

IDisposable...:::Dispose()()
Remarks

Unlike most of the members of ManualResetEventSlim, Dispose() is not thread-safe and may not be used concurrently with other members of this instance.
See Also

ManualResetEventSlim Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
When overridden in a derived class, releases the unmanaged resources used by the `ManualResetEventSlim`, and optionally releases the managed resources.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Overridable Sub Dispose ( _
    disposing As Boolean _
)

C#

protected virtual void Dispose(
    bool disposing
)

Parameters

disposing
    Type: System..::.Boolean
    true to release both managed and unmanaged resources; false to release
    only unmanaged resources.
Remarks

Unlike most of the members of ManualResetEventSlim, Dispose(Boolean) is not thread-safe and may not be used concurrently with other members of this instance.
See Also

ManualResetEventSlim Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Sets the state of the event to nonsignaled, which causes threads to block.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Reset

C#

public void Reset()
Remarks

Unlike most of the members of ManualResetEventSlim, Reset() is not thread-safe and may not be used concurrently with other members of this instance.
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Sets the state of the event to signaled, which allows one or more threads waiting on the event to proceed.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Set

C#

public void Set()
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ManualResetEventSlim...:Wait Method

ManualResetEventSlim Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait()()</td>
<td>Blocks the current thread until the current ManualResetEventSlim is set.</td>
</tr>
<tr>
<td>Wait(Int32)</td>
<td>Blocks the current thread until the current ManualResetEventSlim is set, using a 32-bit signed integer to measure the time interval.</td>
</tr>
<tr>
<td>Wait(CancellationToken)</td>
<td>Blocks the current thread until the current ManualResetEventSlim receives a signal, while observing a CancellationToken.</td>
</tr>
<tr>
<td>Wait(TimeSpan)</td>
<td>Blocks the current thread until the current ManualResetEventSlim is set, using a TimeSpan to measure the time interval.</td>
</tr>
<tr>
<td>Wait(Int32, CancellationToken)</td>
<td>Blocks the current thread until the current ManualResetEventSlim is set, using a 32-bit signed integer to measure the time interval, while observing a CancellationToken.</td>
</tr>
<tr>
<td>Wait(TimeSpan, CancellationToken)</td>
<td>Blocks the current thread until the current ManualResetEventSlim is set, using a TimeSpan to measure the time interval, while observing a CancellationToken.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
ManualResetEventSlim...:Wait Method
ManualResetEventSlim Class  See Also  Send Feedback

Blocks the current thread until the current ManualResetEventSlim is set.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Wait

C#

public void Wait()
Remarks

The caller of this method blocks indefinitely until the current instance is set. The caller will return immediately if the event is currently in a set state.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::InvalidOperationException</td>
<td>The maximum number of waiters has been exceeded.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the current `ManualResetEventSlim` is set, using a 32-bit signed integer to measure the time interval.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    millisecondsTimeout As Integer _
) As Boolean

C#

public bool Wait(
    int millisecondsTimeout
)

Parameters

millisecondsTimeout
    Type: System....Int32
    The number of milliseconds to wait, or Infinite(-1) to wait indefinitely.

Return Value

true if the ManualResetEventSlim was set; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentOutOfRangeException</code></td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td><code>System:::InvalidOperationException</code></td>
<td>The maximum number of waiters has been exceeded.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
ManualResetEventSlim...::Wait Method (CancellationToken)

Blocks the current thread until the current ManualResetEventSlim receives a signal, while observing a CancellationToken.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Wait ( _
cancellationToken As CancellationToken _
)

C#

public void Wait(
    CancellationToken cancellationToken
)

Parameters

cancellationToken
    Type: System.Threading...:::CancellationToken
    The CancellationToken to observe.
Remarks

The caller of this method blocks indefinitely until the current instance is set. The caller will return immediately if the event is currently in a set state.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::InvalidOperationException</td>
<td>The maximum number of waiters has been exceeded.</td>
</tr>
<tr>
<td>OperationCanceledException</td>
<td>cancellationToken was canceled.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the current ManualResetEventSlim is set, using a TimeSpan to measure the time interval.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    timeout As TimeSpan _
) As Boolean

C#

public bool Wait(
    TimeSpan timeout
)

Parameters

timeout
    Type: System::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

true if the ManualResetEventSlim was set; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out - or - timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The maximum number of waiters has been exceeded.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
ManualResetEventSlim::Wait Method (Int32, CancellationToken)

ManualResetEventSlim Class  See Also  Send Feedback

Blocks the current thread until the current ManualResetEventSlim is set, using a 32-bit signed integer to measure the time interval, while observing a CancellationToken.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function Wait ( _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Boolean
```

**C#**

```csharp
public bool Wait(
    int millisecondsTimeout,
    CancellationToken cancellationToken
)
```

### Parameters

**millisecondsTimeout**
- **Type:** System:::Int32
- The number of milliseconds to wait, or Infinite(-1) to wait indefinitely.

**cancellationToken**
- **Type:** System.Threading:::CancellationToken
- The CancellationToken to observe.

### Return Value

true if the ManualResetEventSlim was set; otherwise, false.
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentOutOfRangeException</code></td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td><code>System..::.InvalidOperationException</code></td>
<td>The maximum number of waiters has been exceeded.</td>
</tr>
<tr>
<td><code>OperationCanceledException</code></td>
<td>cancellationToken was canceled.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until the current ManualResetEventSlim is set, using a TimeSpan to measure the time interval, while observing a CancellationToken.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

Public Function Wait ( _
    timeout As TimeSpan, _
    cancellationToken As CancellationToken _
) As Boolean

#### C#

public bool Wait(
    TimeSpan timeout,
    CancellationToken cancellationToken
)

### Parameters

**timeout**
- Type: System...::TimeSpan
- A TimeSpan that represents the number of milliseconds to wait, or a TimeSpan that represents -1 milliseconds to wait indefinitely.

**cancellationToken**
- Type: System.Threading...::CancellationToken
- The CancellationToken to observe.

### Return Value

true if the ManualResetEventSlim was set; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out or timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>OperationCanceledException</td>
<td>cancellationToken was canceled.</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>The maximum number of waiters has been exceeded.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `ManualResetEventSlim` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsSet</td>
<td>Gets whether the event is set.</td>
</tr>
<tr>
<td>SpinCount</td>
<td>Gets the number of spin waits that will be occur before falling back to a true wait.</td>
</tr>
<tr>
<td>WaitHandle</td>
<td>Gets the underlying WaitHandle object for this ManualResetEventSlim.</td>
</tr>
</tbody>
</table>
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether the event is set.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Property IsSet As Boolean

**C#**

public bool IsSet { get; private set; }

**Field Value**

true if the event has is set; otherwise, false.
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic C#
.NET Framework Class Library
ManualResetEventSlim....SpinCount Property
ManualResetEventSlim Class See Also Send Feedback

Gets the number of spin waits that will be occur before falling back to a true wait.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Property SpinCount As Integer

C#

public int SpinCount { get; private set; }
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets the underlying WaitHandle object for this `ManualResetEventSlim`.

**Namespace:**  [System.Threading](#)

**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public ReadOnly Property WaitHandle As WaitHandle

**C#**

public WaitHandle WaitHandle { get; }

**Field Value**

The underlying WaitHandle event object for this [ManualResetEventSlim](https://docs.microsoft.com/en-us/dotnet/api/system.threading.manualreseteventslim).
Remarks

Accessing this property forces initialization of an underlying event object if one hasn't already been created. To simply wait on this ManualResetEventSlim, the public Wait methods should be preferred.
See Also

ManualResetEventSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Limits the number of threads that can access a resource or pool of resources concurrently.

**Namespace:**  System.Threading

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading := True)> _
Public Class SemaphoreSlim _
   Implements IDisposable

C#

[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading = true)]
public class SemaphoreSlim : IDisposable
Remarks

The SemaphoreSlim provides a lightweight semaphore class that doesn't use Windows kernel semaphores.

All public and protected members of SemaphoreSlim are thread-safe and may be used concurrently from multiple threads, with the exception of Dispose, which must only be used when all other operations on the SemaphoreSlim have completed.
Inheritance Hierarchy

System..::.Object
System.Threading..::.SemaphoreSlim
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
SemaphoreSlim Constructor
SemaphoreSlim Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemaphoreSlim(Int32)</td>
<td>Initializes a new instance of the <a href="https://learn.microsoft.com/en-us/dotnet/framework/threads-and-processes/semaphores">SemaphoreSlim</a> class, specifying the initial number of requests that can be granted concurrently.</td>
</tr>
<tr>
<td>SemaphoreSlim(Int32, Int32)</td>
<td>Initializes a new instance of the <a href="https://learn.microsoft.com/en-us/dotnet/framework/threads-and-processes/semaphores">SemaphoreSlim</a> class, specifying the initial and maximum number of requests that can be granted concurrently.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `SemaphoreSlim` class, specifying the initial number of requests that can be granted concurrently.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

**Visual Basic (Declaration)**

Public Sub New ( _
    initialCount As Integer _
)

**C#**

public SemaphoreSlim(  
    int initialCount
)

**Parameters**

initialCount
  Type: System...:::Int32
  The initial number of requests for the semaphore that can be granted concurrently.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>initialCount is less than 0.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
SemaphoreSlim Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the SemaphoreSlim class, specifying the initial and maximum number of requests that can be granted concurrently.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    initialCount As Integer, _
    maxCount As Integer _
)

C#

public SemaphoreSlim(
    int initialCount, 
    int maxCount
)

Parameters

initialCount
  Type: System:::Int32
  The initial number of requests for the semaphore that can be granted concurrently.

maxCount
  Type: System:::Int32
  The maximum number of requests for the semaphore that can be granted concurrently.
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..ArgumentOutOfRangeException</td>
<td>initialCount is less than 0. -or-</td>
</tr>
<tr>
<td></td>
<td>initialCount is greater than maxCount. -or-</td>
</tr>
<tr>
<td></td>
<td>maxCount is less than 0.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
SemaphoreSlim Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `SemaphoreSlim` type exposes the following members.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Release</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Wait</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
SemaphoreSlim...:::Dispose Method
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose()</td>
<td>Releases all resources used by the current instance of SemaphoreSlim.</td>
</tr>
<tr>
<td>Dispose(Boolean)</td>
<td>When overridden in a derived class, releases the unmanaged resources used by the ManualResetEventSlim, and optionally releases the managed resources.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Releases all resources used by the current instance of `SemaphoreSlim`.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub Dispose

C#

public void Dispose()

Implements

IDisposable..::.Dispose()()
Remarks

Unlike most of the members of SemaphoreSlim, Dispose() is not thread-safe and may not be used concurrently with other members of this instance.
See Also

SemaphoreSlim Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
When overridden in a derived class, releases the unmanaged resources used by the `ManualResetEventSlim`, and optionally releases the managed resources.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Protected Overridable Sub Dispose ( _
    disposing As Boolean _
)

C#

protected virtual void Dispose(
    bool disposing
)

Parameters

disposing
    Type: System..::.Boolean
    true to release both managed and unmanaged resources; false to release
    only unmanaged resources.
Remarks

Unlike most of the members of SemaphoreSlim, Dispose(Boolean) is not thread-safe and may not be used concurrently with other members of this instance.
See Also

SemaphoreSlim Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#

Include Protected Members
Include Inherited Members

.NET Framework Class Library
SemaphoreSlim...:::Release Method

SemaphoreSlim Class  See Also  Send Feedback
# Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Release()</code></td>
<td>Exits the <code>SemaphoreSlim</code> once.</td>
</tr>
<tr>
<td><code>Release(Int32)</code></td>
<td>Exits the <code>SemaphoreSlim</code> a specified number of times.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Exits the SemaphoreSlim once.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

`Public Function Release As Integer`

**C#**

`public int Release()`

### Return Value

The previous count of the [SemaphoreSlim](https://docs.microsoft.com/en-us/dotnet/api/system.threading.semaphoreslim).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
Release Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Exits the `SemaphoreSlim` a specified number of times.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Function Release ( releaseCount As Integer ) As Integer

C#

public int Release(int releaseCount)

Parameters

releaseCount
Type: System::Int32
The number of times to exit the semaphore.

Return Value

The previous count of the SemaphoreSlim.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>releaseCount is less than 1.</td>
</tr>
<tr>
<td>System.Threading:::SemaphoreFullException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.semaphoreslim">SemaphoreSlim</a> has already reached its maximum size.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
Release Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
SemaphoreSlim...:::Wait Method
SemaphoreSlim Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait()()</td>
<td>Blocks the current thread until it can enter the <strong>SemaphoreSlim</strong>.</td>
</tr>
<tr>
<td>Wait(Int32)</td>
<td>Blocks the current thread until it can enter the <strong>SemaphoreSlim</strong>, using a 32-bit signed integer to measure the time interval.</td>
</tr>
<tr>
<td>Wait(CancellationToken)</td>
<td>Blocks the current thread until it can enter the <strong>SemaphoreSlim</strong>, while observing a <strong>CancellationToken</strong>.</td>
</tr>
<tr>
<td>Wait(TimeSpan)</td>
<td>Blocks the current thread until it can enter the <strong>SemaphoreSlim</strong>, using a <strong>TimeSpan</strong> to measure the time interval.</td>
</tr>
<tr>
<td>Wait(Int32, CancellationToken)</td>
<td>Blocks the current thread until it can enter the <strong>SemaphoreSlim</strong>, using a 32-bit signed integer to measure the time interval, while observing a <strong>CancellationToken</strong>.</td>
</tr>
<tr>
<td>Wait(TimeSpan, CancellationToken)</td>
<td>Blocks the current thread until it can enter the <strong>SemaphoreSlim</strong>, using a <strong>TimeSpan</strong> to measure the time interval, while observing a <strong>CancellationToken</strong>.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until it can enter the SemaphoreSlim.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Wait

C#

public void Wait()
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.:ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until it can enter the SemaphoreSlim, using a 32-bit signed integer to measure the time interval.

**Namespace:** System.Threading
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    millisecondsTimeout As Integer _
) As Boolean

C#

public bool Wait(
    int millisecondsTimeout
)

Parameters

millisecondsTimeout
    Type: System..:::.Int32
    The number of milliseconds to wait, or Infinite(-1) to wait indefinitely.

Return Value

true if the current thread successfully entered the SemaphoreSlim; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until it can enter the SemaphoreSlim, while observing a CancellationToken.

Namespace: System.Threading
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Wait ( _
cancellationToken As CancellationToken _
)

C#

public void Wait(
    CancellationToken cancellationToken
)

Parameters

cancellationToken
    Type: System.Threading::*:*::CancellationToken
    The CancellationToken token to observe.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::OperationCanceledException</td>
<td>cancellationToken was canceled.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The current instance has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until it can enter the `SemaphoreSlim`, using a `TimeSpan` to measure the time interval.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    timeout As TimeSpan _
) As Boolean

C#

public bool Wait(
    TimeSpan timeout
)

Parameters

timeout
    Type: System::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

true if the current thread successfully entered the SemaphoreSlim; otherwise, false.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::..ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out -or- timeout is greater than MaxValue.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until it can enter the SemaphoreSlim, using a 32-bit signed integer to measure the time interval, while observing a CancellationToken.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public bool Wait(  
    int millisecondsTimeout,  
    CancellationToken cancellationToken
)

Parameters

millisecondsTimeout
    Type: System..::.Int32
    The number of milliseconds to wait, or Infinite(-1) to wait indefinitely.

cancellationToken
    Type: System.Threading..::.CancellationToken
    The CancellationToken to observe.

Return Value

true if the current thread successfully entered the SemaphoreSlim; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>cancellationToken was canceled.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Blocks the current thread until it can enter the SemaphoreSlim, using a TimeSpan to measure the time interval, while observing a CancellationToken.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    timeout As TimeSpan, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public bool Wait(
    TimeSpan timeout,
    CancellationToken cancellationToken
)

Parameters

timeout
    Type: System...:.TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

cancellationToken
    Type: System.Threading...:.CancellationToken
    The CancellationToken to observe.

Return Value

true if the current thread successfully entered the SemaphoreSlim; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out - or - timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>cancellationToken was canceled.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
Wait Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `SemaphoreSlim` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailableWaitHandle</td>
<td>Returns a WaitHandle that can be used to wait on the semaphore.</td>
</tr>
<tr>
<td>CurrentCount</td>
<td>Gets the current count of the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.semaphoreslim">SemaphoreSlim</a></td>
</tr>
</tbody>
</table>


See Also

SemaphoreSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Returns a WaitHandle that can be used to wait on the semaphore.

**Namespace:**  [System.Threading](https://docs.microsoft.com/en-us/dotnet/api/system.threading)  
**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public ReadOnly Property AvailableWaitHandle As WaitHandle

**C#**

public WaitHandle AvailableWaitHandle { get; }

**Field Value**

A WaitHandle that can be used to wait on the semaphore.
Remarks

A successful wait on the AvailableWaitHandle does not imply a successful wait on the SemaphoreSlim itself, nor does it decrement the semaphore's count. AvailableWaitHandle exists to allow a thread to block waiting on multiple semaphores, but such a wait should be followed by a true wait on the target semaphore.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.semaphoreslim">SemaphoreSlim</a> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

SemaphoreSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets the current count of the `SemaphoreSlim`.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public ReadOnly Property CurrentCount As Integer

C#

public int CurrentCount { get; }

Field Value

The current count of the SemaphoreSlim.
See Also

SemaphoreSlim Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Provides a mutual exclusion lock primitive where a thread trying to acquire the lock waits in a loop repeatedly checking until the lock becomes available.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<ComVisibleAttribute(False)> _
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading := True)> _
Public Structure SpinLock

C#

[ComVisibleAttribute(false)]
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading = true)]
public struct SpinLock
**Remarks**

Spin locks can be used for leaf-level locks where the object allocation implied by using a Monitor, in size or due to garbage collection pressure, is overly expensive. Avoiding blocking is another reason that a spin lock can be useful, however if you expect any significant amount of blocking, you are probably best not using spin locks due to excessive spinning. Spinning can be beneficial when locks are fine grained and large in number (for example, a lock per node in a linked list) as well as when lock hold times are always extremely short. In general, while holding a spin lock, one should avoid blocking, calling anything that itself may block, holding more than one spin lock at once, making dynamically dispatched calls (interface and virtuals), making statically dispatched calls into any code one doesn't own, or allocating memory.

SpinLock should only be used when it's been determined that doing so will improve an application's performance. It's also important to note that SpinLock is a value type, for performance reasons. As such, one must be very careful not to accidentally copy a SpinLock instance, as the two instances (the original and the copy) would then be completely independent of one another, which would likely lead to erroneous behavior of the application. If a SpinLock instance must be passed around, it should be passed by reference rather than by value.

Do not store SpinLock instances in readonly fields.

All members of SpinLock are thread-safe and may be used from multiple threads concurrently.
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `SpinLock` structure with the option to track thread IDs to improve debugging.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading` (in `System.Threading.dll`)
**Syntax**

**Visual Basic (Declaration)**

Public Sub New ( _
    enableThreadOwnerTracking As Boolean _
)

**C#**

public SpinLock(
    bool enableThreadOwnerTracking
)

**Parameters**

enableThreadOwnerTracking  
Type: System..::.Boolean  
Whether to capture and use thread IDs for debugging purposes.
Remarks

The default constructor for SpinLock tracks thread ownership.
See Also

SpinLock Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
The SpinLock type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enter</strong></td>
<td>Initializes a new instance of the SpinLock structure with the option to track thread IDs to improve debugging.</td>
</tr>
<tr>
<td><strong>Equals</strong></td>
<td>(Inherited from ValueType.)</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Finalize</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetHashCode</strong></td>
<td>(Inherited from ValueType.)</td>
</tr>
<tr>
<td><strong>GetType</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>MemberwiseClone</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>ToString</strong></td>
<td>(Inherited from ValueType.)</td>
</tr>
<tr>
<td><strong>TryEnter</strong></td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the SpinLock structure with the option to track thread IDs to improve debugging. Acquires the lock in a reliable manner, such that even if an exception occurs within the method call, lockTaken can be examined reliably to determine whether the lock was acquired.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Enter (_
    ByRef lockTaken As Boolean _
)

C#

public void Enter(
    ref bool lockTaken
)

Parameters

lockTaken
    Type: System..::.Boolean %
    True if the lock is acquired; otherwise, false. lockTaken must be initialized to false prior to calling this method.
Remarks

The default constructor for SpinLock tracks thread ownership.
Remarks

SpinLock is a non-reentrant lock, meaning that if a thread holds the lock, it is not allowed to enter the lock again. If thread ownership tracking is enabled (whether it's enabled is available through IsThreadOwnerTrackingEnabled), an exception will be thrown when a thread tries to re-enter a lock it already holds. However, if thread ownership tracking is disabled, attempting to enter a lock already held will result in deadlock.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Threading...:LockRecursionException</td>
<td>Thread ownership tracking is enabled, and the current thread has already acquired this lock.</td>
</tr>
<tr>
<td>System...:ArgumentException</td>
<td>The lockTaken argument must be initialized to false prior to calling Enter.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
SpinLock..::.Exit Method
SpinLock Structure  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit(0)</td>
<td>Releases the lock.</td>
</tr>
<tr>
<td>Exit(Boolean)</td>
<td>Releases the lock.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Releases the lock.

**Namespace:**  [System.Threading]

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Exit

C#

public void Exit()
Remarks

The default overload of `Exit()` provides the same behavior as if calling `Exit(Boolean)` using true as the argument.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Threading.:::SynchronizationLockException</td>
<td>Thread ownership tracking is enabled, and the current thread is not the owner of this lock.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
Exit Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
.NET Framework Class Library
SpinLock...:::Exit Method (Boolean)

SpinLock Structure  See Also  Send Feedback

Releases the lock.

**Namespace:**  System.Threading
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Exit ( _
    useMemoryBarrier As Boolean _
)  

C#

public void Exit(
    bool useMemoryBarrier
)  

Parameters

useMemoryBarrier  
    Type: System..::.Boolean  
    A Boolean value that indicates whether a memory fence should be issued in order to immediately publish the exit operation to other threads.
Remarks

Calling Exit(Boolean) with the useMemoryBarrier argument set to true will improve the fairness of the lock at the expense of some performance. The default Enter(Boolean%) overload behaves as if specifying true for useMemoryBarrier.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Threading.SynchronizationLockException</td>
<td>Thread ownership tracking is enabled, and the current thread is not the owner of this lock.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
Exit Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □ C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
SpinLock...:::TryEnter Method

SpinLock Structure  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TryEnter(Boolean%)</td>
<td>Attempts to acquire the lock in a reliable manner, such that even if an exception occurs within the method call, lockTaken can be examined reliably to determine whether the lock was acquired.</td>
</tr>
<tr>
<td>TryEnter(Int32, Boolean%)</td>
<td>Attempts to acquire the lock in a reliable manner, such that even if an exception occurs within the method call, lockTaken can be examined reliably to determine whether the lock was acquired.</td>
</tr>
<tr>
<td>TryEnter(TimeSpan, Boolean%)</td>
<td>Attempts to acquire the lock in a reliable manner, such that even if an exception occurs within the method call, lockTaken can be examined reliably to determine whether the lock was acquired.</td>
</tr>
</tbody>
</table>
See Also

[SpinLock Structure](#)
[System.Threading Namespace](#)

Send [feedback](#) on this topic to Microsoft.
Attempts to acquire the lock in a reliable manner, such that even if an exception occurs within the method call, lockTaken can be examined reliably to determine whether the lock was acquired.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub TryEnter ( _
    ByRef lockTaken As Boolean _
)

C#

public void TryEnter(
    ref bool lockTaken
)

Parameters

lockTaken
    Type: System..::.Boolean %
    True if the lock is acquired; otherwise, false. lockTaken must be initialized
to false prior to calling this method.
Remarks

Unlike Enter(Boolean%), TryEnter will not block waiting for the lock to be available. If the lock is not available when TryEnter is called, it will return immediately without any further spinning.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Threading:::LockRecursionException</td>
<td>Thread ownership tracking is enabled, and the current thread has already acquired this lock. The lockTaken argument must be initialized to false prior to calling TryEnter.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td></td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
TryEnter Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Attempts to acquire the lock in a reliable manner, such that even if an exception occurs within the method call, lockTaken can be examined reliably to determine whether the lock was acquired.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Sub TryEnter (_
    millisecondsTimeout As Integer, _
    ByRef lockTaken As Boolean _
)  
```

#### C#

```csharp
public void TryEnter(
    int millisecondsTimeout,
    ref bool lockTaken
)
```

### Parameters

millisecondsTimeout
- **Type:** System..::.Int32
- The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

lockTaken
- **Type:** System..::.Boolean %
- True if the lock is acquired; otherwise, false. lockTaken must be initialized to false prior to calling this method.
Remarks

Unlike \texttt{Enter(Boolean\%)}, \texttt{TryEnter} will not block indefinitely waiting for the lock to be available. It will block until either the lock is available or until the \texttt{millisecondsTimeout} has expired.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Threading:::LockRecursionException</td>
<td>Thread ownership tracking is enabled, and the current thread has already acquired this lock.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The lockTaken argument must be initialized to false prior to calling TryEnter.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
TryEnter Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Attempts to acquire the lock in a reliable manner, such that even if an exception occurs within the method call, lockTaken can be examined reliably to determine whether the lock was acquired.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub TryEnter (  
    timeout As TimeSpan,  
    ByRef lockTaken As Boolean  
)  

C#

public void TryEnter(  
    TimeSpan timeout,  
    ref bool lockTaken  
)  

Parameters

timeout
    Type: System...:::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a TimeSpan that represents -1 milliseconds to wait indefinitely.

lockTaken
    Type: System...:::Boolean %
    True if the lock is acquired; otherwise, false. lockTaken must be initialized to false prior to calling this method.
Remarks

Unlike `Enter(Boolean%)`, TryEnter will not block indefinitely waiting for the lock to be available. It will block until either the lock is available or until the timeout has expired.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Threading:::LockRecursionException</td>
<td>Thread ownership tracking is enabled, and the current thread has already acquired this lock.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The lockTaken argument must be initialized to false prior to calling TryEnter.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out - or - timeout is greater than MaxValue milliseconds.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
TryEnter Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The SpinLock type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsHeld</td>
<td>Gets whether the lock is currently held by any thread.</td>
</tr>
<tr>
<td>IsHeldByCurrentThread</td>
<td>Gets whether the lock is currently held by any thread.</td>
</tr>
<tr>
<td>IsThreadOwnerTrackingEnabled</td>
<td>Gets whether thread ownership tracking is enabled for this instance.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether the lock is currently held by any thread.

**Namespace:**  [System.Threading]  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsHeld As Boolean

C#

public bool IsHeld { get; }
See Also

SpinLock Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether the lock is currently held by any thread.
Gets whether the lock is held by the current thread.

**Namespace:**  [System.Threading](#)
**Assembly:**  [System.Threading](#) (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property IsHeldByCurrentThread As Boolean

C#

public bool IsHeldByCurrentThread { get; }
Remarks

If the lock was initialized to track owner threads, this will return whether the lock is acquired by the current thread. It is invalid to use this property when the lock was initialized to not track thread ownership.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.InvalidOperationException</td>
<td>Thread ownership tracking is disabled.</td>
</tr>
</tbody>
</table>
See Also

SpinLock Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether thread ownership tracking is enabled for this instance.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property IsThreadOwnerTrackingEnabled As Boolean

C#

public bool IsThreadOwnerTrackingEnabled { get; }
See Also

SpinLock Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Provides support for spin-based waiting.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading := True)> _
Public Structure SpinWait

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading = true)]
public struct SpinWait
Remarks

SpinWait encapsulates common spinning logic. On single-processor machines, yields are always used instead of busy waits, and on computers with Intel™ processors employing Hyper-Threading™ technology, it helps to prevent hardware thread starvation. SpinWait encapsulates a good mixture of spinning and true yielding.

SpinWait is a value type, which means that low-level code can utilize SpinWait without fear of unnecessary allocation overheads. SpinWait is not generally useful for ordinary applications. In most cases, you should use the synchronization classes provided by the .NET Framework, such as Monitor. For most purposes where spin waiting is required, however, the SpinWait type should be preferred over the SpinWait(Int32) method.

While SpinWait is designed to be used in concurrent applications, it is not designed to be used from multiple threads concurrently. SpinWait's members are not thread-safe. If multiple threads must spin, each should use its own instance of SpinWait.
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
The SpinWait type exposes the following members.
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from ValueType.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from ValueType.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Reset</td>
<td>Resets the spin counter.</td>
</tr>
<tr>
<td>SpinOnce</td>
<td>Performs a single spin.</td>
</tr>
<tr>
<td>SpinUntil</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from ValueType.)</td>
</tr>
</tbody>
</table>
See Also

SpinWait Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Resets the spin counter.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Reset

C#

public void Reset()
Remarks

This makes `SpinOnce()` and `NextSpinWillYield` behave as though no calls to `SpinOnce()` had been issued on this instance. If a `SpinWait` instance is reused many times, it may be useful to reset it to avoid yielding too soon.
See Also

SpinWait Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Performs a single spin.

**Namespace:**  [System.Threading](https://learn.microsoft.com/dotnet/api/system.Threading)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub SpinOnce

C#

public void SpinOnce()
Remarks

This is typically called in a loop, and may change in behavior based on the number of times a SpinOnce()() has been called thus far on this instance.
See Also

SpinWait Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
SpinWait...:::SpinUntil Method
SpinWait Structure  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SpinUntil(Func&lt;(Of &lt;(Boolean)&gt;)&gt;)</code></td>
<td>Spins until the specified condition is satisfied.</td>
</tr>
<tr>
<td><code>SpinUntil(Func&lt;(Of &lt;(Boolean)&gt;)&gt;, Int32)</code></td>
<td>Spins until the specified condition is satisfied or until the specified timeout is expired.</td>
</tr>
<tr>
<td><code>SpinUntil(Func&lt;(Of &lt;(Boolean)&gt;)&gt;, TimeSpan)</code></td>
<td>Spins until the specified condition is satisfied or until the specified timeout is expired.</td>
</tr>
</tbody>
</table>
See Also

SpinWait Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Spins until the specified condition is satisfied.

**Namespace:**  System.Threading

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Sub SpinUntil ( _
    condition As Func(Of Boolean) _
)

C#

public static void SpinUntil(
    Func<bool> condition
)

Parameters

condition
    Type: System..::.Func(Of <(Boolean)>)
    A delegate to be executed over and over until it returns true.
<table>
<thead>
<tr>
<th><strong>Exception</strong></th>
<th><strong>Condition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The condition argument is null.</td>
</tr>
</tbody>
</table>
See Also

SpinWait Structure
SpinUntil Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Spins until the specified condition is satisfied or until the specified timeout is expired.

**Namespace:**  System.Threading  
**Assembly:**  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function SpinUntil ( _
    condition As Func(Of Boolean), _
    millisecondsTimeout As Integer _) As Boolean
```

### C#

```csharp
public static bool SpinUntil(
    Func<bool> condition,
    int millisecondsTimeout
)
```

## Parameters

- **condition**
  - Type: System..::.Func(Of (Of Boolean))
  - A delegate to be executed over and over until it returns true.

- **millisecondsTimeout**
  - Type: System..::.Int32
  - The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

## Return Value

True if the condition is satisfied within the timeout; otherwise, false
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The condition argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
</tbody>
</table>
See Also

SpinWait Structure
SpinUntil Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Spins until the specified condition is satisfied or until the specified timeout is expired.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function SpinUntil (_
    condition As Func(Of Boolean), _
    timeout As TimeSpan _
) As Boolean

C#

public static bool SpinUntil(
    Func<bool> condition,
    TimeSpan timeout
)

Parameters

condition
    Type: System::Func(Of (Boolean))>
    A delegate to be executed over and over until it returns true.

timeout
    Type: System::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

True if the condition is satisfied within the timeout; otherwise, false
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentException</td>
<td>The condition argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out - or - timeout is greater than MaxValue.</td>
</tr>
</tbody>
</table>
See Also

SpinWait Structure
SpinUntil Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `SpinWait` type exposes the following members.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Gets the number of times <code>SpinOnce()</code> has been called on this instance.</td>
</tr>
<tr>
<td></td>
<td>Gets whether the next call to <code>SpinOnce()</code> will yield the processor, triggering a forced context switch.</td>
</tr>
</tbody>
</table>
See Also

SpinWait Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets the number of times SpinOnce() has been called on this instance.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Public ReadOnly Property Count As Integer

**C#**

```csharp
public int Count { get; }
```
See Also

SpinWait Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether the next call to `SpinOnce()` will yield the processor, triggering a forced context switch.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```
Public ReadOnly Property NextSpinWillYield As Boolean
```

**C#**

```
public bool NextSpinWillYield { get; }
```

Field Value

Whether the next call to [SpinOnce()](#) will yield the processor, triggering a forced context switch.
Remarks

On a single-CPU machine, SpinOnce() always yields the processor. On machines with multiple CPUs, SpinOnce() may yield after an unspecified number of calls.
See Also

SpinWait Structure
System.Threading Namespace

Send feedback on this topic to Microsoft.
Provides thread-local storage of data.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading := True)> _
Public Class ThreadLocal(Of T) _
    Implements IDisposable

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading = true)]
public class ThreadLocal<T> : IDisposable
**Type Parameters**

T

Specifies the type of data stored per-thread.
Remarks

With the exception of `Dispose()`, all public and protected members of `ThreadLocal(Of `(Of `(T`)>)` are thread-safe and may be used concurrently from multiple threads.
Inheritance Hierarchy

System..::.Object
System.Threading..::.ThreadLocal(Of (T) )}
See Also

System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic □ C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
ThreadLocal<(Of <(T)>)> Constructor
ThreadLocal<(Of <(T)>)> Class
See Also
Send Feedback
# Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ThreadLocal&lt;T&gt;</code></td>
<td>Initializes the <code>ThreadLocal&lt;T&gt;</code> instance.</td>
</tr>
<tr>
<td><code>ThreadLocal&lt;T&gt;(Func&lt;T&gt;)</code></td>
<td>Initializes the <code>ThreadLocal&lt;T&gt;</code> instance with the specified valueFactory function.</td>
</tr>
</tbody>
</table>
See Also

ThreadLocal<(Of <(T)>)> Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes the `ThreadLocal<Of <(T)>>` instance.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public ThreadLocal()
See Also

ThreadLocal(Of T) Class
ThreadLocal(Of T) Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Initializes the `ThreadLocal<Of <(T)>>()` instance with the specified `valueFactory` function.

**Namespace:**  `System.Threading`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    valueFactory As Func(Of T) _
)

C#

public ThreadLocal(
    Func<T> valueFactory
)

Parameters

valueFactory
Type: System..::.Func(Of (Of <(T)>))
The Func(Of (Of <(T)>)) invoked to produce a lazily-initialized value when an attempt is made to retrieve Value without it having been previously initialized.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>valueFactory is a null reference (Nothing in Visual Basic).</td>
</tr>
</tbody>
</table>
See Also

ThreadLocal(Of T) Class
ThreadLocal(Of T) Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `ThreadLocal<Of <(T)>>` type exposes the following members.
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Dispose</code></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><code>Equals</code></td>
<td>(Inherited from <code>Object</code>.)</td>
</tr>
<tr>
<td><code>Finalize</code></td>
<td>Releases the resources used by this ThreadLocal&lt;<code>(Of &lt;(T)&gt;)</code>&gt; instance. (Overrides <code>Object</code>:::Finalize().)</td>
</tr>
<tr>
<td><code>GetHashCode</code></td>
<td>(Inherited from <code>Object</code>.)</td>
</tr>
<tr>
<td><code>GetType</code></td>
<td>(Inherited from <code>Object</code>.)</td>
</tr>
<tr>
<td><code>MemberwiseClone</code></td>
<td>(Inherited from <code>Object</code>.)</td>
</tr>
<tr>
<td><code>ToString</code></td>
<td>Creates and returns a string representation of this instance for the current thread. (Overrides <code>Object</code>:::ToString().)</td>
</tr>
</tbody>
</table>
See Also

ThreadLocal(Of T) Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
ThreadLocal<(Of <(T)>))::.Dispose Method
ThreadLocal<(Of <(T)>)) Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose()()</td>
<td>Releases the resources used by this ThreadLocal&lt;Of (Of (T)&gt;) instance.</td>
</tr>
<tr>
<td>Dispose(Boolean)</td>
<td>Releases the resources used by this ThreadLocal&lt;Of (Of (T)&gt;) instance.</td>
</tr>
</tbody>
</table>
See Also

ThreadLocal<(Of <(T)>)> Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
ThreadLocal<Of <(T)>>...::Dispose Method

ThreadLocal<Of <(T)>> Class  See Also  Send Feedback

Releases the resources used by this ThreadLocal<Of <(T)>> instance.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Dispose

C#

public void Dispose()

Implements

IDisposable::{:Dispose()}
Remarks

Unlike most of the members of ThreadLocal<Of <(T)>), this method is not thread-safe.
See Also

ThreadLocal<(Of <(T)>)> Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
ThreadLocal<Of (T)> Dispose Method (Boolean)

ThreadLocal<Of (T)> Class  See Also  Send Feedback

Releases the resources used by this ThreadLocal<Of (T)> instance.

Namespace:  System.Threading
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Protected Overridable Sub Dispose ( _
    disposing As Boolean _
)
```

#### C#

```csharp
protected virtual void Dispose(
    bool disposing
)
```

### Parameters

disposing

Type: System..::.Boolean

A Boolean value that indicates whether this method is being called due to a call to `Dispose()`. 

---

*Note: This description is based on a passage that discusses the `Dispose` method in Visual Basic and C#.*
Remarks

Unlike most of the members of ThreadLocal<Of <(T)>>, this method is not thread-safe.
See Also

ThreadLocal(Of T) Class
Dispose Overload
System.Threading Namespace

Send feedback on this topic to Microsoft.
Releases the resources used by this ThreadLocal<Of <T>> instance.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Protected Overrides Sub Finalize

C#
protected override void Finalize()
See Also

ThreadLocal(Of T) Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
ThreadLocal<Of <(T)>>.::.ToString Method

ThreadLocal<Of <(T)>>.::.ToString Method

Creates and returns a string representation of this instance for the current thread.

**Namespace:** System.Threading

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Overrides Function ToString As String

C#

public override string ToString()

Return Value

The result of calling ToString() on the Value.
Remarks

Calling this method forces initialization for the current thread, as is the case with accessing Value directly.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::NullReferenceException</td>
<td>The <strong>Value</strong> for the current thread is a null reference (Nothing in Visual Basic).</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The initialization function referenced <strong>Value</strong> in an improper manner.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The <strong>ThreadLocal&lt;Of &lt;(T)&gt;&gt;</strong> instance has been disposed.</td>
</tr>
</tbody>
</table>
See Also

ThreadLocal<(Of <(T)>)> Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
The `ThreadLocal<Of (T)>()` type exposes the following members.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IsValueCreated</strong></td>
<td>Gets whether <strong>Value</strong> is initialized on the current thread.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>Gets or sets the value of this instance for the current thread.</td>
</tr>
</tbody>
</table>
See Also

ThreadLocal<T> Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets whether `Value` is initialized on the current thread.

**Namespace:** System.Threading  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsValueCreated As Boolean

C#

public bool IsValueCreated { get; }
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The <a href="https://example.com">ThreadLocal&lt;/Of &lt;(T)&gt;</a> instance has been disposed.</td>
</tr>
</tbody>
</table>
See Also

ThreadLocal(Of T) Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Gets or sets the value of this instance for the current thread.

**Namespace:** [System.Threading](https://docs.microsoft.com/en-us/dotnet/api/system.threading)  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Property Value As T

C#

public T Value { get; set; }
Remarks

If this instance was not previously initialized for the current thread, accessing Value will attempt to initialize it. If an initialization function was supplied during the construction, that initialization will happen by invoking the function to retrieve the initial value for Value. Otherwise, the default value of T will be used.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The initialization function referenced Value in an improper manner.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The <code>ThreadLocal&lt;Of&lt;T&gt;&gt;</code> instance has been disposed.</td>
</tr>
</tbody>
</table>
See Also

ThreadLocal(Of T) Class
System.Threading Namespace

Send feedback on this topic to Microsoft.
Visual Basic □ C#
.NET Framework Class Library
System.Threading.Tasks Namespace
Send Feedback
# Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallel</strong></td>
<td>Provides support for parallel loops and regions. Enables iterations of <strong>Parallel</strong> loops to interact with other iterations.</td>
</tr>
<tr>
<td><strong>ParallelLoopState</strong></td>
<td>Stores options that configure the operation of methods on the <strong>Parallel</strong> class.</td>
</tr>
<tr>
<td><strong>ParallelOptions</strong></td>
<td>Represents an asynchronous operation.</td>
</tr>
<tr>
<td><strong>Task</strong></td>
<td>Represents an asynchronous operation that produces a result at some time in the future.</td>
</tr>
<tr>
<td><strong>Task&lt;TResult&gt;</strong></td>
<td>Represents an exception used to communicate task cancellation.</td>
</tr>
<tr>
<td><strong>TaskCanceledException</strong></td>
<td>Represents the producer side of a Task&lt;TResult&gt; unbound to a delegate, providing access to the consumer side through the <strong>Task</strong> property.</td>
</tr>
<tr>
<td><strong>TaskCompletionSource&lt;TResult&gt;</strong></td>
<td>Provides a set of static (Shared in Visual Basic) methods for working with specific kinds of <strong>Task</strong> instances.</td>
</tr>
<tr>
<td><strong>TaskExtensions</strong></td>
<td>Provides support for creating and scheduling <strong>Tasks</strong>.</td>
</tr>
<tr>
<td><strong>TaskFactory</strong></td>
<td>Provides support for creating and scheduling Task{TResult} objects.</td>
</tr>
<tr>
<td><strong>TaskFactory&lt;TResult&gt;</strong></td>
<td>Represents an abstract scheduler for tasks.</td>
</tr>
<tr>
<td>Exception</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TaskSchedulerException</strong></td>
<td>Represents an exception used to communicate an invalid operation by a TaskScheduler.</td>
</tr>
<tr>
<td><strong>UnobservedTaskExceptionEventArgs</strong></td>
<td>Provides data for the event that is raised when a faulted Task's exception goes unobserved.</td>
</tr>
</tbody>
</table>
## Structures

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParallelLoopResult</td>
<td>Provides completion status on the execution of a Parallel loop.</td>
</tr>
</tbody>
</table>
## Enumerations

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskContinuationOptions</td>
<td>Specifies flags that control optional behavior for the creation and execution of continuation tasks.</td>
</tr>
<tr>
<td>TaskCreationOptions</td>
<td>Specifies flags that control optional behavior for the creation and execution of tasks.</td>
</tr>
<tr>
<td>TaskStatus</td>
<td>Represents the current stage in the lifecycle of a Task.</td>
</tr>
</tbody>
</table>

Send [feedback](#) on this topic to Microsoft.
Parallel Class

Provides support for parallel loops and regions.

**Namespace:**  System.Threading.Tasks

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading := True)> _
Public NotInheritable Class Parallel

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading = true)]
public static class Parallel
Remarks

The Parallel class provides library-based data parallel replacements for common operations such as for loops, for each loops, and execution of a set of statements.
Inheritance Hierarchy

System...:::Object
System.Threading.Tasks...:::Parallel
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
Parallel Methods
Parallel Class  See Also  Send Feedback
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Each</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Invoke</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
Parallel...:::For Method
Parallel Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For(Int32, Int32, Action&lt;(Of &lt;(Int32)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For(Int32, Int32, Action&lt;(Of &lt;(Int32, ParallelLoopState)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For(Int64, Int64, Action&lt;(Of &lt;(Int64)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For(Int64, Int64, Action&lt;(Of &lt;(Int64, ParallelLoopState)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For(Int32, Int32, ParallelOptions, Action&lt;(Of &lt;(Int32)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For(Int32, Int32, ParallelOptions, Action&lt;(Of &lt;(Int32, ParallelLoopState)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For(Int64, Int64, ParallelOptions, Action&lt;(Of &lt;(Int64)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For(Int64, Int64, ParallelOptions, Action&lt;(Of &lt;(Int64, ParallelLoopState)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For&lt;(Of &lt;(TLocal)&gt;))(Int32, Int32, Func&lt;(Of &lt;(TLocal)&gt;), Func&lt;(Of &lt;(Int32, ParallelLoopState, TLocal, TLocal)&gt;), Action&lt;(Of &lt;(TLocal)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
<tr>
<td>For&lt;(Of &lt;(TLocal)&gt;))(Int64, Int64, Func&lt;(Of &lt;(TLocal)&gt;), Func&lt;(Of &lt;(Int64, ParallelLoopState, TLocal, TLocal)&gt;), Action&lt;(Of &lt;(TLocal)&gt;))</td>
<td>Executes a for loop in which iterations may run in parallel.</td>
</tr>
</tbody>
</table>
**Action<(Of <(TLocal)>)>**

For<(Of <(TLocal)>)(Int32, Int32, ParallelOptions, Func<(Of <(TLocal)>)), Func<(Of <(Int32, ParallelLoopState, TLocal, TLocal)>), Action<(Of <(TLocal)>))

Executes a for loop in which iterations may run in parallel.

**Supports 64-bit indices.**

For<(Of <(TLocal)>)(Int64, Int64, ParallelOptions, Func<(Of <(TLocal)>)), Func<(Of <(Int64, ParallelLoopState, TLocal, TLocal)>), Action<(Of <(TLocal)>))

Executes a for loop in which iterations may run in parallel.
See Also

Parallel Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function For ( _
    fromInclusive As Integer, _
    toExclusive As Integer, _
    body As Action(Of Integer) _
) As ParallelLoopResult

C#

public static ParallelLoopResult For(
    int fromInclusive,
    int toExclusive,
    Action<int> body
)

Parameters

fromInclusive
    Type: System:::Int32
    The start index, inclusive.

toExclusive
    Type: System:::Int32
    The end index, exclusive.

body
    Type: System:::Action<(Of <(Int32)>)
    The delegate that is invoked once per iteration.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the iteration count (an Int32) as a parameter.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function For ( _
    fromInclusive As Integer, _
    toExclusive As Integer, _
    body As Action(Of Integer, ParallelLoopState) _) As ParallelLoopResult
```

#### C#

```csharp
public static ParallelLoopResult For(
    int fromInclusive, 
    int toExclusive, 
    Action<int, ParallelLoopState> body
)
```

### Parameters

- **fromInclusive**
  - Type: System..::.Int32
  - The start index, inclusive.

- **toExclusive**
  - Type: System..::.Int32
  - The end index, exclusive.

- **body**
  - Type: System..::.Action<(Of <(Int32, ParallelLoopState)>)>)
  - The delegate that is invoked once per iteration.

### Return Value

A **ParallelLoopResult** structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the following parameters: the iteration count (an Int32), and a ParallelLoopState instance that may be used to break out of the loop prematurely.

Calling ParallelLoopState.Break() informs the For operation that iterations after the current one need not execute. However, all iterations before the current one will still need to be executed if they have not already. Therefore, calling Break is similar to using a break operation within a conventional for loop in a language like C#, but it is not a perfect substitute: for example, there is no guarantee that iterations after the current one will definitely not execute.

If executing all iterations before the current one is not necessary, ParallelLoopState.Stop() should be preferred to using Break. Calling Stop informs the For loop that it may abandon all remaining iterations, regardless of whether they're for iterations above or below the current, since all required work has already been completed. As with Break, however, there are no guarantees regarding which other iterations will not execute.

When a loop is ended prematurely, the ParallelLoopState that's returned will contain relevant information about the loop's completion.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function For ( _
    fromInclusive As Long, _
    toExclusive As Long, _
    body As Action(Of Long) _
) As ParallelLoopResult
```

### C#

```csharp
public static ParallelLoopResult For(
    long fromInclusive,
    long toExclusive,
    Action<long> body
)
```

## Parameters

**fromInclusive**
- Type: System:::Int64
- The start index, inclusive.

**toExclusive**
- Type: System:::Int64
- The end index, exclusive.

**body**
- Type: System:::Action(Of <(Int64)>)
- The delegate that is invoked once per iteration.

## Return Value

A `ParallelLoopResult` structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the iteration count (an Int64) as a parameter.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes an for loop in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function For ( _
    fromInclusive As Long, _
    toExclusive As Long, _
    body As Action(Of Long, ParallelLoopState) _
) As ParallelLoopResult
```

### C#

```csharp
public static ParallelLoopResult For(
    long fromInclusive,
    long toExclusive,
    Action<long, ParallelLoopState> body
)
```

## Parameters

**fromInclusive**
Type: System:::Int64
The start index, inclusive.

**toExclusive**
Type: System:::Int64
The end index, exclusive.

**body**
Type: System:::Action<(Of <(Int64, ParallelLoopState)>)> The delegate that is invoked once per iteration.

## Return Value

A `ParallelLoopResult` structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the following parameters: the iteration count (an Int64), and a ParallelLoopState instance that may be used to break out of the loop prematurely.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Shared Function For ( _
    fromInclusive As Integer, _
    toExclusive As Integer, _
    parallelOptions As ParallelOptions, _
    body As Action(Of Integer) _
) As ParallelLoopResult
```

**C#**

```csharp
public static ParallelLoopResult For(
    int fromInclusive,
    int toExclusive,
    ParallelOptions parallelOptions,
    Action<int> body
)
```

**Parameters**

fromInclusive
- Type: System.Int32
- The start index, inclusive.

toExclusive
- Type: System.Int32
- The end index, exclusive.

parallelOptions
- Type: System.Threading.Tasks.ParallelOptions
- A ParallelOptions instance that configures the behavior of this operation.

body
- Type: System.Action<Of (Int32)>)
- The delegate that is invoked once per iteration.
Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the iteration count (an Int32) as a parameter.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function For ( _
    fromInclusive As Integer, _
    toExclusive As Integer, _
    parallelOptions As ParallelOptions, _
    body As Action(Of Integer, ParallelLoopState) _) _
) As ParallelLoopResult

C#

public static ParallelLoopResult For(
    int fromInclusive,
    int toExclusive,
    ParallelOptions parallelOptions,
    Action<int, ParallelLoopState> body
)

Parameters

fromInclusive
    Type: System...:::Int32
    The start index, inclusive.

toExclusive
    Type: System...:::Int32
    The end index, exclusive.

parallelOptions
    Type: System.Threading.Tasks...:::ParallelOptions
    A ParallelOptions instance that configures the behavior of this operation.

body
    Type: System...:::Action<(Of <(Int32, ParallelLoopState)>))
    The delegate that is invoked once per iteration.
Return Value

A `ParallelLoopResult` structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the following parameters: the iteration count (an Int32), and a ParallelLoopState instance that may be used to break out of the loop prematurely.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function For ( _
    fromInclusive As Long, _
    toExclusive As Long, _
    parallelOptions As ParallelOptions, _
    body As Action(Of Long) _
) As ParallelLoopResult

C#

public static ParallelLoopResult For(
    long fromInclusive,
    long toExclusive,
    ParallelOptions parallelOptions,
    Action<long> body
)

Parameters

fromInclusive
    Type: System:::Int64
    The start index, inclusive.

toExclusive
    Type: System:::Int64
    The end index, exclusive.

parallelOptions
    Type: System.Threading.Tasks:::ParallelOptions
    A ParallelOptions instance that configures the behavior of this operation.

body
    Type: System:::Action(Of<(Int64)>)
    The delegate that is invoked once per iteration.
Return Value

A `ParallelLoopResult` structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the iteration count (an Int64) as a parameter.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function For ( _
    fromInclusive As Long, _
    toExclusive As Long, _
    parallelOptions As ParallelOptions, _
    body As Action(Of Long, ParallelLoopState) _
) As ParallelLoopResult
```

**C#**

```csharp
public static ParallelLoopResult For(
    long fromInclusive,
    long toExclusive,
    ParallelOptions parallelOptions,
    Action<long, ParallelLoopState> body
)
```

### Parameters

- **fromInclusive**
  - Type: System:::Int64
  - The start index, inclusive.

- **toExclusive**
  - Type: System:::Int64
  - The end index, exclusive.

- **parallelOptions**
  - Type: System.Threading.Tasks:::ParallelOptions
  - A `ParallelOptions` instance that configures the behavior of this operation.

- **body**
  - Type: System:::Action(Of <(Int64, ParallelLoopState)>)
  - The delegate that is invoked once per iteration.
Return Value

A `ParallelLoopResult` structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the following parameters: the iteration count (an Int64), and a ParallelLoopState instance that may be used to break out of the loop prematurely.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function For(Of TLocal) ( _
  fromInclusive As Integer, _
  toExclusive As Integer, _
  localInit As Func(Of TLocal), _
  body As Func(Of Integer, ParallelLoopState, TLocal, TLocal),
  localFinally As Action(Of TLocal) _
) As ParallelLoopResult

C#

class ParallelLoopResult

class ParallelLoopState

class TLocal

class Integer

public static ParallelLoopResult For<TLocal>(
  int fromInclusive, int toExclusive, Func<TLocal> localInit,
  Func<int, ParallelLoopState, TLocal, TLocal> body,
  Action<TLocal> localFinally
)

Parameters

fromInclusive
  Type: System:::Int32
  The start index, inclusive.

toExclusive
  Type: System:::Int32
  The end index, exclusive.

localInit
  Type: System:::Func<(Of <(TLocal)>)
  The function delegate that returns the initial state of the local data for each thread.

body
  Type: System:::Func<(Of (Int32, ParallelLoopState, TLocal, TLocal)>)}
The delegate that is invoked once per iteration.

localFinally
Type: System::Action(Of <TLocal>)>
The delegate that performs a final action on the local state of each thread.
**Type Parameters**

TLocal
   The type of the thread-local data.

**Return Value**

A [ParallelLoopResult](#) structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the following parameters: the iteration count (an Int32), a ParallelLoopState instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>The exception that is thrown when the <code>localInit</code> argument is null.</td>
</tr>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>The exception that is thrown when the <code>localFinally</code> argument is null.</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#  
.NET Framework Class Library
Parallel...::For<(Of <(TLocal)>>) Method (Int64, Int64, Func<(Of <(TLocal)>)), Func<(Of <(Int64, ParallelLoopState, TLocal, TLocal)>)), Action<(Of <(TLocal)>))
Parallel Class  See Also  Send Feedback

Executes a for loop in which iterations may run in parallel. Supports 64-bit indices.

Namespace: System Threading Tasks
Assembly: System Threading (in System Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function For(Of TLocal) ( _
    fromInclusive As Long, _
    toExclusive As Long, _
    localInit As Func(Of TLocal), _
    body As Func(Of Long, ParallelLoopState, TLocal, TLocal), _
    localFinally As Action(Of TLocal) _
) As ParallelLoopResult

C#

public static ParallelLoopResult For<TLocal>(
    long fromInclusive,
    long toExclusive,
    Func<TLocal> localInit,
    Func<long, ParallelLoopState, TLocal, TLocal> body,
    Action<TLocal> localFinally
)

Parameters

fromInclusive
    Type: System....:Int64
    The start index, inclusive.

toExclusive
    Type: System....:Int64
    The end index, exclusive.

localInit
    Type: System....:Func<(Of <(TLocal)>)>
    The function delegate that returns the initial state of the local data for each thread.

body
    Type: System....:Func<(Of <(Int64, ParallelLoopState, TLocal, TLocal)>)>
The delegate that is invoked once per iteration.

localFinally
Type: System:::Action<Of <(TLocal)>>
The delegate that performs a final action on the local state of each thread.
Type Parameters

TLocal
   The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the following parameters: the iteration count (an Int64), a ParallelLoopState instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function For(Of TLocal) ( _
  fromInclusive As Integer, _
  toExclusive As Integer, _
  parallelOptions As ParallelOptions, _
  localInit As Func(Of TLocal), _
  body As Func(Of Integer, ParallelLoopState, TLocal, TLocal), _
  localFinally As Action(Of TLocal) _) As ParallelLoopResult

C#

public static ParallelLoopResult For<TLocal>(
  int fromInclusive,
  int toExclusive,
  ParallelOptions parallelOptions,
  Func<TLocal> localInit,
  Func<int, ParallelLoopState, TLocal, TLocal> body,
  Action<TLocal> localFinally
)

Parameters

fromInclusive
  Type: System.Int32
  The start index, inclusive.

toExclusive
  Type: System.Int32
  The end index, exclusive.

parallelOptions
  Type: System.Threading.Tasks.ParallelOptions
  A ParallelOptions instance that configures the behavior of this operation.

localInit
Type: System::Func<(Of <(TLocal)>)>  
The function delegate that returns the initial state of the local data for each thread.

body  
Type: System::Func<(Of <(Int32, ParallelLoopState, TLocal, TLocal)>)>  
The delegate that is invoked once per iteration.

localFinally  
Type: System::Action<(Of <(TLocal)>)>  
The delegate that performs a final action on the local state of each thread.
Type Parameters

TLocal

The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the following parameters: the iteration count (an Int32), a ParallelLoopState instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for loop in which iterations may run in parallel.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function For(Of TLocal) ( _
    fromInclusive As Long, _
    toExclusive As Long, _
    parallelOptions As ParallelOptions, _
    localInit As Func(Of TLocal), _
    body As Func(Of Long, ParallelLoopState, TLocal, TLocal), _
    localFinally As Action(Of TLocal) _
) As ParallelLoopResult

C#

public static ParallelLoopResult For<TLocal>(
    long fromInclusive,
    long toExclusive,
    ParallelOptions parallelOptions,
    Func<TLocal> localInit,
    Func<long, ParallelLoopState, TLocal, TLocal> body,
    Action<TLocal> localFinally
)

Parameters

fromInclusive
    Type: System...:::Int64
    The start index, inclusive.

toExclusive
    Type: System...:::Int64
    The end index, exclusive.

parallelOptions
    Type: System.Threading.Tasks...:::ParallelOptions
    A ParallelOptions instance that configures the behavior of this operation.

localInit
Type: System::Func<(Of <(TLocal)>)>  
The function delegate that returns the initial state of the local data for each thread.

body  
Type: System::Func<(Of <(Int64, ParallelLoopState, TLocal, TLocal)>)>  
The delegate that is invoked once per iteration.

localFinally  
Type: System::Action<(Of <(TLocal)>)>  
The delegate that performs a final action on the local state of each thread.
Type Parameters

TLocal
The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each value in the iteration range: [fromInclusive, toExclusive). It is provided with the following parameters: the iteration count (an Int64), a ParallelLoopState instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
For Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
Parallel...:..ForEach Method
Parallel Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ForEach&lt;(Of &lt;(TSource)&gt;), (OrderablePartitioner&lt;(Of &lt;(TSource)&gt;), Action&lt;(Of &lt;(TSource, ParallelLoopState, Int64)&gt;)&gt;))</code></td>
<td>Executes a for each operation on an OrderablePartitioner in which iterations may run in parallel.</td>
</tr>
<tr>
<td><code>ForEach&lt;(Of &lt;(TSource)&gt;), (Partitioner&lt;(Of &lt;(TSource)&gt;), Action&lt;(Of &lt;(TSource)&gt;)&gt;))</code></td>
<td>Executes a for each operation on a Partitioner in which iterations may run in parallel.</td>
</tr>
<tr>
<td><code>ForEach&lt;(Of &lt;(TSource)&gt;), (Partitioner&lt;(Of &lt;(TSource)&gt;), Action&lt;(Of &lt;(TSource, ParallelLoopState)&gt;)&gt;))</code></td>
<td>Executes a for each operation on a Partitioner in which iterations may run in parallel.</td>
</tr>
<tr>
<td><code>ForEach&lt;(Of &lt;(TSource)&gt;), (IEnumerable&lt;(Of &lt;(TSource)&gt;), Action&lt;(Of &lt;(TSource)&gt;)&gt;))</code></td>
<td>Executes a for each operation on an IEnumerable&lt;(Of &lt;(TSource)&gt;)&gt; in which iterations may run in parallel.</td>
</tr>
<tr>
<td><code>ForEach&lt;(Of &lt;(TSource)&gt;), (IEnumerable&lt;(Of &lt;(TSource)&gt;), Action&lt;(Of &lt;(TSource, ParallelLoopState)&gt;)&gt;))</code></td>
<td>Executes a for each operation on an IEnumerable&lt;(Of &lt;(TSource)&gt;)&gt; in which iterations may run in parallel.</td>
</tr>
<tr>
<td><code>ForEach&lt;(Of &lt;(TSource)&gt;), (IEnumerable&lt;(Of &lt;(TSource)&gt;), Action&lt;(Of &lt;(TSource, ParallelLoopState, Int64)&gt;)&gt;))</code></td>
<td>Executes a for each operation on an IEnumerable&lt;(Of &lt;(TSource)&gt;)&gt; in which iterations may run in parallel.</td>
</tr>
</tbody>
</table>
run in parallel.
Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.
ParallelLoopState, Int64, TLocal, TLocal>, Action(Of TLocal))
ForEach(Of (TSource, TLocal)>)
(Partitioner(Of (TSource)>, Func(Of TLocal, Func(Of TSource, ParallelLoopState, TLocal, TLocal), Action(Of TLocal)))

ForEach(Of (TSource, TLocal)>)
(IEnumerable(Of (TSource)>, Func(Of TLocal, Func(Of TSource, ParallelLoopState, TLocal, TLocal), Action(Of TLocal)))

ForEach(Of (TSource, TLocal)>)
(IEnumerable(Of (TSource)>, Func(Of TLocal, Func(Of TSource, ParallelLoopState, TLocal, TLocal), Action(Of TLocal)))

ForEach(Of (TSource, TLocal)>)
(OrderablePartitioner(Of (TSource)>, ParallelOptions, Func(Of TLocal, Func(Of TSource, ParallelLoopState, Int64, TLocal, TLocal), Action(Of TLocal)))

ForEach(Of (TSource, TLocal)>)
(Partitioner(Of (TSource)>, ParallelOptions, Func(Of TLocal, Func(Of TSource, ParallelLoopState, TLocal, TLocal), Action(Of TLocal)))

ForEach(Of (TSource, TLocal)>)
(IEnumerable(Of (TSource)>, ParallelOptions, Func(Of TLocal, Func(Of TSource, ParallelLoopState, TLocal, TLocal), Action(Of TLocal)))

which iterations may run in parallel.
Executes a for each operation on a Partitioner in which iterations may run in parallel.
Executes a for each operation on an IEnumerable(Of (TSource)>) in which iterations may run in parallel.
Executes a for each operation on an OrderablePartitioner in which iterations may run in parallel.
Executes a for each operation on a Partitioner in which iterations may run in parallel.
Executes a for each operation on an IEnumerable(Of (TSource)>) in which iterations may run in parallel.
Executes a for each operation on an IEnumerable(Of TSource) in which iterations may run in parallel.
See Also

Parallel Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

**Namespace:** [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource) ( _
    source As OrderablePartitioner(Of TSource), _
    body As Action(Of TSource, ParallelLoopState, Long) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource>(
    OrderablePartitioner<TSource> source,
    Action<TSource, ParallelLoopState, long> body
)

Parameters

source
Type: System.Collections.Concurrent.OrderablePartitioner<
    (Of <(TSource>>)>
The OrderablePartitioner that contains the original data source.

body
Type: System.Action<
    (Of <(TSource, ParallelLoopState, Int64)>)>
The delegate that is invoked once per iteration.
Type Parameters

TSource
The type of the elements in source.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the following parameters: the current element, a `ParallelLoopState` instance that may be used to break out of the loop prematurely, and the current element's index (an Int64).
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source OrderablePartitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the KeysNormalized property in the source OrderablePartitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source OrderablePartitioner return null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() or GetOrderablePartitions() methods in the source OrderablePartitioner do not return the correct number of partitions.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetDynamicPartitions() or GetDynamicOrderablePartitions() methods in the source OrderablePartitioner return an IEnumerable whose GetEnumerator() method returns null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown to contain</td>
</tr>
</tbody>
</table>
System..::..AggregateException an exception thrown from one of the specified delegates.
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for each operation on a Partitioner in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Function ForEach(Of TSource) ( _
    source As Partitioner(Of TSource), _
    body As Action(Of TSource) _
) As ParallelLoopResult
```

**C#**

```csharp
public static ParallelLoopResult ForEach<TSource>(
    Partitioner<TSource> source,
    Action<TSource> body
)
```

**Parameters**

**source**
- Type: `System.Collections.Concurrent:::Partitioner<Of <(TSource)>)`
  - The Partitioner that contains the original data source.

**body**
- Type: `System:::Action<Of <(TSource)>)`
  - The delegate that is invoked once per iteration.
Type Parameters

TSource
   The type of the elements in source.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the current element as a parameter.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System...:::InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source Partitioner returns false.</td>
</tr>
<tr>
<td>System...:::InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source Partitioner return null.</td>
</tr>
<tr>
<td>System...:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner does not return the correct number of partitions.</td>
</tr>
<tr>
<td>System...:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner returns an IList with at least one null value.</td>
</tr>
<tr>
<td>System...:::InvalidOperationException</td>
<td>The exception that is thrown when the GetDynamicPartitions() method in the source Partitioner returns an IEnumerable whose GetEnumerator() method returns null.</td>
</tr>
<tr>
<td>System...:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for each operation on a Partitioner in which iterations may run in parallel.

**Namespace:**  System.Threading.Tasks

**Assembly:**  System.Threading (in System.Threading.dll)
Syntaxe

**Visual Basic (Declaration)**

```vbnet
Public Shared Function ForEach(Of TSource) ( _
    source As Partitioner(Of TSource), _
    body As Action(Of TSource, ParallelLoopState) _
) As ParallelLoopResult
```

**C#**

```csharp
public static ParallelLoopResult ForEach<TSource>(
    Partitioner<TSource> source,
    Action<TSource, ParallelLoopState> body
)
```

**Parameters**

**source**

Type: System.Collections.Concurrent..:::Partitioner<(Of @(TSource)>)

The Partitioner that contains the original data source.

**body**

Type: System..::.Action<(Of @(TSource, ParallelLoopState)>)

The delegate that is invoked once per iteration.
Type Parameters

TSource

The type of the elements in source.

Return Value

A **ParallelLoopResult** structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the following parameters: the current element, and a ParallelLoopState instance that may be used to break out of the loop prematurely.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source Partitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source Partitioner return null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner does not return the correct number of partitions.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner returns an IList with at least one null value.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetDynamicPartitions() method in the source Partitioner returns an IEnumerable whose GetEnumerator() method returns null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for each operation on an `IEnumerable<Of <(TSource)>>)` in which iterations may run in parallel.

**Namespace:** [System.Threading.Tasks](#)

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function ForEach(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    body As Action(Of TSource) _
) As ParallelLoopResult
```

#### C#

```csharp
public static ParallelLoopResult ForEach<TSource>(
    IEnumerable<TSource> source,
    Action<TSource> body
)
```

### Parameters

**source**

Type: System.Collections.Generic:::IEnumerable<(Of <(TSource)>)>  
An enumerable data source.

**body**

Type: System:::Action<(Of <(TSource)>)>  
The delegate that is invoked once per iteration.
*Type Parameters*

**TSource**

The type of the data in the source.

**Return Value**

A `ParallelLoopResult` structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the current element as a parameter.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel..::.ForEach(Of (Of (TSource)>) Method (IEaggable(Of (Of (TSource)>)>, Action(Of (Of (TSource, ParallelLoopState>)>))

Executes a for each operation on an IEnDuerable(Of (Of (TSource)>) in which iterations may run in parallel.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource) ( _
  source As(IEnumerable(Of TSource), _
  body As Action(Of TSource, ParallelLoopState) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource>(
  IEnumerable<TSource> source,
  Action<TSource, ParallelLoopState> body
)

Parameters

source
  Type: System.Collections.Generic..::.IEnumerable<(Of <(TSource)>>)
  An enumerable data source.

body
  Type: System..::.Action<(Of <(TSource, ParallelLoopState)>))
  The delegate that is invoked once per iteration.
Type Parameters

TSource
The type of the data in the source.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the following parameters: the current element, and a `ParallelLoopState` instance that may be used to break out of the loop prematurely.
**Exceptions**

<table>
<thead>
<tr>
<th><strong>Exception</strong></th>
<th><strong>Condition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel..::.ForEach(Of (Of (TSource)>)) Method (IEnumerable<(Of (TSource)>), Action<(Of (TSource, ParallelLoopState, Int64)>))

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)

Executes a for each operation on an IEnumerable<(Of (TSource)>)) in which iterations may run in parallel.
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource) ( _
  source As IEnumerable(Of TSource), _
  body As Action(Of TSource, ParallelLoopState, Long) _) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource>(
  IEnumerable<TSource> source,
  Action<TSource, ParallelLoopState, long> body
)

Parameters

source
  Type: System.Collections.Generic.IEnumerable(Of TSource)
  An enumerable data source.

body
  Type: System.Action(Of TSource, ParallelLoopState, Int64)
  The delegate that is invoked once per iteration.
### Type Parameters

**TSource**

The type of the data in the source.

### Return Value

A [ParallelLoopResult](#) structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the following parameters: the current element, a `ParallelLoopState` instance that may be used to break out of the loop prematurely, and the current element's index (an Int64).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel:::ForEach(Of (Of (TSOURCE))>, ParallelOptions, Action(Of (TSOURCE, ParallelLoopState, Int64)>)

**Parallel Class**  **See Also**  **Send Feedback**

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

**Namespace:**  System.Threading.Tasks  **Assembly:**  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function ForEach(Of TSource) (_
    source As OrderablePartitioner(Of TSource), _
    parallelOptions As ParallelOptions, _
    body As Action(Of TSource, ParallelLoopState, Long) _) _
) As ParallelLoopResult
```

#### C#

```csharp
public static ParallelLoopResult ForEach<TSource>(
    OrderablePartitioner<TSource> source,
    ParallelOptions parallelOptions,
    Action<TSource, ParallelLoopState, long> body
)
```

### Parameters

**source**
- Type: `System.Collections.Concurrent.OrderablePartitioner`<Of `(TSource)>
- The OrderablePartitioner that contains the original data source.

**parallelOptions**
- Type: `System.Threading.Tasks.ParallelOptions`
- A `ParallelOptions` instance that configures the behavior of this operation.

**body**
- Type: `System..::.Action<Of <(TSource, ParallelLoopState, Int64)>>`
- The delegate that is invoked once per iteration.
**Type Parameters**

**TSource**

The type of the elements in source.

**Return Value**

A [ParallelLoopResult](#) structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, and the current element's index (an Int64).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::OperatingCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source OrderablePartitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the KeysNormalized property in the source OrderablePartitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source OrderablePartitioner return null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() or GetOrderablePartitions() methods in the source OrderablePartitioner do not return the correct number of partitions.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() or GetDynamicPartitions() methods in the source OrderablePartitioner return an IList with at least one null value.</td>
</tr>
<tr>
<td>Exception Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for each operation on a Partitioner in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource) ( _
    source As Partitioner(Of TSource), _
    parallelOptions As ParallelOptions, _
    body As Action(Of TSource) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource>(
    Partitioner<TSource> source,
    ParallelOptions parallelOptions,
    Action<TSource> body
)

Parameters

source
    Type: System.Collections.Concurrent..::.Partitioner(Of (Of TSource)>)
    The Partitioner that contains the original data source.

parallelOptions
    Type: System.Threading.Tasks..::.ParallelOptions
    A ParallelOptions instance that configures the behavior of this operation.

body
    Type: System..::.Action(Of (Of TSource)>)
    The delegate that is invoked once per iteration.
Type Parameters

TSource
   The type of the elements in source.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the current element as a parameter.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source Partitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source Partitioner return null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner does not return the correct number of partitions.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner returns an IList with at least one null value.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetDynamicPartitions() method in the source Partitioner returns an IEnumerable whose GetEnumerator() method returns null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
System::ObjectDisposedException

The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.
See Also

Parallel Class
forEach overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel...:.ForEach<(Of <(TSource)>)> Method (Partitioner<(Of <(TSource)>)>), ParallelOptions, Action<(Of <(TSource, ParallelLoopState)>)>))

Parallel Class  See Also  Send Feedback

Executes a for each operation on a Partitioner in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource) ( _
    source As Partitioner(Of TSource), _
    parallelOptions As ParallelOptions, _
    body As Action(Of TSource, ParallelLoopState) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource>(
    Partitioner<TSource> source,
    ParallelOptions parallelOptions,
    Action<TSource, ParallelLoopState> body
)

Parameters

source
Type: System.Collections.Concurrent..::.Partitioner<Of (TSource)>)
The Partitioner that contains the original data source.

parallelOptions
Type: System.Threading.Tasks..::.ParallelOptions
A ParallelOptions instance that configures the behavior of this operation.

body
Type: System..::.Action<Of (TSource, ParallelLoopState)>)
The delegate that is invoked once per iteration.
Type Parameters

TSource
   The type of the elements in source.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the following parameters: the current element, and a ParallelLoopState instance that may be used to break out of the loop prematurely.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source Partitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source Partitioner return null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner does not return the correct number of partitions.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner returns an IList with at least one null value.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetDynamicPartitions() method in the source Partitioner returns an IEnumerable whose GetEnumerator() method returns null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
System:::ObjectDisposedException  The exception that is thrown when the
the CancellationTokenSource
associated with the the
CancellationToken in the
parallelOptions has been disposed.
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes a for each operation on an IEnumerable(Of TSource) in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Shared Function ForEach(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    parallelOptions As ParallelOptions, _
    body As Action(Of TSource) _
) As ParallelLoopResult
```

**C#**

```csharp
public static ParallelLoopResult.ForEach<TSource>(
    IEnumerable<TSource> source,
    ParallelOptions parallelOptions,
    Action<TSource> body
)
```

**Parameters**

**source**
- Type: System.Collections.Generic.IEnumerable<
  System.Collections.Generic.TSource>
- An enumerable data source.

**parallelOptions**
- Type: System.Threading.Tasks.ParallelOptions
- A ParallelOptions instance that configures the behavior of this operation.

**body**
- Type: System.Action<
  System.Collections.Generic.TSource>
- The delegate that is invoked once per iteration.
**Type Parameters**

**TSource**

The type of the data in the source.

**Return Value**

A `ParallelLoopResult` structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the current element as a parameter.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td><code>OperationCanceledException</code></td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td><code>AggregateException</code></td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td><code>ObjectDisposedException</code></td>
<td>The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel.ForEach(Of TSource) Method (IEnumerable(Of TSource), ParallelOptions, Action(Of TSource, ParallelLoopState>))

Executes a for each operation on an IEnumerable(Of TSource) in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    parallelOptions As ParallelOptions, _
    body As Action(Of TSource, ParallelLoopState) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource>(
    IEnumerable<TSource> source,
    ParallelOptions parallelOptions,
    Action<TSource, ParallelLoopState> body
)

Parameters

source
Type: System.Collections.Generic.IEnumerable(Of TSource)
An enumerable data source.

parallelOptions
Type: System.Threading.Tasks.ParallelOptions
A ParallelOptions instance that configures the behavior of this operation.

body
Type: System.Action(Of TSource, ParallelLoopState)
The delegate that is invoked once per iteration.
Type Parameters

TSource
   The type of the data in the source.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the following parameters: the current element, and a ParallelLoopState instance that may be used to break out of the loop prematurely.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel Class
See Also
Send Feedback

Executes a for each operation on an IEnumerable(Of TSource) in which iterations may run in parallel.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    parallelOptions As ParallelOptions, _
    body As Action(Of TSource, ParallelLoopState, Long) _) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource>(
    IEnumerable<TSource> source,
    ParallelOptions parallelOptions,
    Action<TSource, ParallelLoopState, long> body
)

Parameters

source
    Type: System.Collections.Generic..::.IEnumerable<(Of <(TSource)>))
    An enumerable data source.

parallelOptions
    Type: System.Threading.Tasks..::.ParallelOptions
    A ParallelOptions instance that configures the behavior of this operation.

body
    Type: System..::.Action<(Of <(TSource, ParallelLoopState, Int64)>))
    The delegate that is invoked once per iteration.
Type Parameters

TSource

The type of the data in the source.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, and the current element's index (an Int64).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel..::.ForEach<(Of <(TSource, TLocal)>)) Method
(OrderablePartitioner<(Of <(TSource)>), Func<(Of <(TLocal)>), Func<(Of
<(TSource, ParallelLoopState, Int64, TLocal, TLocal)>), Action<(Of
<(TLocal)>))

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource, TLocal) ( _
    source As OrderablePartitioner(Of TSource), _
    localInit As Func(Of TLocal), _
    body As Func(Of TSource, ParallelLoopState, Long, TLocal, TLocal), _
    localFinally As Action(Of TLocal) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource, TLocal>(
    OrderablePartitioner<TSource> source,
    Func<TLocal> localInit,
    Func<TSource, ParallelLoopState, long, TLocal, TLocal> body,
    Action<TLocal> localFinally
)

Parameters

source
Type: System.Collections.Concurrent.IOrderedPartitioner<TSource>
The OrderablePartitioner that contains the original data source.

localInit
Type: System.Func<TLocal>
The function delegate that returns the initial state of the local data for each thread.

body
Type: System.Func<TSource, ParallelLoopState, Int64, TLocal, TLocal>
The delegate that is invoked once per iteration.

localFinally
Type: System::Action<TLocal>

The delegate that performs a final action on the local state of each thread.
Type Parameters

TSource
   The type of the elements in source.
TLocal
   The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, the current element's index (an Int64), and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td><code>InvalidOperationException</code></td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source OrderablePartitioner returns false.</td>
</tr>
<tr>
<td><code>InvalidOperationException</code></td>
<td>The exception that is thrown when the KeysNormalized property in the source OrderablePartitioner returns false.</td>
</tr>
<tr>
<td><code>InvalidOperationException</code></td>
<td>The exception that is thrown when any methods in the source OrderablePartitioner return null.</td>
</tr>
<tr>
<td><code>InvalidOperationException</code></td>
<td>The exception that is thrown when the <code>GetPartitions()</code> or <code>GetOrderablePartitions()</code> methods in the source OrderablePartitioner do not return the correct number of partitions.</td>
</tr>
<tr>
<td><code>InvalidOperationException</code></td>
<td>The exception that is thrown when the <code>GetDynamicPartitions()</code> or <code>GetDynamicOrderablePartitions()</code> methods in the source OrderablePartitioner return an <code>IList</code> with at least one null value.</td>
</tr>
</tbody>
</table>
System:::InvalidOperationException methods in the source OrderablePartitioner return an IEnumerable whose GetEnumerator() method returns null.

System:::AggregateException The exception that is thrown to contain an exception thrown from one of the specified delegates.
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
Parallel:::ForEach<(Of <(TSource, TLocal)>)) Method (Partitioner<(Of <(TSource)>), Func<(Of <(TLocal)>)), Func<(Of <(TSource, ParallelLoopState, TLocal, TLocal)>)), Action<(Of <(TLocal)>))

Executes a for each operation on a Partitioner in which iterations may run in parallel.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource, TLocal) ( _
    source As Partitioner(Of TSource), _
    localInit As Func(Of TLocal), _
    body As Func(Of TSource, ParallelLoopState, TLocal, TLocal),
    localFinally As Action(Of TLocal) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource, TLocal>(
    Partitioner<TSource> source,
    Func<TLocal> localInit,
    Func<TSource, ParallelLoopState, TLocal, TLocal> body,
    Action<TLocal> localFinally
)

Parameters

source
    Type: System.Collections.Concurrent..:::.Partitioner<(Of (TSource)>)
The Partitioner that contains the original data source.

localInit
    Type: System...:::Func<(Of (TLocal)>)
The function delegate that returns the initial state of the local data for each thread.

body
    Type: System...:::Func<(Of (TSource, ParallelLoopState, TLocal, TLocal)>)
The delegate that is invoked once per iteration.

localFinally
    Type: System...:::Action<(Of (TLocal)>)
The delegate that performs a final action on the local state of each thread.
Type Parameters

TSourse
   The type of the elements in source.
TLocal
   The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::.ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System::.ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System::.ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System::.ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System::.InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source Partitioner returns false.</td>
</tr>
<tr>
<td>System::.InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source Partitioner return null.</td>
</tr>
<tr>
<td>System::.InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner does not return the correct number of partitions.</td>
</tr>
<tr>
<td>System::.InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner returns an IList with at least one null value.</td>
</tr>
<tr>
<td>System::.InvalidOperationException</td>
<td>The exception that is thrown when the GetDynamicPartitions() method in the source Partitioner returns an IEnumerable whose GetEnumerator() method returns null.</td>
</tr>
<tr>
<td>System::.AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel...:::ForEach<Of <(TSource, TLocal)>>) Method (IEnumerable<Of <(TSource)>), Func<Of <(TLocal)>>, Func<Of <(TSource, ParallelLoopState, TLocal, TLocal)>>, Action<Of <(TLocal)>>)

EXECUTES A FOR EACH OPERATION ON AN IEnumerable<Of <(TSource)>> IN WHICH ITERATIONS MAY RUN IN PARALLEL.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource, TLocal) ( _
    source As IEnumerable(Of TSource), _
    localInit As Func(Of TLocal), _
    body As Func(Of TSource, ParallelLoopState, TLocal, TLocal),
    localFinally As Action(Of TLocal) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource, TLocal>(
    IEnumerable<TSource> source,
    Func<TLocal> localInit,
    Func<TSource, ParallelLoopState, TLocal, TLocal> body,
    Action<TLocal> localFinally
)

Parameters

source
    Type: System.Collections.Generic:::IEnumerable<(Of <(TSource)>)>
    An enumerable data source.

localInit
    Type: System:::Func<(Of <(TLocal)>)>
    The function delegate that returns the initial state of the local data for each thread.

body
    Type: System:::Func<(Of <(TSource, ParallelLoopState, TLocal, TLocal)>)>
    The delegate that is invoked once per iteration.

localFinally
    Type: System:::Action<(Of <(TLocal)>)>
The delegate that performs a final action on the local state of each thread.
**Type Parameters**

TSource
   The type of the data in the source.
TLocal
   The type of the thread-local data.

**Return Value**

A [ParallelLoopResult](#) structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel..::.ForEach(Of (TSource, TLocal)>) Method (IEnumerable(Of (TSource)>, Func(Of (TLocal)>, Func(Of (TSource, ParallelLoopState, Int64, TLocal, TLocal)>, Action(Of (TLocal)>)>)

Executes a for each operation on an IEnumerable(Of (TSource)>) in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

```vbnet
Public Shared Function ForEach(Of TSource, TLocal) (   
    source As IEnumerable(Of TSource), _
    localInit As Func(Of TLocal), _
    body As Func(Of TSource, ParallelLoopState, Long, TLocal, TLocal), _
    localFinally As Action(Of TLocal) _) 
) As ParallelLoopResult
```

C#

```csharp
public static ParallelLoopResult ForEach<TSource, TLocal>(
    IEnumerable<TSource> source,
    Func<TLocal> localInit,
    Func<TSource, ParallelLoopState, long, TLocal, TLocal> body,
    Action<TLocal> localFinally
)
```

Parameters

source
  Type: System.Collections.Generic.IEnumerable<(Of <(TSource)>)
  An enumerable data source.

localInit
  Type: System.Func<(Of <(TLocal)>)
  The function delegate that returns the initial state of the local data for each thread.

body
  Type: System.Func<(Of <(TSource, ParallelLoopState, Int64, TLocal, TLocal)>)
  The delegate that is invoked once per iteration.

localFinally
  Type: System.Action<(Of <(TLocal)>)
```
The delegate that performs a final action on the local state of each thread.
**Type Parameters**

**TSource**
- The type of the data in the source.

**TLocal**
- The type of the thread-local data.

**Return Value**

A [ParallelLoopResult](#) structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, the current element's index (an Int64), and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel:::ForEach<(Of <(TSource, TLocal)>)) Method
(OrderablePartitioner<(Of <(TSource)>), ParallelOptions, Func<(Of <(TLocal)>), Func<(Of <(TSource, ParallelLoopState, Int64, TLocal, TLocal)>), Action<(Of <(TLocal)>))>

Parallel Class  See Also  Send Feedback

Executes a for each operation on a OrderablePartitioner in which iterations may run in parallel.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Shared Function ForEach(Of TSource, TLocal) ( _
    source As OrderablePartitioner(Of TSource), _
    parallelOptions As ParallelOptions, _
    localInit As Func(Of TLocal), _
    body As Func(Of TSource, ParallelLoopState, Long, TLocal, TLocal), _
    localFinally As Action(Of TLocal) _) As ParallelLoopResult
```

**C#**

```csharp
public static ParallelLoopResult ForEach<TSource, TLocal>(
    OrderablePartitioner<TSource> source,
    ParallelOptions parallelOptions,
    Func<TLocal> localInit,
    Func<TSource, ParallelLoopState, long, TLocal, TLocal> body,
    Action<TLocal> localFinally
)
```

**Parameters**

**source**

Type: `System.Collections.Concurrent.OrderablePartitioner<TSource>`

The OrderablePartitioner that contains the original data source.

**parallelOptions**

Type: `System.Threading.Tasks.ParallelOptions`

A `ParallelOptions` instance that configures the behavior of this operation.

**localInit**

Type: `System.Func<TLocal>`

The function delegate that returns the initial state of the local data for each thread.

**body**
Type: System:::Func<(Of <(TSource, ParallelLoopState, Int64, TLocal, TLocal)>))
The delegate that is invoked once per iteration.

localFinally
Type: System:::Action<(Of <(TLocal)>))
The delegate that performs a final action on the local state of each thread.
Type Parameters

TSource
The type of the elements in source.

TLocal
The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
**Remarks**

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, the current element's index (an Int64), and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source OrderablePartitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the KeysNormalized property in the source OrderablePartitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source OrderablePartitioner return null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() or GetOrderablePartitions() methods in the source OrderablePartitioner do not return the correct number of partitions.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() or GetOrderablePartitions() methods in</td>
</tr>
</tbody>
</table>
the source OrderablePartitioner return an IList with at least one null value.
The exception that is thrown when the GetDynamicPartitions() or
GetDynamicOrderablePartitions() methods in the source
OrderablePartitioner return an IEnumerable whose GetEnumerator() method returns null.
The exception that is thrown to contain an exception thrown from one of the specified delegates.
The exception that is thrown when the CancellationTokenSource associated with the the CancellationToken in the parallelOptions has been disposed.

System:::InvalidOperationException

System:::AggregateException

System:::ObjectDisposedException
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
Parallel..::.ForEach<(Of <(TSource, TLocal)>)) Method (Partitioner<(Of <(TSource)>), ParallelOptions, Func<(Of <(TLocal)>)), Func<(Of <(TSource, ParallelLoopState, TLocal, TLocal)>), Action<(Of <(TLocal)>))>

Executes a for each operation on a Partitioner in which iterations may run in parallel.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Shared Function ForEach(Of TSource, TLocal) ( _
    source As Partitioner(Of TSource), _
    parallelOptions As ParallelOptions, _
    localInit As Func(Of TLocal), _
    body As Func(Of TSource, ParallelLoopState, TLocal, TLocal), _
    localFinally As Action(Of TLocal) _
) As ParallelLoopResult
```

### C#

```csharp
public static ParallelLoopResult ForEach<TSource, TLocal>(
    Partitioner<TSource> source,
    ParallelOptions parallelOptions,
    Func<TLocal> localInit,
    Func<TSource, ParallelLoopState, TLocal, TLocal> body,
    Action<TLocal> localFinally
)
```

### Parameters

**source**
Type: `System.Collections.Concurrent,:::Partitioner<Of <(TSource)>)`
The Partitioner that contains the original data source.

**parallelOptions**
Type: `System.Threading.Tasks,:::ParallelOptions`
A `ParallelOptions` instance that configures the behavior of this operation.

**localInit**
Type: `System:::Func<Of <(TLocal)>)`
The function delegate that returns the initial state of the local data for each thread.

**body**
Type: `System:::Func<Of <(TSource, ParallelLoopState, TLocal, TLocal)>`
TLocal>)>
The delegate that is invoked once per iteration.

localFinally
Type: System...Action<(Of <(TLocal)>)
The delegate that performs a final action on the local state of each thread.
Type Parameters

TSource
   The type of the elements in source.
TLocal
   The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The Partitioner is used to retrieve the elements to be processed, in place of the original data source. If the current element's index is desired, the source must be an OrderablePartitioner.

The body delegate is invoked once for each element in the source Partitioner. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the SupportsDynamicPartitions property in the source Partitioner returns false.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when any methods in the source Partitioner return null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner does not return the correct number of partitions.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetPartitions() method in the source Partitioner returns an IList with at least one null value.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown when the GetDynamicPartitions() method in the source Partitioner returns an IEnumerable whose GetEnumerator()...</td>
</tr>
</tbody>
</table>
The exception that is thrown to contain an exception thrown from one of the specified delegates.

The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Parallel...:::ForEach<(Of <(TSource, TLocal)>)) Method (IEnumerable<(Of <(TSource)>), ParallelOptions, Func<(Of <(TLocal)>)), Func<(Of <(TSource, ParallelLoopState, TLocal, TLocal)>), Action<(Of <(TLocal)>))
Parallel Class  See Also  Send Feedback

Executes a for each operation on an IEnumerable<(Of <(TSource)>)) in which iterations may run in parallel.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource, TLocal) ( _
    source As IEnumerable(Of TSource), _
    parallelOptions As ParallelOptions, _
    localInit As Func(Of TLocal), _
    body As Func(Of TSource, ParallelLoopState, TLocal, TLocal),
    localFinally As Action(Of TLocal) _
) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource, TLocal>(
    IEnumerable<TSource> source,
    ParallelOptions parallelOptions,
    Func<TLocal> localInit,
    Func<TSource, ParallelLoopState, TLocal, TLocal> body,
    Action<TLocal> localFinally
)

Parameters

source
    Type: System.Collections.Generic.::.IEnumerable<(Of <(TSource)>)> An enumerable data source.

parallelOptions
    Type: System.Threading.Tasks.::.ParallelOptions A ParallelOptions instance that configures the behavior of this operation.

localInit
    Type: System.::.Func<(Of <(TLocal)>)> The function delegate that returns the initial state of the local data for each thread.

body
    Type: System.::.Func<(Of <(TSource, ParallelLoopState, TLocal, TLocal)> The function delegate that processes each element of the source.
TLocal

The delegate that is invoked once per iteration.

localFinally

Type: System..::.Action<Of <(TLocal)>>

The delegate that performs a final action on the local state of each thread.
Type Parameters

TSource
   The type of the data in the source.
TLocal
   The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the following parameters: the current element, a ParallelLoopState instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
Parallel..::.ForEach(Of (Of (TSource, TLocal>)) Method (IEnumerable<Of (TSource>), ParallelOptions, Func<Of (TLocal>), Func<Of (TSource, ParallelLoopState, Int64, TLocal, TLocal>), Action<Of (TLocal>))

**Parallel Class**  [See Also]  [Send Feedback]

Executes a for each operation on an IEnumerable<Of (TSource>)) in which iterations may run in parallel.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function ForEach(Of TSource, TLocal) ( _
    source As IEnumerable(Of TSource), _
    parallelOptions As ParallelOptions, _
    localInit As Func(Of TLocal), _
    body As Func(Of TSource, ParallelLoopState, Long, TLocal, TLocalFinally As Action(Of TLocal) _) ) As ParallelLoopResult

C#

public static ParallelLoopResult ForEach<TSource, TLocal>(
    IEnumerable<TSource> source,
    ParallelOptions parallelOptions,
    Func<TLocal> localInit,
    Func<TSource, ParallelLoopState, long, TLocal, TLocal> body,
    Action<TLocal> localFinally
)

Parameters

source
Type: System.Collections.Generic.IEnumerable<(Of ( Of TSource )>)
An enumerable data source.

parallelOptions
Type: System.Threading.Tasks.ParallelOptions
A ParallelOptions instance that configures the behavior of this operation.

localInit
Type: System.Func<(Of ( Of TLocal )>)
The function delegate that returns the initial state of the local data for each thread.

body
Type: System.Func<(Of ( Of TSource, ParallelLoopState, Int64, TLocal, _, (), _ )>)

TLocal>)>
The delegate that is invoked once per iteration.

localFinally
Type: System..::..Action<Of <(TLocal)>>
The delegate that performs a final action on the local state of each thread.
Type Parameters

TSource
The type of the data in the source.

TLocal
The type of the thread-local data.

Return Value

A ParallelLoopResult structure that contains information on what portion of the loop completed.
Remarks

The body delegate is invoked once for each element in the source enumerable. It is provided with the following parameters: the current element, a `ParallelLoopState` instance that may be used to break out of the loop prematurely, the current element's index (an Int64), and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each thread that participates in the loop's execution and returns the initial local state for each of those threads. These initial states are passed to the first body invocations on each thread. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each thread returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per thread to perform a final action on each thread's local state.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the source argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the body argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the localInit argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the localFinally argument is null.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions argument is set.</td>
</tr>
<tr>
<td><strong>System..::.AggregateException</strong></td>
<td>The exception that is thrown to contain an exception thrown from one of the specified delegates.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
ForEach Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
Parallel:::Invoke Method
Parallel Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Invoke(array&lt;Action&gt;[][])</code></td>
<td>Executes each of the provided actions, possibly in parallel.</td>
</tr>
<tr>
<td><code>Invoke(ParallelOptions, array&lt;Action&gt;[][])</code></td>
<td>Executes each of the provided actions, possibly in parallel.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes each of the provided actions, possibly in parallel.

**Namespace:** [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Sub Invoke ( _
    ParamArray actions As Action() _
)

C#

public static void Invoke(
    params Action[] actions
)

Parameters

actions
    Type: array<System..::.Action>[]
    An array of Actions to execute.
Remarks

This method can be used to execute a set of operations, potentially in parallel. No guarantees are made about the order in which the operations execute or whether they execute in parallel. This method does not return until each of the provided operations has completed, regardless of whether completion occurs due to normal or exceptional termination.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the actions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the actions array contains a null element.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown when any action in the actions array throws an exception.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
Invoke Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Executes each of the provided actions, possibly in parallel.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Sub Invoke ( _
    parallelOptions As ParallelOptions, _
    ParamArray actions As Action() _
)

C#

public static void Invoke(
    ParallelOptions parallelOptions,
    params Action[] actions
)

Parameters

parallelOptions
    Type: System.Threading.Tasks:::ParallelOptions
    A ParallelOptions instance that configures the behavior of this operation.

actions
    Type: array< System...:::Action >[][]
    An array of Actions to execute.
Remarks

This method can be used to execute a set of operations, potentially in parallel. No guarantees are made about the order in which the operations execute or whether they execute in parallel. This method does not return until each of the provided operations has completed, regardless of whether completion occurs due to normal or exceptional termination.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the actions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the parallelOptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the actions array contains a null element.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The exception that is thrown when the CancellationToken in the parallelOptions is set.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The exception that is thrown when any action in the actions array throws an exception.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when the CancellationTokenSource associated with the CancellationToken in the parallelOptions has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Parallel Class
Invoke Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Provides completion status on the execution of a [Parallel](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel) loop.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Structure ParallelLoopResult

**C#**

public struct ParallelLoopResult
Remarks

If $\text{IsCompleted}$ returns true, then the loop ran to completion, such that all iterations of the loop were executed. If $\text{IsCompleted}$ returns false and $\text{LowestBreakIteration}$ returns null, a call to $\text{Stop()}$ was used to end the loop prematurely. If $\text{IsCompleted}$ returns false and $\text{LowestBreakIteration}$ returns a non-null integral value, $\text{Break()}$ was used to end the loop prematurely.
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The **ParallelLoopResult** type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from ValueType.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from ValueType.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from ValueType.)</td>
</tr>
</tbody>
</table>
See Also

ParallelLoopResult Structure
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The **ParallelLoopResult** type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsCompleted</td>
<td>Gets whether the loop ran to completion, such that all iterations of the loop were executed and the loop didn't receive a request to end prematurely.</td>
</tr>
<tr>
<td>LowestBreakIteration</td>
<td>Gets the index of the lowest iteration from which Break()() was called.</td>
</tr>
</tbody>
</table>
See Also

ParallelLoopResult Structure
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets whether the loop ran to completion, such that all iterations of the loop were executed and the loop didn't receive a request to end prematurely.

**Namespace:**  System.Threading.Tasks

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property IsCompleted As Boolean

C#

public bool IsCompleted { get; }
See Also

ParallelLoopResult Structure
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
ParallelLoopResult::LowestBreakIteration Property

Gets the index of the lowest iteration from which `Break()` was called.

**Namespace**: System.Threading.Tasks  
**Assembly**: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property LowestBreakIteration As Nullable(Of Long)

C#

public Nullable<long> LowestBreakIteration { get; }
Remarks

If Break() was not employed, this property will return null.
See Also

ParallelLoopResult Structure
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
ParallelLoopState Class

Enables iterations of Parallel loops to interact with other iterations.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
                    ExternalThreading := True)> _
Public Class ParallelLoopState
```

#### C#

```csharp
[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
                    ExternalThreading = true)]
public class ParallelLoopState
```
Inheritance Hierarchy

System..::..Object
System.Threading.Tasks..::..ParallelLoopState
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `ParallelLoopState` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break</td>
<td>Communicates that the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel">Parallel</a> loop should cease execution at the system's earliest convenience of iterations beyond the current iteration.</td>
</tr>
</tbody>
</table>
| Equals     | (Inherited from `Object`.)
| Finalize   | (Inherited from `Object`).
| GetHashCode | (Inherited from `Object`.)
| GetType    | (Inherited from `Object`.)
| MemberwiseClone | (Inherited from `Object`).
| Stop       | Communicates that the [Parallel](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel) loop should cease execution at the system's earliest convenience. |
| ToString   | (Inherited from `Object`.)

See Also

ParallelLoopState Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Communicates that the `Parallel` loop should cease execution at the system's earliest convenience of iterations beyond the current iteration.

**Namespace:**  `System.Threading.Tasks`  
**Assembly:**  `System.Threading`  (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Public Sub Break

C#

public void Break()
Remarks

Break()() may be used to communicate to the loop that no other iterations after the current iteration need be run. For example, if Break()() is called from the 100th iteration of a for loop iterating in parallel from 0 to 1000, all iterations less than 100 should still be run, but the iterations from 101 through to 1000 are not necessary.

For long-running iterations that may already be executing, Break()() causes LowestBreakIteration to be set to the current iteration's index if the current index is less than the current value of LowestBreakIteration.

Break()() is typically employed in search-based algorithms where an ordering is present in the data source.
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The <strong>Stop</strong> method was previously called. <strong>Break</strong> and <strong>Stop</strong> may not be used in combination by iterations of the same loop.</td>
</tr>
</tbody>
</table>
See Also

ParallelLoopState Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Communicates that the `Parallel` loop should cease execution at the system's earliest convenience.

**Namespace:**  [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Sub Stop

C#
public void Stop()
Remarks

Stop()() may be used to communicate to the loop that no other iterations need be run. For long-running iterations that may already be executing, Stop()() causes IsStopped to return true for all other iterations of the loop, such that another iteration may check IsStopped and exit early if it's observed to be true.

Stop()() is typically employed in search-based algorithms, where once a result is found, no other iterations need be executed.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::..InvalidOperationException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.enumerations.break">Break()</a> method was previously called. <a href="https://docs.microsoft.com/en-us/dotnet/api/system.enumerations.break">Break()</a> and <a href="https://docs.microsoft.com/en-us/dotnet/api/system.enumerations.stop">Stop()</a> may not be used in combination by iterations of the same loop.</td>
</tr>
</tbody>
</table>
See Also

ParallelLoopState Class
System.Threading.Tasks Namespace

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
ParallelLoopState Properties

- **ParallelLoopState Class**  See Also  Send Feedback

The **ParallelLoopState** type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsExceptional</td>
<td>Gets whether any iteration of the loop has thrown an exception that went unhandled by that iteration.</td>
</tr>
<tr>
<td>IsStopped</td>
<td>Gets whether any iteration of the loop has called $\text{Stop()}$.</td>
</tr>
<tr>
<td>LowestBreakIteration</td>
<td>Gets the lowest iteration of the loop from which $\text{Break()}$ was called.</td>
</tr>
<tr>
<td>ShouldExitCurrentIteration</td>
<td>Gets whether the current iteration of the loop should exit based on requests made by this or other iterations.</td>
</tr>
</tbody>
</table>
See Also

ParallelLoopState Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets whether any iteration of the loop has thrown an exception that went unhandled by that iteration.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsExceptional As Boolean

C#

public bool IsExceptional { get; }
See Also

ParallelLoopState Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets whether any iteration of the loop has called `Stop()`.  

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsStopped As Boolean

C#

public bool IsStopped { get; }
See Also

ParallelLoopState Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
**ParallelLoopState::LowestBreakIteration Property**

Gets the lowest iteration of the loop from which `Break()` was called.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property LowestBreakIteration As Nullable(Of Long)

C#

public Nullable<long> LowestBreakIteration { get; }
Remarks

If no iteration of the loop called `Break()`, this property will return null.
See Also

ParallelLoopState Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets whether the current iteration of the loop should exit based on requests made by this or other iterations.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property ShouldExitCurrentIteration As Boolean

C#

class

public

bool

ShouldExitCurrentIteration

{

get;

}
Remarks

When an iteration of a loop calls `Break()` or `Stop()`, or when one throws an exception, or when the loop is canceled, the Parallel class will proactively attempt to prohibit additional iterations of the loop from starting execution. However, there may be cases where it is unable to prevent additional iterations from starting. It may also be the case that a long-running iteration has already begun execution. In such cases, iterations may explicitly check the `ShouldExitCurrentIteration` property and cease execution if the property returns true.
See Also

ParallelLoopState Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Stores options that configure the operation of methods on the `Parallel` class.

**Namespace:** [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Class ParallelOptions

C#

public class ParallelOptions
Remarks

By default, methods on the Parallel class attempt to utilize all available processors, are non-cancelable, and target the default TaskScheduler (TaskScheduler.Default). ParallelOptions enables overriding these defaults.
Inheritance Hierarchy

System...:::Object
System.Threading.Tasks...:::ParallelOptions
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `ParallelOptions` class.

**Namespace:**  `System.Threading.Tasks`  
**Assembly:**  `System.Threading` (in `System.Threading.dll`)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public ParallelOptions()
Remarks

This constructor initializes the instance with default values. `MaxDegreeOfParallelism` is initialized to -1, signifying that there is no upper bound set on how much parallelism should be employed. `CancellationToken` is initialized to a non-cancelable token, and `TaskScheduler` is initialized to the default scheduler (TaskScheduler.Default). All of these defaults may be overwritten using the property set accessors on the instance.
See Also

ParallelOptions Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `ParallelOptions` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
See Also

ParallelOptions Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `ParallelOptions` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancellationToken</td>
<td>Gets or sets the <a href="#">CancellationToken</a> associated with this <a href="#">ParallelOptions</a> instance.</td>
</tr>
<tr>
<td>MaxDegreeOfParallelism</td>
<td>Gets or sets the <a href="#">TaskScheduler</a> associated with this <a href="#">ParallelOptions</a> instance.</td>
</tr>
<tr>
<td>TaskScheduler</td>
<td>Setting this property to null indicates that the current scheduler should be used.</td>
</tr>
</tbody>
</table>
See Also

ParallelOptions Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets or sets the `CancellationToken` associated with this `ParallelOptions` instance.

**Namespace:** [System.Threading.Tasks](http://msdn.microsoft.com/en-us/library/system.threading.tasks.aspx)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Property CancellationToken As CancellationToken

C#

public CancellationToken CancellationToken { get; set; }
Remarks

Providing a `CancellationToken` to a `Parallel` method enables the operation to be exited early. Code external to the operation may cancel the token, and if the operation observes the token being set, it may exit early by throwing an `OperationCanceledException`.
See Also

ParallelOptions Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets or sets the maximum degree of parallelism enabled by this ParallelOptions instance.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Property MaxDegreeOfParallelism As Integer

C#

public int MaxDegreeOfParallelism { get; set; }
Remarks

The MaxDegreeOfParallelism limits the number of concurrent operations run by Parallel method calls that are passed this ParallelOptions instance to the set value, if it is positive. If MaxDegreeOfParallelism is -1, then there is no limit placed on the number of concurrently running operations.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when this MaxDegreeOfParallelism is set to 0 or some value less than -1.</td>
</tr>
</tbody>
</table>
See Also

ParallelOptions Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets or sets the TaskScheduler associated with this ParallelOptions instance. Setting this property to null indicates that the current scheduler should be used.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Property TaskScheduler As TaskScheduler

C#

public TaskScheduler TaskScheduler { get; set; }
See Also

ParallelOptions Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Represents an asynchronous operation.

**Namespace:**  [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading := True)> _
Public Class Task _
    Implements IAsyncResult, IDisposable

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
ExternalThreading = true)]
public class Task : IAsyncResult, IDisposable
Remarks

Task instances may be created in a variety of ways. The most common approach is by using the Task type's Factory property to retrieve a TaskFactory instance that can be used to create tasks for several purposes. For example, to create a Task that runs an action, the factory's StartNew method may be used:

```csharp
// C#
var t = Task.Factory.StartNew(() => DoAction());
```

The Task class also provides constructors that initialize the Task but that do not schedule it for execution. For performance reasons, TaskFactory's StartNew method should be the preferred mechanism for creating and scheduling computational tasks, but for scenarios where creation and scheduling must be separated, the constructors may be used, and the task's Start() method may then be used to schedule the task for execution at a later time.

All members of Task, except for Dispose(), are thread-safe and may be used from multiple threads concurrently.

For operations that return values, the Task(Of TResult) class should be used.

For developers implementing custom debuggers, several internal and private members of Task may be useful (these may change from release to release). The Int32 m_taskId field serves as the backing store for the Id property, however accessing this field directly from a debugger may be more efficient than accessing the same value through the property's getter method (the s_taskIdCounter Int32 counter is used to retrieve the next available ID for a Task). Similarly, the Int32 m_stateFlags field stores information about the current lifecycle stage of the Task, information also accessible through the Status
property. The m_action System.Object field stores a reference to the Task's delegate, and the m_stateObject System.Object field stores the async state passed to the Task by the developer. Finally, for debuggers that parse stack frames, the InternalWait method serves a potential marker for when a Task is entering a wait operation.
Inheritance Hierarchy

System...:::Object
System.Threading.Tasks...:::Task
    System.Threading.Tasks...:::Task(Of TResult)
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
Task Constructor

Task Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task(Action)</td>
<td>Initializes a new Task with the specified action.</td>
</tr>
<tr>
<td>Task(Action, CancellationToken)</td>
<td>Initializes a new Task with the specified action and CancellationToken.</td>
</tr>
<tr>
<td>Task(Action, TaskCreationOptions)</td>
<td>Initializes a new Task with the specified action and creation options.</td>
</tr>
<tr>
<td>Task(Action&lt;Of &lt;(Object)&gt;&gt;, Object)</td>
<td>Initializes a new Task with the specified action and state.</td>
</tr>
<tr>
<td>Task(Action, CancellationToken, TaskCreationOptions)</td>
<td>Initializes a new Task with the specified action and creation options.</td>
</tr>
<tr>
<td>Task(Action&lt;Of &lt;(Object)&gt;&gt;, Object, CancellationToken)</td>
<td>Initializes a new Task with the specified action, state, snd options.</td>
</tr>
<tr>
<td>Task(Action&lt;Of &lt;(Object)&gt;&gt;, Object, TaskCreationOptions)</td>
<td>Initializes a new Task with the specified action, state, snd options.</td>
</tr>
<tr>
<td>Task(Action&lt;Of &lt;(Object)&gt;&gt;, Object, CancellationToken, TaskCreationOptions)</td>
<td>Initializes a new Task with the specified action, state, snd options.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new Task with the specified action.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Sub New ( _
    action As Action _
)
```

**C#**

```csharp
public Task(
    Action action
)
```

### Parameters

**action**

- **Type:** System..::.Action
- The delegate that represents the code to execute in the Task.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The action argument is null.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Task Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new Task with the specified action and CancellationToken.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    action As Action, _
    cancellationToken As CancellationToken _
)

C#

public Task(
    Action action,
    CancellationToken cancellationToken
)

Parameters

action
    Type: System:::Action
    The delegate that represents the code to execute in the Task.

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken that will be assigned to the new Task.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The action argument is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Task Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new Task with the specified action and creation options.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Sub New (_
    action As Action, _
    creationOptions As TaskCreationOptions _
)
```

**C#**

```csharp
public Task(
    Action action,
    TaskCreationOptions creationOptions
)
```

**Parameters**

**action**
- Type: `System::Action`
  - The delegate that represents the code to execute in the task.

**creationOptions**
- Type: `System.Threading.Tasks::TaskCreationOptions`
  - The `TaskCreationOptions` used to customize the Task's behavior.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::..ArgumentNullException</td>
<td>The action argument is null.</td>
</tr>
<tr>
<td>System...::..ArgumentOutOfRangeException</td>
<td>The creationOptions argument specifies an invalid value for</td>
</tr>
<tr>
<td></td>
<td>TaskCreationOptions.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Task Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new `Task` with the specified action and state.

**Namespace:**  `System.Threading.Tasks`

**Assembly:**  `System.Threading` (in `System.Threading.dll`)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Sub New (  
    action As Action(Of Object),  
    state As Object  
)
```

**C#**

```csharp
public Task(
    Action<object> action,
    object state
)
```

**Parameters**

**action**

Type: System:::Action(Of (Of (Object)>))

The delegate that represents the code to execute in the task.

**state**

Type: System:::Object

An object representing data to be used by the action.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The action argument is null.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Task Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task Constructor (Action, CancellationToken, TaskCreationOptions)

Task Class

Initializes a new Task with the specified action and creation options.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New (_
    action As Action, _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions _
)

C#

public Task(
    Action action,
    CancellationToken cancellationToken, 
    TaskCreationOptions creationOptions
)

Parameters

action
  Type: System..::.Action
  The delegate that represents the code to execute in the task.

cancellationToken
  Type: System.Threading..::.CancellationToken
  The CancellationToken()()() that will be assigned to the new task.

creationOptions
  Type: System.Threading.Tasks..::.TaskCreationOptions
  The TaskCreationOptions used to customize the Task's behavior.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The action argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The creationOptions argument specifies an invalid value for TaskCreationOptions.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Task Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new Task with the specified action, state, and options.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Sub New (_
    action As Action(Of Object), _
    state As Object, _
    cancellationToken As CancellationToken _
)
```

#### C#

```csharp
public Task(
    Action<Object> action,
    Object state,
    CancellationToken cancellationToken
)
```

### Parameters

**action**
- Type: System..::.Action(Of (Of Object)>)
- The delegate that represents the code to execute in the task.

**state**
- Type: System..::.Object
- An object representing data to be used by the action.

**cancellationToken**
- Type: System.Threading..::.CancellationToken
- The CancellationToken()()() that will be assigned to the new task.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The action argument is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Task Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new `Task` with the specified action, state, and options.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    action As Action(Of Object), _
    state As Object, _
    creationOptions As TaskCreationOptions _
)

C#

public Task(
    Action<Object> action,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

action
    Type: System:::Action(Of (Of Object))
    The delegate that represents the code to execute in the task.

state
    Type: System:::Object
    An object representing data to be used by the action.

creationOptions
    Type: System.Threading.Tasks:::TaskCreationOptions
    The TaskCreationOptions used to customize the Task's behavior.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>The action argument is null.</td>
</tr>
<tr>
<td>System...::.ArgumentOutOfRangeException</td>
<td>The creationOptions argument specifies an invalid value for TaskCreationOptions.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Task Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task Constructor (Action(Of Object), Object, CancellationToken, TaskCreationOptions)

Initializes a new Task with the specified action, state, and options.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Sub New ( _
    action As Action(Of Object), _
    state As Object, _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions _
)
```

#### C#

```csharp
public Task(
    Action<Object> action,
    Object state,
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions
)
```

### Parameters

**action**

Type: `System:::Action(Of <(Object)>)`
The delegate that represents the code to execute in the task.

**state**

Type: `System:::Object`
An object representing data to be used by the action.

**cancellationToken**

Type: `System.Threading:::CancellationToken`
The CancellationToken()()() that will be assigned to the new task.

**creationOptions**

Type: `System.Threading.Tasks:::TaskCreationOptions`
The TaskCreationOptions used to customize the Task's behavior.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The action argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The creationOptions argument specifies an invalid value for TaskCreationOptions.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Task Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The Task type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContinueWith</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Dispose</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>RunSynchronously</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Start</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Wait</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>WaitAll</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>WaitAny</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace
Send feedback on this topic to Microsoft.
Visual Basic  □ C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
Task...:ContinueWith Method

Task Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContinueWith(Action(Of &lt;Task&gt;))</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;TResult&gt;)(Func(Of &lt;Task, TResult&gt;))</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;Task&gt;), CancellationToken</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;Task&gt;), TaskContinuationOptions</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;Task&gt;), TaskScheduler</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;TResult&gt;)(Func(Of &lt;Task, TResult&gt;), CancellationToken)</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;TResult&gt;)(Func(Of &lt;Task, TResult&gt;), TaskContinuationOptions)</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;TResult&gt;)(Func(Of &lt;Task, TResult&gt;), TaskScheduler)</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;TResult&gt;)(Func(Of &lt;Task, TResult&gt;), CancellationToken, TaskContinuationOptions, TaskScheduler)</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
<tr>
<td>ContinueWith(Of &lt;TResult&gt;)(Func(Of &lt;Task, TResult&gt;), CancellationToken, TaskContinuationOptions, TaskScheduler)</td>
<td>Creates a continuation that executes when the target Task completes.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation that executes when the target Task completes.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
```
Visual Basic (Declaration)

Public Function ContinueWith ( _
    continuationAction As Action(Of Task) _
) As Task

C#

public Task ContinueWith(
    Action<Task> continuationAction
)
```

**Parameters**

continuationAction

Type: System::Action(Of (Task))

An action to run when the Task completes. When run, the delegate will be passed the completed task as an argument.

**Return Value**

A new continuation Task.
Remarks

The returned Task will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation that executes when the target Task completes.

**Namespace:**  [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task-class)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith(Of TResult) ( _
    continuationFunction As Func(Of Task, TResult) _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWith<TResult>(
    Func<Task, TResult> continuationFunction
)

Parameters

continuationFunction
    Type: System...::.Func(Of (Task, TResult)>)
    A function to run when the Task completes. When run, the delegate will be
passed the completed task as an argument.
### Type Parameters

**TResult**
- The type of the result produced by the continuation.

**Return Value**

A new continuation `Task<Of <(TResult)>>`.
Remarks

The returned `Task(Of (TResult)>)` will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..:::ArgumentNullException</td>
<td>The <code>continuationFunction</code> argument is null.</td>
</tr>
<tr>
<td>System..:::ObjectDisposedException</td>
<td>The <code>Task</code> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library

Task...::ContinueWith Method (Action(Of (Of (Task)>(Task)), CancellationToken)

Task Class  See Also  Send Feedback

Creates a continuation that executes when the target Task completes.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith ( _
    continuationAction As Action(Of Task), _
    cancellationToken As CancellationToken _
) As Task

C#

public Task ContinueWith(
    Action<Task> continuationAction,
    CancellationToken cancellationToken
)

Parameters

continuationAction
    Type: System...::Action<(Of <(Task)>)
    An action to run when the Task completes. When run, the delegate will be passed the completed task as an argument.

cancellationToken
    Type: System.Threading...::CancellationToken
    The CancellationToken()() that will be assigned to the new continuation task.

Return Value

A new continuation Task.
Remarks

The returned Task will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System....:ArgumentNullException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System....:ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
<tr>
<td>System....:ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation that executes when the target Task completes.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWith ( _
    continuationAction As Action(Of Task), _
    continuationOptions As TaskContinuationOptions _
) As Task
```

**C#**

```csharp
public Task ContinueWith(
    Action<Task> continuationAction,
    TaskContinuationOptions continuationOptions
)
```

**Parameters**

- `continuationAction`
  - Type: `System..::.Action(Of (Task))`
  - An action to run when the `Task` completes. When run, the delegate will be passed the completed task as an argument.

- `continuationOptions`
  - Type: `System.Threading.Tasks..::.TaskContinuationOptions`
  - Options for when the continuation is scheduled and how it behaves. This includes criteria, such as `OnlyOnCanceled`, as well as execution options, such as `ExecuteSynchronously`.

**Return Value**

A new continuation `Task`. 
Remarks

The returned Task will not be scheduled for execution until the current task has completed. If the continuation criteria specified through the continuationOptions parameter are not met, the continuation task will be canceled instead of scheduled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The continuationOptions argument specifies an invalid value for TaskContinuationOptions.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
Task...::.ContinueWith Method (Action(Of (Of Task>), TaskScheduler)
Task Class  See Also  Send Feedback

Creates a continuation that executes when the target Task completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWith ( _
    continuationAction As Action(Of Task), _
    scheduler As TaskScheduler _
) As Task
```

**C#**

```csharp
public Task ContinueWith(
    Action<Task> continuationAction,
    TaskScheduler scheduler
)
```

### Parameters

**continuationAction**
- Type: `System::.Action(Of `(Task)`)`
- An action to run when the `Task` completes. When run, the delegate will be passed the completed task as an argument.

**scheduler**
- Type: `System.Threading.Tasks::.TaskScheduler`
- The `TaskScheduler` to associate with the continuation task and to use for its execution.

### Return Value

A new continuation `Task`. 
Remarks

The returned Task will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
.NET Framework Class Library
Task.ContinueWith<(Of <(TResult>>)>) Method (Func<(Of <(Task, TResult>>)), CancellationToken)

Task Class  See Also  Send Feedback

Creates a continuation that executes when the target Task completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith(Of TResult) ( _
    continuationFunction As Func(Of Task, TResult), _
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWith<TResult>(
    Func<Task, TResult> continuationFunction,
    CancellationToken cancellationToken
)

Parameters

continuationFunction
Type: System...::.Func<Of <(Task, TResult)>>)
A function to run when the Task completes. When run, the delegate will be passed the completed task as an argument.

cancellationToken
Type: System.Threading...::.CancellationToken
The CancellationToken()() that will be assigned to the new continuation task.
Type Parameters

TResult
The type of the result produced by the continuation.

Return Value

A new continuation Task<Of<ForResult>>.
Remarks

The returned `Task<Of <(TResult)>` will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a> has been disposed.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task...::ContinueWith(Of (TResult)>) Method (Func(Of (Task, TResult)>)(), TaskContinuationOptions)

Task Class  See Also  Send Feedback

Creates a continuation that executes when the target Task completes.

**Namespace:**  System.Threading.Tasks
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith(Of TResult) ( _
    continuationFunction As Func(Of Task, TResult), _
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWith<TResult>(
    Func<Task, TResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)

Parameters

continuationFunction
Type: System::Func(Of (Task, TResult))
A function to run when the Task completes. When run, the delegate will be passed the completed task as an argument.

continuationOptions
Type: System.Threading.Tasks::TaskContinuationOptions
Options for when the continuation is scheduled and how it behaves. This includes criteria, such as OnlyOnCanceled, as well as execution options, such as ExecuteSynchronously.
Type Parameters

TResult
   The type of the result produced by the continuation.

Return Value

A new continuation Task<(Of <(TResult)>)>.
Remarks

The returned `Task<Of<TResult>)>` will not be scheduled for execution until the current task has completed. If the continuation criteria specified through the `continuationOptions` parameter are not met, the continuation task will be canceled instead of scheduled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The continuationOptions argument specifies an invalid value for TaskContinuationOptions.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation that executes when the target Task completes.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function ContinueWith(Of TResult) ( _
    continuationFunction As Func(Of Task, TResult), _
    scheduler As TaskScheduler _
) As Task(Of TResult)
```

#### C#

```csharp
public Task<TResult> ContinueWith<TResult>(
    Func<Task, TResult> continuationFunction,
    TaskScheduler scheduler
)
```

### Parameters

**continuationFunction**
- Type: `System...::Func<(Of (Task, TResult)>)`
  - A function to run when the `Task` completes. When run, the delegate will be passed the completed task as an argument.

**scheduler**
- Type: `System.Threading.Tasks...::TaskScheduler`
  - The `TaskScheduler` to associate with the continuation task and to use for its execution.
Type Parameters

TResult
   The type of the result produced by the continuation.

Return Value

A new continuation Task<Of<(TResult)>>. 
Remarks

The returned $\text{Task}\langle\text{Of } \langle\text{TResult}\rangle\rangle$ will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
Task...::ContinueWith Method (Action(Of Task), CancellationToken, TaskContinuationOptions, TaskScheduler)

Task Class  See Also  Send Feedback

Creates a continuation that executes when the target Task completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWith ( _
    continuationAction As Action(Of Task), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task
```

**C#**

```csharp
public Task ContinueWith(
    Action<Task> continuationAction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)
```

### Parameters

**continuationAction**
- **Type:** System...:::Action<Of <Task>>
- An action to run when the `Task` completes. When run, the delegate will be passed the completed task as an argument.

**cancellationToken**
- **Type:** System.Threading...:::CancellationToken
- The CancellationToken()() that will be assigned to the new continuation task.

**continuationOptions**
- **Type:** System.Threading.Tasks...:::TaskContinuationOptions
- Options for when the continuation is scheduled and how it behaves. This includes criteria, such as `OnlyOnCanceled`, as well as execution options, such as `ExecuteSynchronously`. 
scheduler

Type: System.Threading.Tasks.TaskScheduler

The TaskScheduler to associate with the continuation task and to use for its execution.

**Return Value**

A new continuation Task.
Remarks

The returned Task will not be scheduled for execution until the current task has completed. If the criteria specified through the continuationOptions parameter are not met, the continuation task will be canceled instead of scheduled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The continuationOptions argument specifies an invalid value for TaskContinuationOptions.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task...:::ContinueWith(Of<TResult>[]) Method (Func(Of(Task, TResult)[]), CancellationToken, TaskContinuationOptions, TaskScheduler)

Task Class  See Also  Send Feedback

Creates a continuation that executes when the target Task completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWith(Of TResult) ( _
    continuationFunction As Func(Of Task, TResult), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _) As Task(Of TResult)
```

**C#**

```csharp
public Task<TResult> ContinueWith<TResult>(
    Func<Task, TResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)
```

**Parameters**

- **continuationFunction**
  Type: System..::.Func(Of (Task, TResult)>)
  A function to run when the Task completes. When run, the delegate will be passed the completed task as an argument.

- **cancellationToken**
  Type: System.Threading,.::.CancellationToken
  The CancellationToken()() that will be assigned to the new continuation task.

- **continuationOptions**
  Type: System.Threading.Tasks,.::.TaskContinuationOptions
  Options for when the continuation is scheduled and how it behaves. This includes criteria, such as OnlyOnCanceled, as well as execution options, such as ExecuteSynchronously.
scheduler
Type: `System.Threading.Tasks::*::TaskScheduler`
The `TaskScheduler` to associate with the continuation task and to use for its execution.
Type Parameters

**TResult**

The type of the result produced by the continuation.

**Return Value**

A new continuation `Task<(Of <(TResult)>)>`. 
Remarks

The returned `Task<(Of<(TResult)>)>` will not be scheduled for execution until the current task has completed. If the criteria specified through the `continuationOptions` parameter are not met, the continuation task will be canceled instead of scheduled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The continuationOptions argument specifies an invalid value for TaskContinuationOptions.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
Task...::Dispose Method
Task Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose()()</td>
<td>Disposes the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=netcore8">Task</a> resource.</td>
</tr>
<tr>
<td>Dispose(Boolean)</td>
<td>Disposes the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=netcore8">Task</a>, releasing all of its unmanaged resources.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task...:::Dispose Method

Disposes the Task, releasing all of its unmanaged resources.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Dispose

C#

public void Dispose()

Implements

IDisposable...:::Dispose()()
Remarks

Unlike most of the members of Task, this method is not thread-safe. Also, Dispose() may only be called on a Task that is in one of the final states: RanToCompletion, Faulted, or Canceled.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The exception that is thrown if the Task is not in one of the final states: RanToCompletion, Faulted, or Canceled.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Dispose Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Disposes the Task, releasing all of its unmanaged resources.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Overridable Sub Dispose ( _
    disposing As Boolean _
)

C#

protected virtual void Dispose(
    bool disposing
)

Parameters

disposing
    Type: System..::.Boolean
    A Boolean value that indicates whether this method is being called due to a call to Dispose().
Remarks

Unlike most of the members of Task, this method is not thread-safe.
See Also

Task Class
Dispose Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
Task...: RunSynchronously Method

Task Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunSynchronously()()()</td>
<td>Runs the <a href="https://www.example.com">Task</a> synchronously on the current <a href="https://www.example.com">TaskScheduler</a>.</td>
</tr>
<tr>
<td>RunSynchronously(TaskScheduler)</td>
<td>Runs the <a href="https://www.example.com">Task</a> synchronously on the <a href="https://www.example.com">scheduler</a> provided.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Runs the Task synchronously on the current TaskScheduler.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub RunSynchronously

C#

public void RunSynchronously()
Remarks

A task may only be started and run only once. Any attempts to schedule a task a second time will result in an exception.

Tasks executed with RunSynchronously() will be associated with the current TaskScheduler.

If the target scheduler does not support running this Task on the current thread, the Task will be scheduled for execution on the scheduler, and the current thread will block until the Task has completed execution.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.task">Task</a> is not in a valid state to be started. It may have already been started, executed, or canceled, or it may have been created in a manner that doesn't support direct scheduling.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.task">Task</a> instance has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
RunSynchronously Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Runs the Task synchronously on the scheduler provided.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub RunSynchronously ( _
    scheduler As TaskScheduler _
)"

C#

public void RunSynchronously(
    TaskScheduler scheduler
)

Parameters

scheduler
Type: System.Threading.Tasks.TaskScheduler
The scheduler on which to attempt to run this task inline.
Remarks

A task may only be started and run only once. Any attempts to schedule a task a second time will result in an exception.

If the target scheduler does not support running this Task on the current thread, the Task will be scheduled for execution on the scheduler, and the current thread will block until the Task has completed execution.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.InvalidOperationException</td>
<td>The Task is not in a valid state to be started. It may have already been started, executed, or canceled, or it may have been created in a manner that doesn't support direct scheduling.</td>
</tr>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The Task instance has been disposed.</td>
</tr>
<tr>
<td>System...::.ArgumentNullException</td>
<td>The scheduler parameter is null.</td>
</tr>
</tbody>
</table>
See Also

Task Class
RunSynchronously Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
Task:::Start Method

Task Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start()()</td>
<td>Starts the Task, scheduling it for execution to the current TaskScheduler.</td>
</tr>
<tr>
<td>Start(TaskScheduler)</td>
<td>Starts the Task, scheduling it for execution to the specified TaskScheduler.</td>
</tr>
</tbody>
</table>

*TaskScheduler*
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Starts the Task, scheduling it for execution to the current TaskScheduler.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Start

C#

public void Start()
Remarks

A task may only be started and run only once. Any attempts to schedule a task a second time will result in an exception.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::InvalidOperationException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=netframework-4.7.2">Task</a> is not in a valid state to be started. It may have already been started, executed, or canceled, or it may have been created in a manner that doesn't support direct scheduling.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=netframework-4.7.2">Task</a> instance has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Start Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Starts the Task, scheduling it for execution to the specified TaskScheduler.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub Start (_
    scheduler As TaskScheduler _
)  

C#

public void Start(_
    TaskScheduler scheduler
)  

Parameters

scheduler

Type: System.Threading.Tasks.TaskScheduler

The TaskScheduler with which to associate and execute this task.
Remarks

A task may only be started and run only once. Any attempts to schedule a task a second time will result in an exception.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System..::.InvalidOperationException</td>
<td>The Task is not in a valid state to be started. It may have already been started, executed, or canceled, or it may have been created in a manner that doesn't support direct scheduling.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The Task instance has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class  
Start Overload  
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic   C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
Task...: Wait Method

Task Class   See Also   Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Wait()</code></td>
<td>Waits for the <code>Task</code> to complete execution.</td>
</tr>
<tr>
<td><code>Wait(Int32)</code></td>
<td>Waits for the <code>Task</code> to complete execution.</td>
</tr>
<tr>
<td><code>Wait(CancellationToken)</code></td>
<td>Waits for the <code>Task</code> to complete execution.</td>
</tr>
<tr>
<td><code>Wait(TimeSpan)</code></td>
<td>Waits for the <code>Task</code> to complete execution.</td>
</tr>
<tr>
<td><code>Wait(Int32, CancellationToken)</code></td>
<td>Waits for the <code>Task</code> to complete execution.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for the `Task` to complete execution.

**Namespace:**  `System.Threading.Tasks`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub Wait

C#

public void Wait()
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.AggregateException</td>
<td>The Task was canceled -or- an exception was thrown during the execution of the Task.</td>
</tr>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Wait Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for the Task to complete execution.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Public Function Wait ( _
    millisecondsTimeout As Integer _
) As Boolean

**C#**

public bool Wait(  
    int millisecondsTimeout
)

**Parameters**

millisecondsTimeout  
    Type: System.Int32  
    The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

**Return Value**

true if the Task completed execution within the allotted time; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentOutOfRangeException</code></td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out. The <code>Task</code> was canceled -or- an exception was thrown during the execution of the <code>Task</code>.</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>The <code>Task</code> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Wait Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for the Task to complete execution.

**Namespace:** System-Threading.Tasks  
**Assembly:** System-Threading (in System-Threading.dll)
Syntax

Visual Basic (Declaration)

```vbnet
Public Sub Wait (_
    cancellationToken As CancellationToken _
)```

C#

```csharp
public void Wait(
    CancellationToken cancellationToken
)```

Parameters

cancellationToken


A CancellationToken()() to observe while waiting for the task to complete.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::OperationCanceledException</code></td>
<td>The cancellationToken was canceled.</td>
</tr>
<tr>
<td><code>System:::AggregateException</code></td>
<td>The Task was canceled -or- an exception was thrown during the execution of the Task.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Wait Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for the Task to complete execution.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    timeout As TimeSpan _
) As Boolean

C#

public bool Wait(
    TimeSpan timeout
)

Parameters

timeout
    Type: System..::.TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

ture if the Task completed execution within the allotted time; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::AggregateException</code></td>
<td>The Task was canceled -or- an exception was thrown during the execution of the Task.</td>
</tr>
<tr>
<td><code>System:::ArgumentOutOfRangeException</code></td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out -or- timeout is greater than MaxValue.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Wait Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for the `Task` to complete execution.

**Namespace:**  `System.Threading.Tasks`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Function Wait ( _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public bool Wait(
    int millisecondsTimeout,
    CancellationToken cancellationToken
)

Parameters

millisecondsTimeout
    Type: System:::Int32
    The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

cancellationToken
    Type: System:::Threading:::CancellationToken
    A CancellationToken()() to observe while waiting for the task to complete.

Return Value

true if the Task completed execution within the allotted time; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AggregateException</code></td>
<td>The <a href="https://docs.microsoft.com">Task</a> was canceled -or- an exception was thrown during the execution of the <a href="https://docs.microsoft.com">Task</a>.</td>
</tr>
<tr>
<td><code>ObjectDisposedException</code></td>
<td>The <a href="https://docs.microsoft.com">Task</a> has been disposed.</td>
</tr>
<tr>
<td><code>ArgumentOutOfRangeException</code></td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td><code>OperationCanceledException</code></td>
<td>The cancellationToken was canceled.</td>
</tr>
</tbody>
</table>
See Also

Task Class
Wait Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#

Include Protected Members
Include Inherited Members

.NET Framework Class Library

Task:::WaitAll Method

Task Class  See Also  Send Feedback
# Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WaitAll(array&lt;Task&gt;[][])</code></td>
<td>Waits for all of the provided <code>Task</code> objects to complete execution.</td>
</tr>
<tr>
<td><code>WaitAll(array&lt;Task&gt;[][], Int32)</code></td>
<td>Waits for all of the provided <code>Task</code> objects to complete execution.</td>
</tr>
<tr>
<td><code>WaitAll(array&lt;Task&gt;[][], CancellationToken)</code></td>
<td>Waits for all of the provided <code>Task</code> objects to complete execution.</td>
</tr>
<tr>
<td><code>WaitAll(array&lt;Task&gt;[][], TimeSpan)</code></td>
<td>Waits for all of the provided <code>Task</code> objects to complete execution.</td>
</tr>
<tr>
<td><code>WaitAll(array&lt;Task&gt;[][], Int32, CancellationToken)</code></td>
<td>Waits for all of the provided <code>Task</code> objects to complete execution.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
Task...::WaitAll Method (array<Task>[][])
Syntax

Visual Basic (Declaration)

Public Shared Sub WaitAll ( _
    ParamArray tasks As Task() _
)

C#

public static void WaitAll(
    params Task[] tasks
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task >[]
    An array of Task instances on which to wait.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td></td>
<td>At least one of the <a href="#">Task</a> instances was canceled -or- an exception was thrown during the execution of at least one of the <a href="#">Task</a> instances.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>The <a href="#">Task</a> has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for all of the provided `Task` objects to complete execution.

**Namespace:**  [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function WaitAll ( _
   tasks As Task(), _
   millisecondsTimeout As Integer _
) As Boolean

C#

public static bool WaitAll(
   Task[] tasks,
   int millisecondsTimeout
)

Parameters

tasks
   Type: array< System.Threading.Tasks.Task >[]
   An array of Task instances on which to wait.

millisecondsTimeout
   Type: System.Int32
   The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

Return Value

ture if all of the Task instances completed execution within the allotted time; otherwise, false.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>At least one of the Task instances was canceled -or- an exception was thrown during the execution of at least one of the Task instances.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for all of the provided Task objects to complete execution.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Shared Sub WaitAll ( _
    tasks As Task(), _
    cancellationToken As CancellationToken _
)
```

**C#**

```csharp
public static void WaitAll(
    Task[] tasks,
    CancellationToken cancellationToken
)
```

### Parameters

- **tasks**
  - Type: array< `System.Threading.Tasks.Task` >[]
  - An array of `Task` instances on which to wait.

- **cancellationToken**
  - Type: `System.Threading.CancellationToken`
  - A CancellationToken()() to observe while waiting for the tasks to complete.

### Return Value

- true if all of the `Task` instances completed execution within the allotted time; otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>At least one of the Task instances was canceled -or- an exception was thrown during the execution of at least one of the Task instances.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The cancellationToken was canceled.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for all of the provided Task objects to complete execution.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function WaitAll ( _
    tasks As Task(), _
    timeout As TimeSpan _
) As Boolean

C#

public static bool WaitAll(
    Task[] tasks,
    TimeSpan timeout
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task>[]
    An array of Task instances on which to wait.

timeout
    Type: System.TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

true if all of the Task instances completed execution within the allotted time;
otherwise, false.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System..::.AggregateException</td>
<td>At least one of the Task instances was canceled -or- an exception was thrown during the execution of at least one of the Task instances.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out -or- timeout is greater than MaxValue.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for all of the provided Task objects to complete execution.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function WaitAll ( _
    tasks As Task(), _
    millisecondsTimeout As Integer, _
    cancellationToken As CancellationToken _
) As Boolean

C#

public static bool WaitAll(
    Task[] tasks,
    int millisecondsTimeout,
    CancellationToken cancellationToken
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task >[]
    An array of Task instances on which to wait.

millisecondsTimeout
    Type: System:::Int32
    The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

cancellationToken
    Type: System.Threading:::CancellationToken
    A CancellationToken()() to observe while waiting for the tasks to complete.

Return Value

true if all of the Task instances completed execution within the allotted time; otherwise, false.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System:::AggregateException</td>
<td>At least one of the Task instances was canceled -or- an exception was thrown during the execution of at least one of the Task instances.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The cancellationToken was canceled.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
Task...::WaitAny Method
Task Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WaitAny(array&lt;Task&gt;[][])</code></td>
<td>Waits for any of the provided <code>Task</code> objects to complete execution.</td>
</tr>
<tr>
<td><code>WaitAny(array&lt;Task&gt;[][], Int32)</code></td>
<td>Waits for any of the provided <code>Task</code> objects to complete execution.</td>
</tr>
<tr>
<td><code>WaitAny(array&lt;Task&gt;[][], CancellationToken)</code></td>
<td>Waits for any of the provided <code>Task</code> objects to complete execution.</td>
</tr>
<tr>
<td><code>WaitAny(array&lt;Task&gt;[][], TimeSpan)</code></td>
<td>Waits for any of the provided <code>Task</code> objects to complete execution.</td>
</tr>
<tr>
<td><code>WaitAny(array&lt;Task&gt;[][], Int32, CancellationToken)</code></td>
<td>Waits for any of the provided <code>Task</code> objects to complete execution.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for any of the provided Task objects to complete execution.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function WaitAny ( _
    ParamArray tasks As Task() _
) As Integer

C#

public static int WaitAny(
    params Task[] tasks
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task >[]
    An array of Task instances on which to wait.

Return Value

The index of the completed task in the tasks array argument.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for any of the provided Task objects to complete execution.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Shared Function WaitAny ( _
    tasks As Task(), _
    millisecondsTimeout As Integer _
) As Integer

**C#**

public static int WaitAny(
    Task[] tasks,
    int millisecondsTimeout
)

**Parameters**

tasks
    Type: array< System.Threading.Tasks.Task >[]
    An array of Task instances on which to wait.

millisecondsTimeout
    Type: System.Int32
    The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

**Return Value**

The index of the completed task in the tasks array argument, or -1 if the timeout occurred.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for any of the provided `Task` objects to complete execution.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Shared Function WaitAny ( _
    tasks As Task(), _
    cancellationToken As CancellationToken _
) As Integer
```

**C#**

```csharp
public static int WaitAny(
    Task[] tasks,
    CancellationToken cancellationToken
)
```

**Parameters**

- **tasks**
  - Type: array< System.Threading.Tasks Хотя Task >[]()[]
  - An array of Task instances on which to wait.

- **cancellationToken**
  - Type: System.Threading.CancellationToken
  - A CancellationToken()() to observe while waiting for a task to complete.

**Return Value**

- The index of the completed task in the tasks array argument.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task">Task</a> has been disposed.</td>
</tr>
<tr>
<td>System..::.OperationCanceledException</td>
<td>The cancellationToken was canceled.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task...::WaitAny Method (array<Task>[][], TimeSpan)

Waits for any of the provided Task objects to complete execution.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function WaitAny ( _
    tasks As Task(), _
    timeout As TimeSpan _
) As Integer

C#

public static int WaitAny(
    Task[] tasks,
    TimeSpan timeout
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task >[]()
    An array of Task instances on which to wait.

timeout
    Type: System::TimeSpan
    A TimeSpan that represents the number of milliseconds to wait, or a
    TimeSpan that represents -1 milliseconds to wait indefinitely.

Return Value

The index of the completed task in the tasks array argument, or -1 if the timeout occurred.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The Task has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>timeout is a negative number other than -1 milliseconds, which represents an infinite time-out - or - timeout is greater than MaxValue.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Waits for any of the provided Task objects to complete execution.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Shared Function WaitAny ( _
  tasks As Task(), _
  millisecondsTimeout As Integer, _
  cancellationToken As CancellationToken _
) As Integer
```

#### C#

```csharp
public static int WaitAny(
  Task[] tasks,
  int millisecondsTimeout,
  CancellationToken cancellationToken
)
```

### Parameters

- **tasks**
  - Type: array< `System.Threading.Tasks.Task` >[]()
  - An array of `Task` instances on which to wait.

- **millisecondsTimeout**
  - Type: `System.Int32`
  - The number of milliseconds to wait, or Infinite (-1) to wait indefinitely.

- **cancellationToken**
  - Type: `System.Threading.CancellationToken`
  - A `CancellationToken` to observe while waiting for a task to complete.

### Return Value

The index of the completed task in the tasks array argument, or -1 if the timeout occurred.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The tasks argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The tasks argument contains a null element.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>millisecondsTimeout is a negative number other than -1, which represents an infinite time-out.</td>
</tr>
<tr>
<td>System:::OperationCanceledException</td>
<td>The cancellationToken was canceled.</td>
</tr>
</tbody>
</table>
See Also

Task Class
WaitAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The Task type exposes the following members.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsyncState</td>
<td>Gets the state object supplied when the Task was created, or null if none was supplied.</td>
</tr>
<tr>
<td>CreationOptions</td>
<td>Gets the TaskCreationOptions used to create this task.</td>
</tr>
<tr>
<td>CurrentId</td>
<td>Returns the unique ID of the currently executing Task.</td>
</tr>
<tr>
<td>Exception</td>
<td>Gets the Exception that caused the Task to end prematurely. If the Task completed successfully or has not yet thrown any exceptions, this will return null.</td>
</tr>
<tr>
<td>Factory</td>
<td>Provides access to factory methods for creating Task and Task(Of TResult) instances.</td>
</tr>
<tr>
<td>Id</td>
<td>Gets a unique ID for this Task instance.</td>
</tr>
<tr>
<td>IsCanceled</td>
<td>Gets whether this Task instance has completed execution due to being canceled.</td>
</tr>
<tr>
<td>IsCompleted</td>
<td>Gets whether this Task has completed.</td>
</tr>
<tr>
<td>IsFaulted</td>
<td>Gets whether the Task completed due to an unhandled exception.</td>
</tr>
<tr>
<td>Status</td>
<td>Gets the TaskStatus of this Task.</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAsyncResult::AsyncWaitHandle</td>
<td>Gets a WaitHandle that can be used to wait for the task to complete.</td>
</tr>
<tr>
<td>IAsyncResult::CompletedSynchronously</td>
<td>Gets an indication of whether the asynchronous operation completed synchronously.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets a WaitHandle that can be used to wait for the task to complete.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Private ReadOnly Property AsyncWaitHandle As WaitHandle
    Implements IAsyncResult.AsyncWaitHandle

C#

WaitHandle IAsyncResult.AsyncWaitHandle { get; }

Implements

IAsyncResult.AsyncWaitHandle
Remarks

Using the wait functionality provided by `Wait()` should be preferred over using `AsyncWaitHandle` for similar functionality.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets an indication of whether the asynchronous operation completed synchronously.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
%Syntax%

**Visual Basic (Declaration)**

Private Readonly Property CompletedSynchronously As Boolean
Implements IAsyncResult.CompletedSynchronously

**C#**

bool IAsyncResult.CompletedSynchronously { get; }

**Field Value**

true if the asynchronous operation completed synchronously; otherwise, false.

**Implements**

IAsyncResult...:.CompletedSynchronously
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the state object supplied when the Task was created, or null if none was supplied.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property AsyncState As Object

C#

public Object AsyncState { get; }

Implements

IAsyncResult.AsyncState
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the `TaskCreationOptions` used to create this task.

**Namespace:**  `System.Threading.Tasks`

**Assembly:**  `System.Threading (in System.Threading.dll)`
### Syntax

**Visual Basic (Declaration)**

Public ReadOnly Property CreationOptions As TaskCreationOptions

**C#**

public TaskCreationOptions CreationOptions { get; }
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Returns the unique ID of the currently executing `Task`.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared ReadOnly Property CurrentId As Nullable(Of Integer)

C#

public static Nullable<int> CurrentId { get; }
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the Exception that caused the Task to end prematurely. If the Task completed successfully or has not yet thrown any exceptions, this will return null.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Exception As AggregateException

C#

public AggregateException Exception { get; }
Remarks

Tasks that throw unhandled exceptions store the resulting exception and propagate it wrapped in a `AggregateException` in calls to `Wait` or in accesses to the `Exception` property. Any exceptions not observed by the time the Task instance is garbage collected will be propagated on the finalizer thread.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ObjectDisposedException</code></td>
<td>The Task has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Provides access to factory methods for creating `Task` and `Task<TResult>` instances.

**Namespace:**  `System.Threading.Tasks`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Shared ReadOnly Property Factory As TaskFactory

C#

public static TaskFactory Factory { get; }
Remarks

The factory returned from Factory is a default instance of TaskFactory, as would result from using the default constructor on TaskFactory.
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets a unique ID for this Task instance.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property Id As Integer

C#
public int Id { get; }
Remarks

Task IDs are assigned on-demand and do not necessarily represent the order in which Task instances were created.
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets whether this Task instance has completed execution due to being canceled.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsCanceled As Boolean

C#

public bool IsCanceled { get; }
Remarks

A Task will complete in Canceled state either if its CancellationToken was marked for cancellation before the task started executing, or if the task acknowledged the cancellation request on its already signaled CancellationToken by throwing an OperationCanceledException that bears the same CancellationToken.
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets whether this Task has completed.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsCompleted As Boolean

C#

public bool IsCompleted { get; }

Implements

IAsyncResult...::IsCompleted
Remarks

IsCompleted will return true when the Task is in one of the three final states: `RanToCompletion`, `Faulted`, or `Canceled`.
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets whether the `Task` completed due to an unhandled exception.

**Namespace:**  `System.Threading.Tasks`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public ReadOnly Property IsFaulted As Boolean

C#

public bool IsFaulted { get; }
Remarks

If IsFaulted is true, the Task’s Status will be equal to TaskStatus.Faulted, and its Exception property will be non-null.
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task.Status Property

Gets the TaskStatus of this Task.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Status As TaskStatus

C#

public TaskStatus Status { get; }
See Also

Task Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Represents an asynchronous operation that produces a result at some time in the future.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading := True)> _
Public Class Task(Of TResult) _
    Inherits Task

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, SynchronizationExternalThreading = true)]
public class Task<TResult> : Task
Type Parameters

TResult
  The type of the result produced by this Task<Of <(TResult)>>>.
Remarks

Task<Of (TResult)?> instances may be created in a variety of ways. The most common approach is by using the task's Factory property to retrieve a TaskFactory<Of (TResult)?> instance that can be used to create tasks for several purposes. For example, to create a Task<Of (TResult)?> that runs a function, the factory's StartNew method may be used:

```csharp
// C#
var t = Task<int>.Factory.StartNew(() => GenerateResult());
- or -
var t = Task.Factory.StartNew(() => GenerateResult());
```

The Task<Of (TResult)?> class also provides constructors that initialize the task but that do not schedule it for execution. For performance reasons, the StartNew method should be the preferred mechanism for creating and scheduling computational tasks, but for scenarios where creation and scheduling must be separated, the constructors may be used, and the task's Start method may then be used to schedule the task for execution at a later time.

All members of Task<Of (TResult)?>, except for Dispose, are thread-safe and may be used from multiple threads concurrently.
Inheritance Hierarchy

System..::.Object

System.Threading.Tasks..::.Task

System.Threading.Tasks..::.Task<

(TResult)>
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □ C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
Task<(Of <(TResult)>)> Constructor

Task<(Of <(TResult)>)> Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task(Of TResult&gt;(Func(Of TResult&gt;))</td>
<td>Initializes a new Task(Of TResult&gt; with the specified function.</td>
</tr>
<tr>
<td>Task(Of TResult&gt;(Func(Of TResult&gt;, CancellationToken)</td>
<td>Initializes a new Task(Of TResult&gt; with the specified function.</td>
</tr>
<tr>
<td>Task(Of TResult&gt;(Func(Of TResult&gt;, TaskCreationOptions)</td>
<td>Initializes a new Task(Of TResult&gt; with the specified function and creation options.</td>
</tr>
<tr>
<td>Task(Of TResult&gt;(Func(Of Object, TResult&gt;)</td>
<td>Initializes a new Task(Of TResult&gt; with the specified function and state.</td>
</tr>
<tr>
<td>Task(Of TResult&gt;(Func(Of Object, CancellationToken, TaskCreationOptions)</td>
<td>Initializes a new Task(Of TResult&gt; with the specified function and creation options.</td>
</tr>
<tr>
<td>Task(Of TResult&gt;(Func(Of Object, CancellationToken)</td>
<td>Initializes a new Task(Of TResult&gt; with the specified action, state, and options.</td>
</tr>
<tr>
<td>Task(Of TResult&gt;(Func(Of Object, CancellationToken, TaskCreationOptions)</td>
<td>Initializes a new Task(Of TResult&gt; with the specified action, state, and options.</td>
</tr>
<tr>
<td>Task(Of TResult&gt;(Func(Of Object, CancellationToken, TaskCreationOptions)</td>
<td>Initializes a new Task(Of TResult&gt; with the specified action, state, and options.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of (TResult)>>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new `Task(Of TResult>)` with the specified function.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Sub New ( _
    function As Func(Of TResult) _
)

C#

public Task(
    Func<TResult> function
)

Parameters

function
    Type: System...: Func(Of (TResult)>)
The delegate that represents the code to execute in the task. When the function has completed, the task's Result property will be set to return the result value of the function.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentException</code></td>
<td>The function argument is null.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of TResult)> Class
Task<(Of TResult)> Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new `Task(Of TResult)>(Of TResult)>` with the specified function.

**Namespace:**  `System.Threading.Tasks`  
**Assembly:**  `System.Threading.dll`
Syntax

Visual Basic (Declaration)

Public Sub New (_
    function As Func(Of TResult), _
    cancellationToken As CancellationToken _
)

C#

public Task(
    Func<TResult> function,
    CancellationToken cancellationToken
)

Parameters

function
    Type: System..::.Func<Of (TResult)>)
    The delegate that represents the code to execute in the task. When the function has completed, the task's Result property will be set to return the result value of the function.

cancellationToken
    Type: System.Threading..::.CancellationToken
    The CancellationToken to be assigned to this task.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentException</td>
<td>The function argument is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
Task(Of TResult>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic ▪ C#
.NET Framework Class Library
Task(Of TResult>) Constructor (Func(Of TResult>), TaskCreationOptions)
Task(Of TResult>) Class  See Also  Send Feedback

Initializes a new Task(Of TResult>) with the specified function and creation options.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    function As Func(Of TResult), _
    creationOptions As TaskCreationOptions _
)

C#

public Task(
    Func<TResult> function,
    TaskCreationOptions creationOptions
)

Parameters

function
    Type: System...::.Func<(Of <(TResult)>)
    The delegate that represents the code to execute in the task. When the function has completed, the task's Result property will be set to return the result value of the function.

creationOptions
    Type: System.Threading.Tasks...::.TaskCreationOptions
    The TaskCreationOptions used to customize the task's behavior.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentException</td>
<td>The function argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The creationOptions argument specifies an invalid value for TaskCreationOptions.</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
Task(Of TResult>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new `Task<Of<TResult>>` with the specified function and state.

**Namespace:** [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    function As Func(Of Object, TResult), _
    state As Object _
)

C#

public Task(
    Func<Object, TResult> function,
    Object state
)

Parameters

function
    Type: System::.Func<(Of <(Object, TResult)>)
    The delegate that represents the code to execute in the task. When the function has completed, the task's Result property will be set to return the result value of the function.

state
    Type: System::.Object
    An object representing data to be used by the action.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentException</td>
<td>The function argument is null.</td>
</tr>
</tbody>
</table>
### See Also

- Task(Of (TResult)>)
- Task(Of (TResult)>)
- Overload
- System.Threading.Tasks Namespace

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Initializes a new `Task(Of TResult)()` with the specified function and creation options.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    function As Func(Of TResult), _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions _
)  

C#

public Task(
    Func<TResult> function,
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions
)  

Parameters

function
  Type: System..::.Func<Of <(TResult)>>
  The delegate that represents the code to execute in the task. When the function has completed, the task's Result property will be set to return the result value of the function.

cancellationToken
  Type: System.Threading..::.CancellationToken
  The CancellationToken that will be assigned to the new task.

creationOptions
  Type: System.Threading.Tasks..::.TaskCreationOptions
  The TaskCreationOptions used to customize the task's behavior.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentException</td>
<td>The function argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The creationOptions argument specifies an invalid value for TaskCreationOptions.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
Task(Of TResult>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task<Of <(TResult)>>) Constructor (Func<Of <(Object, TResult)>), Object, CancellationToken)

Task<Of <(TResult)>>) Class  See Also  Send Feedback

Initializes a new Task<Of <(TResult)>>) with the specified action, state, and options.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Sub New ( _
    function As Func(Of Object, TResult), _
    state As Object, _
    cancellationToken As CancellationToken _
) _
```

**C#**

```csharp
public Task(
    Func<Object, TResult> function,
    Object state,
    CancellationToken cancellationToken
)
```

**Parameters**

**function**

Type: `System:::Func<(Of <(Object, TResult)>)`

The delegate that represents the code to execute in the task. When the function has completed, the task's `Result` property will be set to return the result value of the function.

**state**

Type: `System:::Object`

An object representing data to be used by the function.

**cancellationToken**

Type: `System.Threading:::CancellationToken`

The `CancellationToken` to be assigned to the new task.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentException</td>
<td>The function argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided <a href="https://example.com">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task(Of (TResult)>) Class
Task(Of (TResult)>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task(Of TResult>) Constructor (Func(Of (Object, TResult>), Object, TaskCreationOptions)

Task(Of TResult>) Class

Initializes a new Task(Of TResult>) with the specified action, state, and options.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    function As Func(Of Object, TResult), _
    state As Object, _
    creationOptions As TaskCreationOptions _
)

C#

public Task(
    Func<Object, TResult> function,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

function
    Type: System...::Func<(Of <(Object, TResult)>)
    The delegate that represents the code to execute in the task. When the function has completed, the task's Result property will be set to return the result value of the function.

state
    Type: System...::Object
    An object representing data to be used by the function.

creationOptions
    Type: System.Threading.Tasks...::TaskCreationOptions
    The TaskCreationOptions used to customize the task's behavior.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.EventArgsException</td>
<td>The function argument is null.</td>
</tr>
<tr>
<td>System.ArgumentOutOfRangeException</td>
<td>The creationOptions argument specifies an invalid value for</td>
</tr>
<tr>
<td></td>
<td>TaskCreationOptions.</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
Task(Of TResult>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task<(Of <(TResult)>)> Constructor (Func<(Of <(Object, TResult)>)>, Object, CancellationToken, TaskCreationOptions)

Task<(Of <(TResult)>)> Class  See Also  Send Feedback

Initializes a new Task<(Of <(TResult)>)> with the specified action, state, and options.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Visual Basic (Declaration)

Public Sub New ( _
    function As Func(Of Object, TResult), _
    state As Object, _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions _
)

C#

public Task(
    Func<Object, TResult> function,
    Object state,
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions
)

Parameters

function
    Type: System...:::Func<(Of <(Object, TResult)>)
    The delegate that represents the code to execute in the task. When the function has completed, the task's Result property will be set to return the result value of the function.

state
    Type: System...:::Object
    An object representing data to be used by the function.

cancellationToken
    Type: System.Threading...:::CancellationToken
    The CancellationToken to be assigned to the new task.

creationOptions
    Type: System.Threading.Tasks...:::TaskCreationOptions
The `TaskCreationOptions` used to customize the task's behavior.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:ArgumentException</td>
<td>The function argument is null.</td>
</tr>
<tr>
<td>System...:ArgumentOutOfRangeException</td>
<td>The creationOptions argument specifies an invalid value for TaskCreationOptions.</td>
</tr>
<tr>
<td>System...:ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
Task(Of TResult>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

Task<(Of <(TResult)>)> Methods

Task<(Of <(TResult)>)> Class

The Task<(Of <(TResult)>)> type exposes the following members.
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContinueWith</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Dispose</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>RunSynchronously</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Start</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Wait</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
Task<(Of (TResult)>).::.ContinueWith Method
Task<(Of (TResult)>).::.Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ContinueWith(Action&lt;Of &lt;(Task)&gt;&gt;)</code></td>
<td>Creates a continuation that executes when the target <code>Task</code> completes. (Inherited from <code>Task</code>.)</td>
</tr>
<tr>
<td><code>ContinueWith&lt;Of &lt;(TResult)&gt;&gt;(Func&lt;Of &lt;(Task, TResult)&gt;&gt;)</code></td>
<td>Creates a continuation that executes when the target <code>Task</code> completes. (Inherited from <code>Task</code>.)</td>
</tr>
<tr>
<td><code>ContinueWith&lt;Of &lt;(TResult)&gt;&gt;(Func&lt;Of &lt;(Task, TNewResult)&gt;&gt;)</code></td>
<td>Creates a continuation that executes when the target <code>Task&lt;Of &lt;(TResult)&gt;&gt;</code> completes.</td>
</tr>
<tr>
<td><code>ContinueWith&lt;Of &lt;(TNewResult)&gt;&gt;(Func&lt;Of &lt;(Task&lt;Of &lt;(TResult)&gt;&gt;, TNewResult)&gt;&gt;)</code></td>
<td>Creates a continuation that executes when the target <code>Task&lt;Of &lt;(TResult)&gt;&gt;</code> completes.</td>
</tr>
<tr>
<td><code>ContinueWith&lt;Of &lt;(Task)&gt;&gt;, CancellationToken&gt;</code></td>
<td>Creates a continuation that executes when the target <code>Task</code> completes. (Inherited from <code>Task</code>.)</td>
</tr>
<tr>
<td><code>ContinueWith&lt;Of &lt;(Task)&gt;&gt;, TaskContinuationOptions&gt;</code></td>
<td>Creates a continuation that executes when the target <code>Task</code> completes. (Inherited from <code>Task</code>.)</td>
</tr>
<tr>
<td><code>ContinueWith&lt;Of &lt;(Task)&gt;&gt;, TaskScheduler&gt;</code></td>
<td>Creates a continuation that executes when the target <code>Task</code> completes. (Inherited from <code>Task</code>.)</td>
</tr>
<tr>
<td><code>ContinueWith&lt;Of &lt;(TResult)&gt;&gt;(Func&lt;Of &lt;(Task, TResult)&gt;&gt;)</code></td>
<td>Creates a continuation that executes when the target <code>Task</code> completes. (Inherited from <code>Task</code>.)</td>
</tr>
<tr>
<td>Task&lt;TResult, CancellationToken&gt;</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes. (Inherited from \texttt{Task}.)</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ContinueWith(Of TResult&gt;(Func(Of Task, TResult&gt;), TaskContinuationOptions)</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes. (Inherited from \texttt{Task}.)</td>
</tr>
<tr>
<td>ContinueWith(Of TResult&gt;(Func(Of Task, TResult&gt;), TaskScheduler)</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes. (Inherited from \texttt{Task}.)</td>
</tr>
<tr>
<td>ContinueWith(Action(Of Task(Of TResult&gt;)[&gt;]), CancellationToken)</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes.</td>
</tr>
<tr>
<td>ContinueWith(Action(Of Task(Of TResult&gt;)[&gt;]), TaskContinuationOptions)</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes.</td>
</tr>
<tr>
<td>ContinueWith(Action(Of Task(Of TResult&gt;)[&gt;]), TaskScheduler)</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes.</td>
</tr>
<tr>
<td>ContinueWith(Of TNewResult&gt;(Func(Of Task(Of TResult&gt;)[&gt;], TNewResult&gt;), CancellationToken)</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes.</td>
</tr>
<tr>
<td>ContinueWith(Of TNewResult&gt;(Func(Of Task(Of TResult&gt;)[&gt;], TNewResult&gt;), TaskContinuationOptions)</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes.</td>
</tr>
<tr>
<td>ContinueWith(Of TNewResult&gt;)</td>
<td>Creates a continuation that executes when the target \texttt{Task} completes.</td>
</tr>
</tbody>
</table>
(Func<(Of <(Task<(Of <(TResult>>)>, TNewResult)>>), TaskScheduler) target Task<(Of <(TResult>>)>
completes.

ContinueWith(Action<(Of <(Task)>>), CancellationToken,
TaskContinuationOptions, TaskScheduler)
Creates a continuation that executes when the target Task completes.
(Inherited from Task.)

ContinueWith<(Of <(TResult)>>)>(Func<(Of <(Task, TResult)>>), CancellationToken,
TaskContinuationOptions, TaskScheduler)
Creates a continuation that executes when the target Task completes.
(Inherited from Task.)

ContinueWith<(Of <(TNewResult)>>)
(Func<(Of <(Task<(Of <(TResult)))>, TNewResult)>>), CancellationToken,
TaskContinuationOptions, TaskScheduler)
Creates a continuation that executes when the target Task completes.
See Also

Task<(Of (TResult)>>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task(Of TResult>)(::ContinueWith Method (Action(Of Task(Of TResult>)>)))

Task(Of TResult>) Class  See Also  Send Feedback

Creates a continuation that executes when the target Task(Of TResult>) completes.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith ( _
    continuationAction As Action(Of Task(Of TResult)) _
) As Task

C#

public Task ContinueWith(
    Action<Task<TResult>> continuationAction
)

Parameters

continuationAction
    Type: System...:Action<(Of <(Task<(Of <(Task>)>(Of TResult)>>)>)
    An action to run when the Task<(Of <(Task)>)> completes. When run, the delegate will be passed the completed task as an argument.

Return Value

A new continuation Task.
Remarks

The returned Task will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The Task&lt;Of &lt;(TResult)&gt; &gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation that executes when the target `Task<(Of <(TResult)>)>` completes.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith(Of TNewResult) ( _
    continuationFunction As Func(Of Task(Of TResult), TNewResult)_
) As Task(Of TNewResult)

C#

public Task<TNewResult> ContinueWith<TNewResult>(
    Func<Task<TResult>, TNewResult> continuationFunction
)

Parameters

continuationFunction
    Type: System...::Func(Of (Of Task(Of TResult)()
A function to run when the Task(Of TResult)() completes. When run, the delegate will be passed the completed task as an argument.
### Type Parameters

**TNewResult**

The type of the result produced by the continuation.

### Return Value

A new continuation `Task<Of<(TResult)>>`. 
The returned `Task<Of<(TResult)>>()` will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The Task(Of TResult)&gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task<Of <(TResult)>>.ContinueWith Method (Action<Of <(Task<Of <(TResult)>>)>, CancellationToken)

Task<Of <(TResult)>>.Class

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)

Creates a continuation that executes when the target Task<Of <(TResult)>>
completes.
Syntax

Visual Basic (Declaration)

Public Function ContinueWith ( _
    continuationAction As Action(Of Task(Of TResult)), _
    cancellationToken As CancellationToken _
) As Task

C#

public Task ContinueWith(
    Action<Task<TResult>> continuationAction,
    CancellationToken cancellationToken
)

Parameters

continuationAction
    Type: System...:::Action<(Of Task<(Of TResult)>)>)
    An action to run when the Task<(Of TResult)> completes. When run, the delegate will be passed the completed task as an argument.

cancellationToken
    Type: System.Threading...:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.

Return Value

A new continuation Task.
Remarks

The returned `Task` will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The <code>Task&lt;T&gt;(Of&lt;TResult&gt;)</code> has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided <code>CancellationToken</code> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task(Of TResult&gt;).:::ContinueWith Method (Action(Of Task(Of TResult&gt;))&, TaskContinuationOptions)

Task(Of TResult&gt;) Class

Creates a continuation that executes when the target Task(Of TResult&gt;) completes.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith ( _
    continuationAction As Action(Of Task(Of TResult)), _
    continuationOptions As TaskContinuationOptions _
) As Task

C#

public Task ContinueWith(
    Action<Task<TResult>> continuationAction,
    TaskContinuationOptions continuationOptions
)

Parameters

continuationAction
    Type: System...::Action<(Of <(Task<(Of <(TResult)>)>))>)
    An action to run when the Task<(Of <(TResult)>)> completes. When run, the delegate will be passed the completed task as an argument.

continuationOptions
    Type: System.Threading.Tasks...::TaskContinuationOptions
    Options for when the continuation is scheduled and how it behaves. This includes criteria, such as OnlyOnCanceled, as well as execution options, such as ExecuteSynchronously.

Return Value

A new continuation Task.
Remarks

The returned Task will not be scheduled for execution until the current task has completed. If the continuation criteria specified through the continuationOptions parameter are not met, the continuation task will be canceled instead of scheduled.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The continuationOptions argument specifies an invalid value for TaskContinuationOptions.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The Task&lt;Of &lt;TResult&gt;&gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task<*(Of *(TResult*>)*>) Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task(Of TResult&gt;)::.ContinueWith Method (Action(Of (Task(Of TResult&gt;)&gt;)&gt;), TaskScheduler)

Task(Of TResult&gt;) Class  See Also  Send Feedback

Creates a continuation that executes when the target Task(Of TResult&gt;) completes.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith (_
    continuationAction As Action(Of Task(Of TResult)), _
    scheduler As TaskScheduler _) As Task

C#

public Task ContinueWith(
    Action<Task<TResult>> continuationAction,
    TaskScheduler scheduler
)

Parameters

continuationAction
Type: System...:::Action(Of (Of TResult)>)>
An action to run when the Task(Of TResult)> completes. When run, the delegate will be passed the completed task as an argument.

scheduler
Type: System.Threading.Tasks...:::TaskScheduler
The TaskScheduler to associate with the continuation task and to use for its execution.

Return Value

A new continuation Task.
Remarks

The returned Task will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The <code>Task&lt;Of &lt;(TResult)&gt;&gt;</code> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task<Of TResult>) Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#  
.NET Framework Class Library

Task(Of TResult>).ContinueWith(Of TNewResult) Method (Func(Of Task(Of TResult), TNewResult), CancellationToken)  

Task(Of TResult) Class  See Also  Send Feedback

Creates a continuation that executes when the target Task(Of TResult) completes.

Namespace: System.Threading.Tasks  
Assembly: System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function ContinueWith(Of TNewResult) (continuationFunction As Func(Of Task(Of TResult), TNewResult), cancellationToken As CancellationToken) _
) As Task(Of TNewResult)
```

#### C#

```csharp
public Task<TNewResult> ContinueWith<TNewResult>(
    Func<Task<TResult>, TNewResult> continuationFunction,
    CancellationToken cancellationToken
)
```

### Parameters

**continuationFunction**
- Type: `System:::Func(Of (Task(Of TResult)>), TNewResult>)`
- A function to run when the `Task(Of TResult)>`) completes. When run, the delegate will be passed the completed task as an argument.

**cancellationToken**
- Type: `System.Threading:::CancellationToken`
- The `CancellationToken` that will be assigned to the new task.
Type Parameters

TNewResult
   The type of the result produced by the continuation.

Return Value

A new continuation `Task<Of <(TResult)>>`.
Remarks

The returned `Task<(Of <(TResult)>)>` will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task&lt;(&lt;TResult&gt;)&gt; has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of<TResult>>>) Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
Task<(Of <(TResult)>)>..::.ContinueWith<(Of <(TNewResult)>)> Method (Func<(Of <(Task<(Of <(TResult)>)>) TNewResult)>)
TaskContinuationOptions)
Task<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates a continuation that executes when the target Task<(Of <(TResult)>)>
completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWith(Of TNewResult) ( ___
    continuationFunction As Func(Of Task(Of TResult), TNewResult ___
    continuationOptions As TaskContinuationOptions ___
) As Task(Of TNewResult)
```

**C#**

```csharp
public Task<TNewResult> ContinueWith<TNewResult>(
    Func<Task<TResult>, TNewResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)
```

**Parameters**

continuationFunction

Type: System...::.Func<Of <(Task<Of <(TResult)>)>, TNewResult>>)>
A function to run when the Task<Of <(TResult)>)> completes. When run, the delegate will be passed the completed task as an argument.

continuationOptions

Type: System.Threading.Tasks...::.TaskContinuationOptions
Options for when the continuation is scheduled and how it behaves. This includes criteria, such as OnlyOnCanceled, as well as execution options, such as ExecuteSynchronously.
Type Parameters

TNewResult
   The type of the result produced by the continuation.

Return Value

A new continuation Task<(Of <(TResult)>)>.
Remarks

The returned `Task<Of (TResult)>` will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.

The continuationFunction, when executed, should return a `Task<Of (TResult)>`. This task's completion state will be transferred to the task returned from the ContinueWith call.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The continuationOptions argument specifies an invalid value for TaskContinuationOptions.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task&lt;TResult&gt; has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
Task(Of TResult).::.ContinueWith(Of TResult) Method
(Func(Of Task(Of TResult), TResult), TaskScheduler)
Task(Of TResult) Class  See Also  Send Feedback

Creates a continuation that executes when the target Task(Of TResult) completes.

Namespace: System Threading Tasks
Assembly: System Threading (in System Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith(Of TNewResult) (_
    continuationFunction As Func(Of Task(Of TResult), TNewResult,
        scheduler As TaskScheduler _
) As Task(Of TNewResult)

C#

public Task<TNewResult> ContinueWith<TNewResult>(
    Func<Task<TResult>, TNewResult> continuationFunction,
    TaskScheduler scheduler
)

Parameters

continuationFunction
    Type: System...::.Func<Of <(Task<Of <(TResult)>)>, TNewResult>)>
    A function to run when the Task<Of <(TResult)>> completes. When run,
    the delegate will be passed the completed task as an argument.

scheduler
    Type: System.Threading.Tasks...::.TaskScheduler
    The TaskScheduler to associate with the continuation task and to use for its
    execution.
Type Parameters

TNewResult
   The type of the result produced by the continuation.

Return Value

A new continuation Task<(Of <(TResult)>)>.
Remarks

The returned `Task<(Of <(TResult)>)>` will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The <code>Task&lt;Out(TResult)&gt;</code> has been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
.NET Framework Class Library
Task<(Of '<TResult>')>::ContinueWith Method (Action<(Of '<Task<(Of '<TResult>')>')), CancellationToken, TaskContinuationOptions, TaskScheduler)

Task<(Of '<TResult>')> Class  See Also  Send Feedback

Creates a continuation that executes when the target Task<(Of '<TResult>')> completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function ContinueWith ( _
    continuationAction As Action(Of Task(Of TResult)), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task
```

### C#

```csharp
public Task ContinueWith(  
    Action<Task<TResult>> continuationAction,  
    CancellationToken cancellationToken,  
    TaskContinuationOptions continuationOptions,  
    TaskScheduler scheduler
)
```

## Parameters

**continuationAction**

Type: System::Action(Of Task(Of TResult))

An action to run when the Task(Of TResult) completes. When run, the delegate will be passed the completed task as an argument.

**cancellationToken**

Type: System.Threading::CancellationToken

The CancellationToken that will be assigned to the new continuation task.

**continuationOptions**

Type: System.Threading.Tasks::TaskContinuationOptions

Options for when the continuation is scheduled and how it behaves. This includes criteria, such as OnlyOnCanceled, as well as execution options, such as ExecuteSynchronously.

**scheduler**
Type: `System.Threading.Tasks.TaskScheduler`

The `TaskScheduler` to associate with the continuation task and to use for its execution.

**Return Value**

A new continuation `Task`. 
Remarks

The returned Task will not be scheduled for execution until the current task has completed. If the criteria specified through the continuationOptions parameter are not met, the continuation task will be canceled instead of scheduled.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The continuationOptions argument specifies an invalid value for TaskContinuationOptions.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task(Of TResult) has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Task<(<TResult>)>.ContinueWith<(<TNewResult>)> Method (Func<(<Task<(<TResult>)>), TNewResult>), CancellationToken, TaskContinuationOptions, TaskScheduler)

Task<(<TResult>)> Class  See Also  Send Feedback

Creates a continuation that executes when the target Task<(<TResult>)> completes.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWith(Of TNewResult) ( _
    continuationFunction As Func(Of Task(Of TResult), TNewResult)
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TNewResult)

C#

public Task<TNewResult> ContinueWith<TNewResult>(
    Func<Task<TResult>, TNewResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

continuationFunction
Type: System..::.Func(Of @(Task<(@(Of (TResult))@)>), TNewResult@)>)
A function to run when the Task<(@(Of <(TResult)@)>)> completes. When run, the delegate will be passed as an argument this completed task.

cancellationToken
Type: System.Threading..::.CancellationToken
The CancellationToken that will be assigned to the new task.

continuationOptions
Type: System.Threading.Tasks..::.TaskContinuationOptions
Options for when the continuation is scheduled and how it behaves. This includes criteria, such as OnlyOnCanceled, as well as execution options, such as ExecuteSynchronously.

scheduler
Type: `System.Threading.Tasks.TaskScheduler`
The `TaskScheduler` to associate with the continuation task and to use for its execution.
Type Parameters

TNewResult
The type of the result produced by the continuation.

Return Value

A new continuation `Task<(Of <(TResult)>)>`. 
Remarks

The returned `Task<Of<TResult>>` will not be scheduled for execution until the current task has completed, whether it completes due to running to completion successfully, faulting due to an unhandled exception, or exiting out early due to being canceled.

The continuationFunction, when executed, should return a `Task<Of<TResult>>`. This task's completion state will be transferred to the task returned from the ContinueWith call.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The continuationOptions argument specifies an invalid value for TaskContinuationOptions.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task\langle TResult\rangle has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
ContinueWith Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#  Include Protected Members  Include Inherited Members
.NET Framework Class Library
Task<(Of <(TResult)>)>:::Dispose Method
Task<(Of <(TResult)>)> Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Dispose()()" /></td>
<td>Disposes the <a href="#">Task</a>, releasing all of its unmanaged resources. (Inherited from <a href="#">Task</a>.)</td>
</tr>
<tr>
<td><img src="image" alt="Dispose(Boolean)" /></td>
<td>Disposes the <a href="#">Task</a>, releasing all of its unmanaged resources. (Inherited from <a href="#">Task</a>.)</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
Task<Of (TResult)>::RunSynchronously Method
Task<Of (TResult)> Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunSynchronously()()()</td>
<td>Runs the Task synchronously on the current TaskScheduler. (Inherited from Task.)</td>
</tr>
<tr>
<td>RunSynchronously(TaskScheduler)</td>
<td>Runs the Task synchronously on the scheduler provided. (Inherited from Task.)</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start()()</td>
<td>Starts the Task, scheduling it for execution to the current TaskScheduler.</td>
</tr>
<tr>
<td></td>
<td>(Inherited from Task.)</td>
</tr>
<tr>
<td>Start(TaskScheduler)</td>
<td>Starts the Task, scheduling it for execution to the specified TaskScheduler.</td>
</tr>
<tr>
<td></td>
<td>(Inherited from Task.)</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic □ C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
Task(Of TResult>).::.Wait Method

Task(Of TResult). Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Wait()()</code></td>
<td>Waits for the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a> to complete execution. (Inherited from <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a>.)</td>
</tr>
<tr>
<td><code>Wait(Int32)</code></td>
<td>Waits for the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a> to complete execution. (Inherited from <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a>.)</td>
</tr>
<tr>
<td><code>Wait(CancellationToken)</code></td>
<td>Waits for the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a> to complete execution. (Inherited from <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a>.)</td>
</tr>
<tr>
<td><code>Wait(TimeSpan)</code></td>
<td>Waits for the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a> to complete execution. (Inherited from <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a>.)</td>
</tr>
<tr>
<td><code>Wait(Int32, CancellationToken)</code></td>
<td>Waits for the <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a> to complete execution. (Inherited from <a href="https://docs.microsoft.com/en-us/dotnet/api/system.threading.task">Task</a>.)</td>
</tr>
</tbody>
</table>
See Also

Task<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `Task<(Of <(TResult)>)>` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsyncState</td>
<td>Gets the state object supplied when the Task was created, or null if none was supplied. (Inherited from Task.)</td>
</tr>
<tr>
<td>CreationOptions</td>
<td>Gets the TaskCreationOptions used to create this task. (Inherited from Task.)</td>
</tr>
<tr>
<td>Exception</td>
<td>Gets the Exception that caused the Task to end prematurely. If the Task completed successfully or has not yet thrown any exceptions, this will return null. (Inherited from Task.)</td>
</tr>
<tr>
<td>Factory</td>
<td>Provides access to factory methods for creating Task(Of TResult) instances.</td>
</tr>
<tr>
<td>Id</td>
<td>Gets a unique ID for this Task instance. (Inherited from Task.)</td>
</tr>
<tr>
<td>IsCanceled</td>
<td>Gets whether this Task instance has completed execution due to being canceled. (Inherited from Task.)</td>
</tr>
<tr>
<td>IsCompleted</td>
<td>Gets whether this Task has completed. (Inherited from Task.)</td>
</tr>
<tr>
<td>IsFaulted</td>
<td>Gets whether the Task completed due to an unhandled exception. (Inherited from Task.)</td>
</tr>
<tr>
<td>Result</td>
<td>Gets the result value of this Task(Of TResult).</td>
</tr>
<tr>
<td>Status</td>
<td>Gets the TaskStatus of this Task. (Inherited from Task.)</td>
</tr>
</tbody>
</table>
## Explicit Interface Implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAsyncResult.AsyncWaitHandle</td>
<td>Gets a WaitHandle that can be used to wait for the task to complete. (Inherited from Task.)</td>
</tr>
<tr>
<td>IAsyncResult.CompletedSynchronously</td>
<td>Gets an indication of whether the asynchronous operation completed synchronously. (Inherited from Task.)</td>
</tr>
</tbody>
</table>
See Also

Task(Of TResult>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Provides access to factory methods for creating `Task(Of TResult>)` instances.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared ReadOnly Property Factory As TaskFactory(Of TResult)

C#

public static TaskFactory<TResult> Factory { get; }
Remarks

The factory returned from Factory is a default instance of TaskFactory<(Of TResult)>), as would result from using the default constructor on the factory type.
See Also

Task<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
Task(Of TResult>).::.Result Property
Task(Of TResult).Class

Gets the result value of this Task(Of TResult).

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Property Result As TResult

C#

public TResult Result { get; internal set; }
Remarks

The get accessor for this property ensures that the asynchronous operation is complete before returning. Once the result of the computation is available, it is stored and will be returned immediately on later calls to Result.
See Also

Task<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Represents an exception used to communicate task cancellation.

**Namespace:**  [System.Threading.Tasks](#)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute> _
Public Class TaskCanceledException _
    Inherits OperationCanceledException

C#

[SerializableAttribute]
public class TaskCanceledException : OperationCanceledException
Inheritance Hierarchy

System..::.Object
System..::.Exception
System..::.SystemException
   System..::.OperationCanceledException
      System.Threading.Tasks..::.TaskCanceledException
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
TaskCanceledException Constructor

TaskCanceledException Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskCanceledException()()</td>
<td>Initializes a new instance of the TaskCanceledException class.</td>
</tr>
<tr>
<td>TaskCanceledException(String)</td>
<td>Initializes a new instance of the TaskCanceledException class with a specified error message.</td>
</tr>
<tr>
<td>TaskCanceledException(Task)</td>
<td>Initializes a new instance of the TaskCanceledException class with a reference to the Task that has been canceled.</td>
</tr>
<tr>
<td>TaskCanceledException(SerializationInfo, StreamingContext)</td>
<td>Initializes a new instance of the TaskCanceledException class with serialized data.</td>
</tr>
<tr>
<td>TaskCanceledException(String, Exception)</td>
<td>Initializes a new instance of the TaskCanceledException class with a specified error message and a reference to the inner exception that is the cause of this exception.</td>
</tr>
</tbody>
</table>
See Also

TaskCanceledException Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCanceledException Constructor

Initializes a new instance of the `TaskCanceledException` class.

**Namespace:**  `System.Threading.Tasks`

**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)
Public Sub New

C#

public TaskCanceledException()
See Also

TaskCanceledException Class
TaskCanceledException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the TaskCanceledException class with a specified error message.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    message As String _
)

C#

public TaskCanceledException(
    string message
)

Parameters

message
    Type: System..::.String
    The error message that explains the reason for the exception.
See Also

TaskCanceledException Class
TaskCanceledException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCanceledException Constructor (Task)

Initializes a new instance of the TaskCanceledException class with a reference to the Task that has been canceled.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    task As Task _
)

C#

public TaskCanceledException(
    Task task
)

Parameters

task
    Type: System.Threading.Tasks.Task
    A task that has been canceled.
See Also

TaskCanceledException Class
TaskCanceledException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCanceledException Constructor (SerializationInfo, StreamingContext)

TaskCanceledException Class  See Also  Send Feedback

Initializes a new instance of the TaskCanceledException class with serialized data.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Sub New (_
    info As SerializationInfo, _
    context As StreamingContext _
)

C#

protected TaskCanceledException(
    SerializationInfo info,
    StreamingContext context
)

Parameters

info
    Type: System.Runtime.Serialization..:::SerializationInfo
    The SerializationInfo that holds the serialized object data about the
    exception being thrown.

context
    Type: System.Runtime.Serialization..:::StreamingContext
    The StreamingContext that contains contextual information about the
    source or destination.
See Also

TaskCanceledException Class
TaskCanceledException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCanceledException Constructor (String, Exception)

Initializes a new instance of the TaskCanceledException class with a specified error message and a reference to the inner exception that is the cause of this exception.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New (_
    message As String, _
    innerException As Exception _
)

C#

public TaskCanceledException(
    string message,
    Exception innerException
)

Parameters

message
    Type: System:::String
    The error message that explains the reason for the exception.

innerException
    Type: System:::Exception
    The exception that is the cause of the current exception.
See Also

TaskCanceledException Class
TaskCanceledException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The **TaskCanceledException** type exposes the following members.
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetBaseException</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetObjectData</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Exception.)</td>
</tr>
</tbody>
</table>
See Also

TaskCanceledException Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `TaskCanceledException` type exposes the following members.
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>HelpLink</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>HResult</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>InnerException</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Message</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Source</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>StackTrace</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>TargetSite</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Task</td>
<td>Gets the task associated with this exception.</td>
</tr>
</tbody>
</table>
See Also

TaskCanceledException Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the task associated with this exception.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Task As Task

C#

public Task Task { get; }
Remarks

It is permissible for no Task to be associated with a `TaskCanceledException`, in which case this property will return null.
See Also

TaskCanceledException Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Represents the producer side of a Task(Of (TResult)>) unbound to a delegate, providing access to the consumer side through the Task property.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization ExternalThreading := True)> 
Public Class TaskCompletionSource(Of TResult)

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization ExternalThreading = true)]
public class TaskCompletionSource<TResult>
Type Parameters

TResult
The type of the result value associated with this TaskCompletionSource<(Of <(TResult)>)).
Remarks

It is often the case that a Task<Of<TResult)> is desired to represent another asynchronous operation. TaskCompletionSource is provided for this purpose. It enables the creation of a task that can be handed out to consumers, and those consumers can use the members of the task as they would any other. However, unlike most tasks, the state of a task created by a TaskCompletionSource is controlled explicitly by the methods on TaskCompletionSource. This enables the completion of the external asynchronous operation to be propagated to the underlying Task. The separation also ensures that consumers are not able to transition the state without access to the corresponding TaskCompletionSource.

All members of TaskCompletionSource<Of<TResult)> are thread-safe and may be used from multiple threads concurrently.
Inheritance Hierarchy

System:::Object
System.Threading.Tasks:::TaskCompletionSource<((TResult))>
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
TaskCompletionSource<(Of<TResult>)> Constructor
TaskCompletionSource<(Of<TResult>)> Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskCompletionSource(&lt;\text{Of }\langle\text{TResult}\rangle&gt;)()()()</td>
<td>Creates a TaskCompletionSource(&lt;\text{Of }\langle\text{TResult}\rangle&gt;).</td>
</tr>
<tr>
<td>TaskCompletionSource(&lt;\text{Of }\langle\text{TResult}\rangle&gt;(\text{Object}))</td>
<td>Creates a TaskCompletionSource(&lt;\text{Of }\langle\text{TResult}\rangle&gt;) with the specified state.</td>
</tr>
<tr>
<td>TaskCompletionSource(&lt;\text{Of }\langle\text{TResult}\rangle&gt;(\text{TaskCreationOptions}))</td>
<td>Creates a TaskCompletionSource(&lt;\text{Of }\langle\text{TResult}\rangle&gt;) with the specified options.</td>
</tr>
<tr>
<td>TaskCompletionSource(&lt;\text{Of }\langle\text{TResult}\rangle&gt;(\text{Object}, \text{TaskCreationOptions}))</td>
<td>Creates a TaskCompletionSource(&lt;\text{Of }\langle\text{TResult}\rangle&gt;) with the specified state and options.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<(Of <(TResult>)>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCompletionSource<(Of <(TResult)>)> Constructor

Creates a TaskCompletionSource<(Of <(TResult)>)>.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public TaskCompletionSource()
See Also

TaskCompletionSource<Of (TResult)> Class
TaskCompletionSource<Of (TResult)> Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCompletionSource<(Of <(TResult)>)> Constructor (Object)

Creates a TaskCompletionSource<(Of <(TResult)>)> with the specified state.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
        state As Object _
    )

C#

public TaskCompletionSource(
        Object state
    )

Parameters

state
    Type: System..::.Object
    The state to use as the underlying Task<(<TResult>)>'s AsyncState.
See Also

TaskCompletionSource(Of<TResult>) Class
TaskCompletionSource(Of<TResult>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCompletionSource<(Of '<(TResult)>)> Constructor (TaskCreationOptions)

Creates a TaskCompletionSource<(Of '<(TResult)>)> with the specified options.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    creationOptions As TaskCreationOptions _
)

C#

public TaskCompletionSource( _
    TaskCreationOptions creationOptions
)

Parameters

creationOptions
    Type: System.Threading.Tasks.TaskCreationOptions
    The options to use when creating the underlying Task(Of TResult).
Remarks

The Task<(Of <(TResult)>)> created by this instance and accessible through its Task property will be instantiated using the specified creationOptions.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The creationOptions represent options invalid for use with a TaskCompletionSource(&lt;{\text{TResult}}&gt;).</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<Of<TResult>> Class
TaskCompletionSource<Of<TResult>> Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a `TaskCompletionSource(Of TResult)` with the specified state and options.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Sub New ( _
    state As Object, _
    creationOptions As TaskCreationOptions _
)
```

**C#**

```csharp
public TaskCompletionSource(
    Object state,
    TaskCreationOptions creationOptions
)
```

### Parameters

**state**

- **Type:** `System::Object`
- The state to use as the underlying `Task(Of TResult)`)`'s AsyncState.

**creationOptions**

- **Type:** `System.Threading.Tasks::TaskCreationOptions`
- The options to use when creating the underlying `Task(Of TResult)`)`. 
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System::ArgumentOutOfRangeException</code></td>
<td>The <code>creationOptions</code> represent options invalid for use with a <code>TaskCompletionSource&lt;</code>(\text{TResult})&gt;).</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource(Of TResult>) Class
TaskCompletionSource(Of TResult>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

TaskCompletionSource<TResult> Methods

The TaskCompletionSource<TResult> type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>SetCanceled</strong></td>
<td>Transitions the underlying Task(&lt;\text{Of }\langle\text{TResult}\rangle&gt;) into the <strong>Canceled</strong> state.</td>
</tr>
<tr>
<td><strong>SetException</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>SetResult</strong></td>
<td>Transitions the underlying Task(&lt;\text{Of }\langle\text{TResult}\rangle&gt;) into the <strong>RanToCompletion</strong> state.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>TrySetCanceled</strong></td>
<td>Attempts to transition the underlying Task(&lt;\text{Of }\langle\text{TResult}\rangle&gt;) into the <strong>Canceled</strong> state.</td>
</tr>
<tr>
<td><strong>TrySetException</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>TrySetResult</strong></td>
<td>Attempts to transition the underlying Task(&lt;\text{Of }\langle\text{TResult}\rangle&gt;) into the <strong>RanToCompletion</strong> state.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<Of <(TResult)>>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Transitions the underlying Task<Of (TResult)> into the Canceled state.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub SetCanceled

C#

public void SetCanceled()
<table>
<thead>
<tr>
<th><strong>Exception</strong></th>
<th><strong>Condition</strong></th>
</tr>
</thead>
</table>
| System:::InvalidOperationException                   | The underlying Task<*(Of *(TResult]*)>* is already in one of the three final states:  
|                                                   | RanToCompletion, Faulted, or Canceled.                                           |
| System:::ObjectDisposedException                  | The Task was disposed.                                                           |
See Also

TaskCompletionSource<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCompletionSource(Of TResult)::.SetException Method

TaskCompletionSource(Of TResult) Class
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SetException(IEnumerable&lt;Of Exception&gt;)</code></td>
<td>Transitions the underlying Task&lt;(Of TResult&gt;) into the <strong>Faulted</strong> state.</td>
</tr>
<tr>
<td><code>SetException(Exception)</code></td>
<td>Transitions the underlying Task&lt;(Of TResult&gt;) into the <strong>Faulted</strong> state.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Transitions the underlying Task(Of TResult) into the Faulted state.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub SetException ( _
    exceptions As IEnumerable(Of Exception) _
)

C#

public void SetException ( _
    IEnumerable<Exception> exceptions _
)

Parameters

exceptions
    Type: System.Collections.Generic:::IEnumerable<(Of <(Exception)>)>  
The collection of exceptions to bind to this Task<(Of <(TResult)>)>.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exceptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>There are one or more null elements in exceptions.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The underlying Task&lt;@(Of &lt;(TResult)&gt;)&gt; is already in one of the three final states:</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The Task was disposed.</td>
</tr>
</tbody>
</table>


See Also

TaskCompletionSource<(Of <(TResult)>)> Class
SetException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Transitions the underlying Task\(<\text{Of }<(\text{TResult}>)>\) Class into the \text{Faulted} state.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub SetException ( _ 
    exception As Exception _
)

C#

public void SetException( 
    Exception exception 
)

Parameters

exception
    Type: System.GetType.Exception
    The exception to bind to this Task<TResult>.>
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception argument is null.</td>
</tr>
<tr>
<td><code>InvalidOperationException</code></td>
<td>The underlying <code>Task&lt;Of &lt;(TResult)&gt;&gt;</code> is already in one of the three final states: <code>RanToCompletion</code>, <code>Faulted</code>, or <code>Canceled</code>.</td>
</tr>
<tr>
<td><code>ObjectDisposedException</code></td>
<td>The <code>Task</code> was disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<(Of <(TResult)>>) Class
SetException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Transitions the underlying Task<TResult> into the RanToCompletion state.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub SetResult (  
    result As TResult  
)

C#

public void SetResult(  
    TResult result  
)

Parameters

result
  Type: TResult  
The result value to bind to this Task<TResult>.  

### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::InvalidOperationException</td>
<td>The underlying Task&lt;(&lt;TResult&gt;)&gt; is already in one of the three final states: RanToCompletion, Faulted, or Canceled.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task was disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<Of (TResult)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Attempts to transition the underlying Task<(Of (TResult)>) into the Canceled state.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TrySetCanceled As Boolean

C#

public bool TrySetCanceled()
Remarks

This operation will return false if the Task<(Of <(TResult)>)> is already in one of the three final states: RanToCompletion, Faulted, or Canceled.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The Task was disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<(Of <(TResult<>)>)) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
TaskCompletionSource<(Of <(TResult)>)>::TrySetException Method
TaskCompletionSource<(Of <(TResult)>)> Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TrySetException(IEnumerable&lt;Of (Exception)&gt;)</code></td>
<td>Attempts to transition the underlying <code>Task&lt;Of TResult&gt;</code> into the <strong>Faulted</strong> state.</td>
</tr>
<tr>
<td><code>TrySetException(Exception)</code></td>
<td>Attempts to transition the underlying <code>Task&lt;Of TResult&gt;</code> into the <strong>Faulted</strong> state.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<Of TResult>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Attempts to transition the underlying Task<
(Of <(TResult)>)> into the Faulted state.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TrySetException ( _
    exceptions As IEnumerable(Of Exception) _
) As Boolean

C#

public bool TrySetException(
    IEnumerable<Exception> exceptions
)

Parameters

exceptions
    Type: System.Collections.Generic.IEnumerable<Exception>
    The collection of exceptions to bind to this Task<TResult>.

Return Value

True if the operation was successful; otherwise, false.
Remarks

This operation will return false if the Task<((Of <(TResult)>))> is already in one of the three final states: RanToCompletion, Faulted, or Canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exceptions argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>There are one or more null elements in exceptions.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exceptions collection is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The <a href="#">Task</a> was disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<(Of <(TResult)>)> Class
TrySetException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Attempts to transition the underlying Task<itect(TResult)>
into the Faulted state.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TrySetException ( _
   exception As Exception _
) As Boolean

C#

public bool TrySetException(
   Exception exception
)

Parameters

exception
   Type: System...:::Exception
   The exception to bind to this Task<Of <TResult>>.

Return Value

True if the operation was successful; otherwise, false.
Remarks

This operation will return false if the Task<(Of <(TResult)>)> is already in one of the three final states: RanToCompletion, Faulted, or Canceled.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The Task was disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<(Of <(TResult)>)> Class
TrySetException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskCompletionSource(Of TResult)::.TrySetResult Method

Attempts to transition the underlying Task(Of TResult) into the RanToCompletion state.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function TrySetResult ( _
    result As TResult _
) As Boolean

C#

public bool TrySetResult(
    TResult result
)

Parameters

result
    Type: TResult
    The result value to bind to this Task<(Of <(TResult)>)>.

Return Value

True if the operation was successful; otherwise, false.
Remarks

This operation will return false if the Task<(Of <(TResult)>)> is already in one of the three final states: RanToCompletion, Faulted, or Canceled.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The Task was disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<(Of <(TResult)>)) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `TaskCompletionSource(Of TResult)` type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Gets the Task&lt;(Of &lt;(TResult)&gt;)&gt; created by this TaskCompletionSource&lt;(Of &lt;(TResult)&gt;)&gt;.</td>
</tr>
</tbody>
</table>
See Also

TaskCompletionSource<(Of (TResult>)) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#

.NET Framework Class Library

TaskCompletionSource<(Of <(TResult)>)>.::.Task Property

TaskCompletionSource<(Of <(TResult)>)> Class  See Also  Send Feedback

Gets the Task<(Of <(TResult)>)> created by this TaskCompletionSource<(Of <(TResult)>)>.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Task As Task(Of TResult)

C#

public Task<TResult> Task { get; }
Remarks

This property enables a consumer access to the Task<(Of <(TResult)>)> that is controlled by this instance. The SetResult(TResult), SetException(Exception), SetException(IEnumerable<Of <(Exception)>>), and SetCanceled() methods (and their "Try" variants) on this instance all result in the relevant state transitions on this underlying Task.
See Also

TaskCompletionSource<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Specifies flags that control optional behavior for the creation and execution of continuation tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute> _
<FlagsAttribute> _
Public Enumeration TaskContinuationOptions

C#

[SerializableAttribute]
[FlagsAttribute]
public enum TaskContinuationOptions
## Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Continuations, by default, will be scheduled when the antecedent task completes, regardless of the task's final <a href="#">TaskStatus</a>. A hint to a <a href="#">TaskScheduler</a> to schedule a task in as fair a manner as possible, meaning that tasks scheduled sooner will be more likely to be run sooner, and tasks scheduled later will be more likely to be run later. Specifies that a task will be a long-running, course-grained operation. It provides a hint to the <a href="#">TaskScheduler</a> that oversubscription may be warranted.</td>
</tr>
<tr>
<td>PreferFairness</td>
<td>Specifies that a task is attached to a parent in the task hierarchy. Specifies that the continuation task should not be scheduled if its antecedent ran to completion. This option is not valid for multi-task continuations. Specifies that the continuation task should not be scheduled if its antecedent threw an unhandled exception. This option is not valid for multi-task continuations. Specifies that the continuation task should not be scheduled if its antecedent was canceled. This option is not valid for multi-task continuations. Specifies that the continuation task should be scheduled only if its antecedent ran to completion. This option is not valid for</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>OnlyOnFaulted</td>
<td>Specifies that the continuation task should be scheduled only if its antecedent threw an unhandled exception. This option is not valid for multi-task continuations.</td>
</tr>
<tr>
<td>OnlyOnCanceled</td>
<td>Specifies that the continuation task should be scheduled only if its antecedent was canceled. This option is not valid for multi-task continuations.</td>
</tr>
<tr>
<td>ExecuteSynchronously</td>
<td>Specifies that the continuation task should be executed synchronously. With this option specified, the continuation will be run on the same thread that causes the antecedent task to transition into its final state. If the antecedent is already complete when the continuation is created, the continuation will run on the thread creating the continuation. Only very short-running continuations should be executed synchronously.</td>
</tr>
</tbody>
</table>
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Specifies flags that control optional behavior for the creation and execution of tasks.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute> _
<FlagsAttribute> _
Public Enumeration TaskCreationOptions

C#

[SerializableAttribute]
[FlagsAttribute]
public enum TaskCreationOptions
### Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Specifies that the default behavior should be used. A hint to a TaskScheduler to schedule a task in as fair a manner as possible, meaning that tasks scheduled sooner will be more likely to be run sooner, and tasks scheduled later will be more likely to be run later.</td>
</tr>
<tr>
<td>PreferFairness</td>
<td>Specifies that a task will be a long-running, coarse-grained operation. It provides a hint to the TaskScheduler that oversubscription may be warranted.</td>
</tr>
<tr>
<td>LongRunning</td>
<td>Specifies that a task is attached to a parent in the task hierarchy.</td>
</tr>
<tr>
<td>AttachedToParent</td>
<td></td>
</tr>
</tbody>
</table>
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Provides a set of static (Shared in Visual Basic) methods for working with specific kinds of `Task` instances.

**Namespace:**  `System.Threading.Tasks`  
**Assembly:**  `System.Threading` (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public NotInheritable Class TaskExtensions

C#

public static class TaskExtensions
Inheritance Hierarchy

System...:::Object
System.Threading.Tasks...:::TaskExtensions
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
TaskExtensions Methods

TaskExtensions Class  See Also  Send Feedback
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s Unwrap</td>
<td>Overloaded.</td>
</tr>
</tbody>
</table>
See Also

TaskExtensions Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
- Include Protected Members
- Include Inherited Members
.NET Framework Class Library
TaskExtensions:::Unwrap Method

TaskExtensions Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Unwrap&lt;(Of &lt;(TResult&gt;&gt;)&gt;</code></td>
<td>Creates a proxy <code>Task{TResult}</code> that represents the asynchronous operation of a <code>Task{Task{TResult}}</code>.</td>
</tr>
<tr>
<td><code>Unwrap(Task&lt;(Of (Task&lt;(Of &lt;(TResult&gt;&gt;)&gt;)&gt;))</code></td>
<td>Creates a proxy <code>Task</code> that represents the asynchronous operation of a <code>Task{Task}</code>.</td>
</tr>
</tbody>
</table>
See Also

TaskExtensions Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskExtensions..::.Unwrap(Of (TResult)) Method (Task(Of (Task(Of (TResult))))))

TaskExtensions Class  See Also  Send Feedback

Creates a proxy Task{TResult} that represents the asynchronous operation of a Task{Task{TResult}}.
Unwraps a Task that returns another Task.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Unwrap(Of TResult) ( _
    task As Task(Of Task(Of TResult))) _
) As Task(Of TResult)

C#

public static Task<TResult> Unwrap<TResult>(
    Task<TTask<TResult>> task
)

Parameters

task
    Type: System.Threading.Tasks.<>c__DisplayClass1_0<TResult>
    The Task{Task{TResult}} to unwrap.
Type Parameters

TResult

Return Value

A Task{TResult} that represents the asynchronous operation of the provided Task{Task{TResult}}.
Remarks

It is often useful to be able to return a Task\{TResult\} from a Task\{TResult\}, where the inner Task\{TResult\} represents work done as part of the outer Task\{TResult\}. However, doing so results in a Task\{Task\{TResult\}\}, which, if not dealt with carefully, could produce unexpected behavior. Unwrap solves this problem by creating a proxy Task\{TResult\} that represents the entire asynchronous operation of such a Task\{Task\{TResult\}\}.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown if the task argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskExtensions Class
Unwrap Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskExtensions:::Unwrap Method (Task<(Of <(Task)>))
TaskExtensions Class  See Also  Send Feedback

Creates a proxy Task that represents the asynchronous operation of a Task{Task}.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Function Unwrap ( _
    task As Task(Of Task) _) As Task

C#

public static Task Unwrap(
    Task<Task> task
)

Parameters

task
    Type: System.Threading.Tasks.Task(Of Task)<(Task)>
    The Task{Task} to unwrap.

Return Value

A Task that represents the asynchronous operation of the provided Task{Task}.
Remarks

It is often useful to be able to return a Task from a Task{TResult}, where the inner Task represents work done as part of the outer Task{TResult}. However, doing so results in a Task{Task}, which, if not dealt with carefully, could produce unexpected behavior. Unwrap solves this problem by creating a proxy Task that represents the entire asynchronous operation of such a Task{Task}. 
<table>
<thead>
<tr>
<th><strong>Exception</strong></th>
<th><strong>Condition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown if the task argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskExtensions Class
Unwrap Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Provides support for creating and scheduling Tasks.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading := True)> _
Public Class TaskFactory

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
   ExternalThreading = true)]
public class TaskFactory
Remarks

There are many common patterns for which tasks are relevant. The TaskFactory class encodes some of these patterns into methods that pick up default settings, which are configurable through its constructors.

A default instance of TaskFactory is available through the Task.Factory property.
Inheritance Hierarchy

System..:::Object
System.Threading.Tasks..:::TaskFactory
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
TaskFactory Constructor

TaskFactory Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskFactory()()()</td>
<td>Initializes a <a href="#">TaskFactory</a> instance with the default configuration.</td>
</tr>
<tr>
<td>TaskFactory(CancellationToken)</td>
<td>Initializes a <a href="#">TaskFactory</a> instance with the specified configuration.</td>
</tr>
<tr>
<td>TaskFactory(TaskScheduler)</td>
<td>Initializes a <a href="#">TaskFactory</a> instance with the specified configuration.</td>
</tr>
<tr>
<td>TaskFactory(TaskCreationOptions, TaskContinuationOptions)</td>
<td>Initializes a <a href="#">TaskFactory</a> instance with the specified configuration.</td>
</tr>
<tr>
<td>TaskFactory(CancellationToken, TaskCreationOptions, TaskContinuationOptions, TaskScheduler)</td>
<td>Initializes a <a href="#">TaskFactory</a> instance with the specified configuration.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a TaskFactory instance with the default configuration.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public TaskFactory()
Remarks

This constructor creates a TaskFactory instance with a default configuration. The TaskCreationOptions property is initialized to TaskCreationOptions.None, the TaskContinuationOptions property is initialized to TaskContinuationOptions.None, and the TaskScheduler property is initialized to the current scheduler (see TaskScheduler.Current).
See Also

TaskFactory Class
TaskFactory Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a TaskFactory instance with the specified configuration.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
cancellationToken As CancellationToken _
)

C#

public TaskFactory(
    CancellationToken cancellationToken
)

Parameters

cancellationToken

Type: System.Threading...:::CancellationToken

The default CancellationToken that will be assigned to tasks created by this TaskFactory unless another CancellationToken is explicitly specified while calling the factory methods.
Remarks

This constructor creates a TaskFactory instance with a default configuration. The TaskCreationOptions property is initialized to TaskCreationOptions.None, the TaskContinuationOptions property is initialized to TaskContinuationOptions.None, and the TaskScheduler property is initialized to the current scheduler (see TaskScheduler.Current).
See Also

TaskFactory Class
TaskFactory Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory Constructor (TaskScheduler)

Initializes a TaskFactory instance with the specified configuration.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    scheduler As TaskScheduler _
) _

C#

public TaskFactory(
    TaskScheduler scheduler
)

Parameters

scheduler

Type: System.Threading.Tasks.TaskScheduler

The TaskScheduler to use to schedule any tasks created with this TaskFactory. A null value indicates that the current TaskScheduler should be used.
Remarks

With this constructor, the `TaskCreationOptions` property is initialized to `TaskCreationOptions.None`, the `TaskContinuationOptions` property is initialized to `TaskContinuationOptions.None`, and the `TaskScheduler` property is initialized to scheduler, unless it’s null, in which case the property is initialized to the current scheduler (see `TaskScheduler.Current`).
See Also

TaskFactory Class
TaskFactory Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory Constructor (TaskCreationOptions, TaskContinuationOptions)

TaskFactory Class   See Also   Send Feedback

Initializes a TaskFactory instance with the specified configuration.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    creationOptions As TaskCreationOptions, _
    continuationOptions As TaskContinuationOptions _
)

C#

public TaskFactory(
    TaskCreationOptions creationOptions,
    TaskContinuationOptions continuationOptions
)

Parameters

creationOptions
    Type: System.Threading.Tasks::{:TaskCreationOptions
    The default TaskCreationOptions to use when creating tasks with this TaskFactory.

continuationOptions
    Type: System.Threading.Tasks::{:TaskContinuationOptions
    The default TaskContinuationOptions to use when creating continuation tasks with this TaskFactory.
Remarks

With this constructor, the `TaskCreationOptions` property is initialized to `creationOptions`, the `TaskContinuationOptions` property is initialized to `continuationOptions`, and the `TaskScheduler` property is initialized to the current scheduler (see `TaskScheduler.Current`).
Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::..ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument or the</td>
</tr>
<tr>
<td></td>
<td>continuationOptions argument specifies an invalid value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
TaskFactory Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a TaskFactory instance with the specified configuration.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Sub New ( _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
)
```

### C#

```csharp
public TaskFactory(  
    CancellationToken cancellationToken,  
    TaskCreationOptions creationOptions,  
    TaskContinuationOptions continuationOptions,  
    TaskScheduler scheduler
)
```

## Parameters

**cancellationToken**
- **Type:** `System.Threading.CancellationToken`
  - The default `CancellationToken` that will be assigned to tasks created by this `TaskFactory` unless another CancellationToken is explicitly specified while calling the factory methods.

**creationOptions**
- **Type:** `System.Threading.Tasks.TaskCreationOptions`
  - The default `TaskCreationOptions` to use when creating tasks with this `TaskFactory`.

**continuationOptions**
- **Type:** `System.Threading.Tasks.TaskContinuationOptions`
  - The default `TaskContinuationOptions` to use when creating continuation tasks with this `TaskFactory`.
**scheduler**

Type: [System.Threading.Tasks.TaskScheduler](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskscheduler)

The default [TaskScheduler](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskscheduler) to use to schedule any Tasks created with this TaskFactory. A null value indicates that TaskScheduler.Current should be used.
Remarks

With this constructor, the `TaskCreationOptions` property is initialized to `creationOptions`, the `TaskContinuationOptions` property is initialized to `continuationOptions`, and the `TaskScheduler` property is initialized to `scheduler`, unless it's null, in which case the property is initialized to the current scheduler (see `TaskScheduler.Current`).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..:..ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument or the continuationOptions argument specifies an invalid value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
TaskFactory Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `TaskFactory` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ContinueWhenAll</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>ContinueWhenAny</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>Equals</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>Finalize</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>FromAsync</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>GetHashCode</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>GetType</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>MemberwiseClone</strong></td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td><strong>StartNew</strong></td>
<td>Overloaded.</td>
</tr>
<tr>
<td><strong>ToString</strong></td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
System.Threading.Tasks Namespace
Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
TaskFactory:::ContinueWhenAll Method
TaskFactory Class  See Also  Send Feedback
### Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ContinueWhenAll(array&lt;Task&gt;[][], Action&lt;Of (array&lt;Task&gt;[][])&gt;)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll&lt;Of&lt; TResult&gt;&gt; (array&lt;Task&gt;[][], Func&lt;Of&lt; (array&lt;Task&gt;[][])&gt;, TResult&gt;)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll&lt;Of&lt; (TAntecedentResult)&gt;&gt;(array&lt;Task&lt;Of&lt; (TAntecedentResult)&gt;&gt;[][], Action&lt;Of&lt; (array&lt;Task&lt;Of&lt; (TAntecedentResult)&gt;&gt;[][])&gt;&gt;, TResult&gt;)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll&lt;Of&lt; (TAntecedentResult, TResult)&gt;(array&lt;Task&lt;Of&lt; (TAntecedentResult, TResult)&gt;&gt;&gt;[], Func&lt;Of&lt; (array&lt;Task&lt;Of&lt; (TAntecedentResult, TResult)&gt;&gt;&gt;[], TResult)&gt;)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll(array&lt;Task&gt;[][], Action&lt;Of (array&lt;Task&gt;[][])&gt;, CancellationToken)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
</tbody>
</table>
ContinueWhenAll(array<Task>[][], Action(Of (array<Task>[][])[], TaskContinuationOptions)) that will be started upon the completion of a set of provided Tasks.

ContinueWhenAll(Of (TResult)[]) (array<Task>[][], Func(Of (array<Task>[][])[], TResult[])[], CancellationToken) Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

ContinueWhenAll(Of (TResult)[]) (array<Task>[][], Func(Of (array<Task>[][])[], TResult[])[], TaskContinuationOptions) Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

ContinueWhenAll(Of (TAntecedentResult)[]) (array<Task<(Of (TAntecedentResult)[])[]>[][], Action(Of (array<Task<(Of (TAntecedentResult)[])[]>[][])[], CancellationToken)) Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

ContinueWhenAll(Of (TAntecedentResult)[]) (array<Task<(Of (TAntecedentResult)[])[]>[][], Action(Of (array<Task<(Of (TAntecedentResult)[])[]>[][])[], TaskContinuationOptions)) Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

ContinueWhenAll(Of (TAntecedentResult, TResult)[]) (array<Task<(Of (TAntecedentResult, TResult)[])[]>[][], Func(Of (array<Task<(Of (TAntecedentResult, TResult)[])[]>[][])[], CancellationToken)) Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

ContinueWhenAll(Of (TAntecedentResult, TResult)[]) (array<Task<(Of (TAntecedentResult, TResult)[])[]>[][], Func(Of (array<Task<(Of (TAntecedentResult, TResult)[])[]>[][])[], TaskContinuationOptions)) Creates a continuation Task that will be started upon the completion of a set of provided Tasks.
ContinueWhenAll(array<Task>[][], Func<Of (array<Task>(Of (TAntecedentResult>))[[]], TResult>), TaskContinuationOptions, TaskScheduler)

ContinueWhenAll<Of (TResult)>>(array<Task>[][], Func<Of (array<Task>[][])>, CancellationToken, TaskContinuationOptions, TaskScheduler)

ContinueWhenAll<Of (TAntecedentResult)>(array<Task>(Of (TAntecedentResult>))[[]], Action<Of (array<Task>(Of (TAntecedentResult>))[[]], TResult>), CancellationToken, TaskContinuationOptions, TaskScheduler)

ContinueWhenAll<Of (TAntecedentResult, TResult)>(array<Task>(Of (TAntecedentResult>))[[]], Func<Of (array<Task>(Of (TAntecedentResult>))[[]], TResult>), CancellationToken, TaskContinuationOptions, TaskScheduler)

upon the completion of a set of provided Tasks.

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWhenAll ( _
    tasks As Task(), _
    continuationAction As Action(0f Task()) _) As Task
```

**C#**

```csharp
public Task ContinueWhenAll(
    Task[] tasks,
    Action<Task[]> continuationAction
)
```

### Parameters

**tasks**
- The array of tasks from which to continue.

**continuationAction**
- Type: System.Action<(Of <(System.Threading.Tasks.Task [])>)>
- The action delegate to execute when all tasks in the tasks array have completed.

### Return Value

The new continuation Task.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::ContinueWhenAll(Of <(TResult)> ) Method (array<Task>[] [], Func(Of <(array<Task>[][])[], TResult>))

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace:  System.Threading.Tasks  
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TResult) ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task(), TResult) _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll<TResult>(
    Task[] tasks,
    Func<Task[], TResult> continuationFunction
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task >[]
    The array of tasks from which to continue.

continuationFunction
    Type: System.Func< Of (array< Task >[][], TResult)>)
    The function delegate to execute when all tasks in the tasks array have completed.
Type Parameters

TResult
   The type of the result that is returned by the continuationFunction delegate and associated with the created Task<(Of <(TResult)>)>.

Return Value

The new continuation Task<(Of <(TResult)>)>.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td><code>ArgumentException</code></td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td><code>ArgumentException</code></td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td><code>ObjectDisposedException</code></td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskFactory.::.ContinueWhenAll(Of (TAntecedentResult)>) Method
(array<Task<Of (TAntecedentResult)>>[], Action<Of
(array<Task<Of (TAntecedentResult)>>)[]()>)
TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of
provided Tasks.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationAction As Action(Of Task(Of TAntecedentResult)())
) As Task

C#

public Task ContinueWhenAll<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Action<Task<TAntecedentResult>[] > continuationAction
)

Parameters

tasks
Type: array< System.Threading.Tasks:::Task ><(Of <(TAntecedentResult)>)>[]0[]
The array of tasks from which to continue.

continuationAction
Type: System:::Action<(Of <(array< Task ><(Of <(TAntecedentResult)>)>)>0[])>)
The action delegate to execute when all tasks in the tasks array have completed.
Type Parameters

TAntecedentResult  
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::ContinueWhenAll<(Of <(TAntecedentResult, TResult)>)> Method (array<Task<(Of <(TAntecedentResult)>)>>[], Func<(Of <(array<Task<(Of <(TAntecedentResult)>)>>[])[], TResult>>)  

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWhenAll(Of TAntecedentResult, TResult)( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult)();
) As Task(Of TResult)
```

**C#**

```csharp
public TResult ContinueWhenAll<TAntecedentResult, TResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>[][], TResult> continuationFunction
)
```

**Parameters**

tasks
Type: array< System.Threading.Tasks.Task<>(Of <(TAntecedentResult)>[]) >[]
The array of tasks from which to continue.

continuationFunction
Type: System.Func<Of <(array< Task<>(Of <(TAntecedentResult)>[]) >[]) , TResult>
The function delegate to execute when all tasks in the tasks array have completed.
**Type Parameters**

TAntecedentResult  
The type of the result of the antecedent tasks.

TResult  
The type of the result that is returned by the continuationFunction delegate and associated with the created Task<Of <(TResult)>).

**Return Value**

The new continuation Task<Of <(TResult)>).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAll ( _
    tasks As Task(), _
    continuationAction As Action(Of Task()), _
    cancellationToken As CancellationToken _) As Task
```

### C#

```csharp
public Task ContinueWhenAll(
    Task[] tasks,
    Action<Task[]> continuationAction,
    CancellationToken cancellationToken
)
```

## Parameters

- **tasks**
  - Type: array< System.Threading.Tasks.Task >[]
  - The array of tasks from which to continue.

- **continuationAction**
  - Type: System..::.Action<(Of <(array< Task >[])[]>>)
  - The action delegate to execute when all tasks in the tasks array have completed.

- **cancellationToken**
  - Type: System.Threading..::.CancellationToken
  - The CancellationToken that will be assigned to the new continuation task.

## Return Value

The new continuation Task.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System::.ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::ContinueWhenAll Method (array<Task>[][], Action<Of <(array<Task>[][])>>, TaskContinuationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace:  System Threading Tasks
Assembly:  System Threading (in System Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll ( _
    tasks As Task(), _
    continuationAction As Action(Of Task()), _
    continuationOptions As TaskContinuationOptions _
) As Task

C#

public Task ContinueWhenAll(
    Task[] tasks,
    Action<Task[]> continuationAction,
    TaskContinuationOptions continuationOptions
)

Parameters

tasks
    Type: array<System.Threading.Tasks.Task>[]
    The array of tasks from which to continue.

continuationAction
    Type: System..:::Action(Of <(array< Task >[])[]>)
    The action delegate to execute when all tasks in the tasks array have completed.

continuationOptions
    Type: System..:::TaskContinuationOptions
    The TaskContinuationOptions value that controls the behavior of the created continuation Task.

Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory:::ContinueWhenAll(Of TResult>) Method

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
# Syntax

## Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAll(Of TResult) ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task(), TResult), _
    cancellationToken As CancellationToken _
) As Task(Of TResult)
```

## C#

```csharp
public Task<TResult> ContinueWhenAll<TResult>(
    Task[] tasks,
    Func<Task[], TResult> continuationFunction,
    CancellationToken cancellationToken
)
```

## Parameters

- **tasks**
  - **Type:** array< System.Threading.Tasks.Task >[]
  - The array of tasks from which to continue.

- **continuationFunction**
  - **Type:** System::Func<Of <array< Task >[][], TResult>>
  - The function delegate to execute when all tasks in the tasks array have completed.

- **cancellationToken**
  - **Type:** System.Threading.CancellationToken
  - The CancellationToken that will be assigned to the new continuation task.
Type Parameters

TResult
The type of the result that is returned by the continuationFunction delegate and associated with the created Task<(Of <(TResult)>)>.

Return Value
The new continuation Task<(Of <(TResult)>)>.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System...::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System...::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System...::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System...::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory:::ContinueWhenAll<Of <(TResult)>>() Method (array<Task>[][], Func<Of <(array<Task>[][])[], TResult>>), TaskContinuationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

**Namespace:**  System.Threading.Tasks
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TResult) ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task(), TResult), _
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll<TResult>(
    Task[] tasks,
    Func<Task[], TResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)

Parameters

tasks
Type: array< System.Threading.Tasks.Task >[]()[]
The array of tasks from which to continue.

continuationFunction
Type: System.Func(Of (array< Task >[]()[], TResult)>)
The function delegate to execute when all tasks in the tasks array have completed.

continuationOptions
Type: System.Threading.Tasks.TaskContinuationOptions
The TaskContinuationOptions value that controls the behavior of the created continuation Task.
Type Parameters

TResult
The type of the result that is returned by the continuationFunction delegate and associated with the created Task<(Of <(TResult)>)>.

Return Value

The new continuation Task<(Of <(TResult)>)>.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory.:::ContinueWhenAll(Of <(TAntecedentResult)>>) Method
(array<Task<Of <(TAntecedentResult)>>>[], Action<Of
<(array<Task<Of <(TAntecedentResult)>>>>[], CancellationToken)

Creates a continuation Task that will be started upon the completion of a set of
provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

Public Function ContinueWhenAll(Of TAntecedentResult) (  
    tasks As Task(Of TAntecedentResult)(),  
    continuationAction As Action(Of Task(Of TAntecedentResult)()),  
    cancellationToken As CancellationToken  
) As Task

**C#**

public Task ContinueWhenAll<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Action<Task<TAntecedentResult>[]> continuationAction,
    CancellationToken cancellationToken
)

**Parameters**

- **tasks**
  Type: array< System.Threading.Tasks:::Task <(Of <(TAntecedentResult)>)[]> >[]
  The array of tasks from which to continue.

- **continuationAction**
  Type: System:::Action<(Of <(array< Task<(Of <(TAntecedentResult)>)>[])[]>)>
  The action delegate to execute when all tasks in the tasks array have completed.

- **cancellationToken**
  Type: System.Threading:::CancellationToken
  The CancellationToken that will be assigned to the new continuation task.
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskFactory..::.ContinueWhenAll<(Of <(TAntecedentResult)>)> Method
(array<Task<(Of <(TAntecedentResult)>)>[])[], Action<(Of
(array<Task<(Of <(TAntecedentResult)>)>[])[])[]),
TaskContinuationOptions)
TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of
provided Tasks.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationAction As Action(Of Task(Of TAntecedentResult)()),
    continuationOptions As TaskContinuationOptions _
) As Task

C#

public Task ContinueWhenAll<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Action<Task<TAntecedentResult>[] > continuationAction,
    TaskContinuationOptions continuationOptions
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task<(Of
        (Of TAntecedentResult)>[]) >[]0[]
    The array of tasks from which to continue.

continuationAction
    Type: System:::Action<(Of <(array< Task<(Of (Of TAntecedentResult)>)> >[0][]>)>
    The action delegate to execute when all tasks in the tasks array have completed.

continuationOptions
    Type: System.Threading.Tasks:::TaskContinuationOptions
    The TaskContinuationOptions value that controls the behavior of the created continuation Task.
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory:::ContinueWhenAll<Of <(TAntecedentResult, TResult)>>() Method (array<Task<Of <(TAntecedentResult)>>>>[][], Func<Of <(array<Task<Of <(TAntecedentResult)>>>>[][], TResult)>>, CancellationToken)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAll(Of TAntecedentResult, TResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult)(), _
    cancellationToken As CancellationToken _
) As Task(Of TResult)
```

#### C#

```csharp
public Task<TResult> ContinueWhenAll<TAntecedentResult, TResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>[][], TResult> continuationFunction,
    CancellationToken cancellationToken)
```

### Parameters

**tasks**

- **Type:** `array< System.Threading.Tasks:::Task<Of <(TAntecedentResult)>>()>[][]`
- The array of tasks from which to continue.

**continuationFunction**

- **Type:** `System:::Func<Of <(array< Task<Of <(TAntecedentResult)>>()>[][])>, TResult>`)`
- The function delegate to execute when all tasks in the tasks array have completed.

**cancellationToken**

- **Type:** `System.Threading:::CancellationToken`
- The `CancellationToken` that will be assigned to the new continuation task.
**Type Parameters**

TAntecedentResult
   The type of the result of the antecedent tasks.

TResult
   The type of the result that is returned by the continuationFunction delegate and associated with the created Task<(Of <(TResult)>)>.

**Return Value**

The new continuation Task<(Of <(TResult)>)>.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.Exception.ArgumentNullException</code></td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td><code>System.Exception.ArgumentNullException</code></td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td><code>System.Exception.ArgumentNullException</code></td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td><code>System.Exception.ArgumentNullException</code></td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td><code>System.Exception.ObjectDisposedException</code></td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td><code>System.Exception.ObjectDisposedException</code></td>
<td>The provided <a href="https://docs.microsoft.com/en-us/dotnet/api/system.cancellationtoken">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory.....ContinueWhenAll<(Of <(TAntecedentResult, TResult)>)> Method (array<Task<(Of <(TAntecedentResult)>)>[]>[], Func<(Of <(array<Task<(Of <(TAntecedentResult)>)>[])[]>[], TResult>), TaskContinuationOptions)

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult, TResult)( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult)();
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll<TAntecedentResult, TResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>[][], TResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task<(Of
        (Of TAntecedentResult>)[]) >[0][]
The array of tasks from which to continue.

continuationFunction
    Type: System:::Func<(Of array< Task<(Of (Of TAntecedentResult>)>) >
        [[][], TResult>)>
The function delegate to execute when all tasks in the tasks array have completed.

continuationOptions
    Type: System.Threading.Tasks:::TaskContinuationOptions
    The TaskContinuationOptions value that controls the behavior of the created continuation Task.
Type Parameters

TAntecedentResult
  The type of the result of the antecedent tasks.
TResult
  The type of the result that is returned by the continuationFunction delegate and associated with the created Task<Of (TResult)>.

Return Value

The new continuation Task<Of (TResult)>.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic ▪ C#
.NET Framework Class Library
TaskFactory....:ContinueWhenAll Method (array<Task>[][], Action<Of <(array<Task>[][])[]>>, CancellationToken, TaskContinuationOptions, TaskScheduler)
TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWhenAll ( _
    tasks As Task(), _
    continuationAction As Action(Of Task()), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task
```

**C#**

```csharp
public Task ContinueWhenAll(
    Task[] tasks,
    Action<Task[]> continuationAction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)
```

**Parameters**

- **tasks**
  Type: array< System.Threading.Tasks:::Task >[]([])
  The array of tasks from which to continue.

- **continuationAction**
  Type: System...:::Action<(Of <(array< Task >[]([])>)>)
  The action delegate to execute when all tasks in the tasks array have completed.

- **cancellationToken**
  Type: System.Threading...:::CancellationToken
  The CancellationToken that will be assigned to the new continuation task.

- **continuationOptions**
  Type: System.Threading.Tasks...:::TaskContinuationOptions
The `TaskContinuationOptions` value that controls the behavior of the created continuation `Task`.

scheduler
    Type: `System.Threading.Tasks.TaskScheduler`
The `TaskScheduler` that is used to schedule the created continuation `Task`.

**Return Value**

The new continuation `Task`. 
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
.NET Framework Class Library
TaskFactory....::ContinueWhenAll<Of <(TResult)>>) Method (array<Task>[](),
[], Func<Of <(array<Task>[][])[], TResult>>), CancellationToken,
TaskContinuationOptions, TaskScheduler)
TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of
provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAll(Of TResult) ( _
    tasks As Task(), _,
    continuationFunction As Func(Of Task(), TResult), _,
    cancellationToken As CancellationToken, _,
    continuationOptions As TaskContinuationOptions, _,
    scheduler As TaskScheduler _
) As Task(Of TResult)
```

#### C#

```csharp
public Task<TResult> ContinueWhenAll<TResult>(
    Task[] tasks,
    Func<Task[], TResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)
```

### Parameters

**tasks**
- Type: array of System.Threading.Tasks:: Task
- The array of tasks from which to continue.

**continuationFunction**
- Type: System:: Func<Of <(array of Task>[][], TResult)>>
- The function delegate to execute when all tasks in the tasks array have completed.

**cancellationToken**
- Type: System.Threading:: CancellationToken
- The CancellationToken that will be assigned to the new continuation task.

**continuationOptions**
- Type: System.Threading.Tasks:: TaskContinuationOptions
The `TaskContinuationOptions` value that controls the behavior of the created continuation Task.

scheduler
Type: `System.Threading.Tasks.TaskScheduler`

The `TaskScheduler` that is used to schedule the created continuation Task<Of<TResult>>.
Type Parameters

TResult
The type of the result that is returned by the continuationFunction delegate and associated with the created Task<(Of <(TResult)>)>.

ReturnValue
The new continuation Task<(Of <(TResult)>)>.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::ContinueWhenAll<(Of <(TAntecedentResult)>)> Method (array<Task<(Of <(TAntecedentResult)>)>[], Action<(Of array<Task<(Of <(TAntecedentResult)>)>[])[][], CancellationToken, TaskContinuationOptions, TaskScheduler)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationAction As Action(Of Task(Of TAntecedentResult)(), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _) As Task

C#

public Task ContinueWhenAll<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Action<Task<TAntecedentResult>[]> continuationAction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task<Of
        (Of TAntecedentResult)> []> []
The array of tasks from which to continue.

continuationAction
    Type: System....Action<Of <array< Task<Of
        (Of TAntecedentResult)>> []>
    The action delegate to execute when all tasks in the tasks array have completed.

cancellationToken
    Type: System.Threading.CancellationToken
    The CancellationToken that will be assigned to the new continuation task.
continuationOptions
Type: `System.Threading.Tasks::TaskContinuationOptions`
The `TaskContinuationOptions` value that controls the behavior of the created continuation `Task`.

scheduler
Type: `System.Threading.Tasks::TaskScheduler`
The `TaskScheduler` that is used to schedule the created continuation `Task`.
Type Parameters

TAntecedentResult
The type of the result of the antecedent tasks.

Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic ▪ C#

.NET Framework Class Library

TaskFactory....:ContinueWhenAll<(Of <(TAntecedentResult, TResult>)>)
Method (array<Task<(Of <(TAntecedentResult)>)>[][], Func<(Of (array<Task<(Of <(TAntecedentResult)>)>[][])[], TResult)[]), CancellationToken, TaskContinuationOptions, TaskScheduler)

TaskFactory Class ▪ See Also ▪ Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult, TResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult)(), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll<TAntecedentResult, TResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>[], TResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

tasks
    Type: array System.Threading.Tasks:::Task<Of ($(TAntecedentResult)>)[[]]
    The array of tasks from which to continue.

continuationFunction
    Type: System:::Func<Of ($(array< Task<Of ($(TAntecedentResult)>)> >)[[]]
    The function delegate to execute when all tasks in the tasks array have completed.

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.
continuationOptions
Type: System.Threading.Tasks.TaskContinuationOptions
The TaskContinuationOptions value that controls the behavior of the created continuation Task.

scheduler
Type: System.Threading.Tasks.TaskScheduler
The TaskScheduler that is used to schedule the created continuation Task<Of <(TResult)>).
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.
TResult
   The type of the result that is returned by the continuationFunction delegate and associated with the created Task<Of <(TResult)>).

Return Value

The new continuation Task<Of <(TResult)>).
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#  Include Protected Members  Include Inherited Members
.NET Framework Class Library  TaskFactory....:ContinueWhenAny Method
TaskFactory Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ContinueWhenAny(array&lt;Task&gt;[][], Action&lt;Of&lt;(Task)&gt;&gt;&gt;)</code></td>
<td>Creates a continuation <code>Task</code> that will be started upon the completion of any <code>Task</code> in the provided set.</td>
</tr>
<tr>
<td><code>ContinueWhenAny&lt;Of&lt;(TResult)&gt;&gt;&gt;(array&lt;Task&gt;[][], Func&lt;Of&lt;(Task, TResult)&gt;&gt;)</code></td>
<td>Creates a continuation Task that will be started upon the completion of any <code>Task</code> in the provided set.</td>
</tr>
<tr>
<td><code>ContinueWhenAny&lt;Of&lt;(TAntecedentResult)&gt;&gt;(array&lt;Task&gt;&lt;(Of&lt;(TAntecedentResult)&gt;[])[], Action&lt;Of&lt;(Task&lt;Of&lt;(TAntecedentResult)&gt;[])&gt;&gt;&gt;)</code></td>
<td>Creates a continuation <code>Task</code> that will be started upon the completion of any <code>Task</code> in the provided set.</td>
</tr>
<tr>
<td><code>ContinueWhenAny&lt;Of&lt;(TAntecedentResult, TResult)&gt;&gt;&gt;(array&lt;Task&gt;&lt;(Of&lt;(TAntecedentResult)&gt;[])[], Func&lt;Of&lt;(Task&lt;Of&lt;(TAntecedentResult)&gt;&gt;, TResult)&gt;&gt;)</code></td>
<td>Creates a continuation <code>Task</code> that will be started upon the completion of any <code>Task</code> in the provided set.</td>
</tr>
<tr>
<td><code>ContinueWhenAny(array&lt;Task&gt;[][], Action&lt;Of&lt;(Task)&gt;&gt;, CancellationToken)</code></td>
<td>Creates a continuation <code>Task</code> that will be started upon the completion of any <code>Task</code> in the provided set.</td>
</tr>
</tbody>
</table>
ContinueWhenAny(array<Task>[][], Action<Of <(Task)>>, TaskContinuationOptions)

ContinueWhenAny<Of <(TResult)>>(array<Task>[][], Func<Of <(Task, TResult)>>, CancellationToken)

ContinueWhenAny<Of <(TResult)>>(array<Task>[][], Func<Of <(Task, TResult)>>, TaskContinuationOptions)

ContinueWhenAny<Of <(TAntecedentResult)>>(array<Task><(Of <(TAntecedentResult)>)>[][], Action<Of <(Task<Of <(TAntecedentResult)>>)>>>), CancellationToken)

ContinueWhenAny<Of <(TAntecedentResult)>>(array<Task><(Of <(TAntecedentResult)>)>[][], Action<Of <(Task<Of <(TAntecedentResult)>>)>>>), TaskContinuationOptions)

ContinueWhenAny<Of <(TAntecedentResult, TResult)>>(array<Task><(Of <(TAntecedentResult, TResult)>)>[][], Func<Of <(Task<Of <(TAntecedentResult)>>)>>>), CancellationToken)

ContinueWhenAny<Of <(TAntecedentResult, TResult)>>(array<Task><(Of <(TAntecedentResult, TResult)>)>[][], Func<Of <(Task<Of <(TAntecedentResult)>>)>>>), CancellationToken)
ContinueWhenAny(array<Task>[][], Func<Of (Task<Of <(TAntecedentResult)>>)>, CancellationToken, TaskContinuationOptions, TaskScheduler)

ContinueWhenAny(Of <(TResult)>>) (array<Task>[][], Func<Of (Task, TResult)>, CancellationToken, TaskContinuationOptions, TaskScheduler)

ContinueWhenAny<Of <(TAntecedentResult)>)> (array<Task<Of <(TAntecedentResult)>)>[][], Action<Of (Task<Of <(TAntecedentResult)>)>>), CancellationToken, TaskContinuationOptions, TaskScheduler)

ContinueWhenAny<Of <(TAntecedentResult, TResult)>)> (array<Task<Of <(TAntecedentResult, TResult)>)>[][], Func<Of (Task<Of <(TAntecedentResult)>)>, TResult)>, CancellationToken, TaskContinuationOptions, TaskScheduler)

upon the completion of any Task in the provided set.

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Creates a continuation Task that will be started upon the completion of any Task in the provided set.
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....ContinueWhenAny Method (array<Task>[][], Action<Of <(Task)>>)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny ( _
    tasks As Task(), _
    continuationAction As Action(Of Task) _
) As Task

C#

public Task ContinueWhenAny(
    Task[] tasks,
    Action<Task> continuationAction
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task >[]
    The array of tasks from which to continue when one task completes.

continuationAction
    Type: System...Action<(Of <Task>)>
    The action delegate to execute when one task in the tasks array completes.

Return Value

The new continuation Task.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class  
ContinueWhenAny Overload  
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TResult) ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task, TResult) _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny<TResult>(
    Task[] tasks,
    Func<Task, TResult> continuationFunction
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task>[]
The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System.Func(Of (Task, TResult))
The function delegate to execute when one task in the tasks array completes.
Type Parameters

TResult
The type of the result that is returned by the continuationFunction delegate and associated with the created Task(Of TResult).

Return Value

The new continuation Task(Of TResult).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::ContinueWhenAny<(Of <(TAntecedentResult)>)> Method (array<Task<(Of <(TAntecedentResult)>)>[])[], Action<(Of <(Task<(Of <(TAntecedentResult)>)>)>))>

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationAction As Action(Of Task(Of TAntecedentResult))
) As Task

C#

public Task ContinueWhenAny<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Action<Task<TAntecedentResult>> continuationAction
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task<((TAntecedentResult)>) >[]
    The array of tasks from which to continue when one task completes.

continuationAction
    Type: System:::Action<((Task<((TAntecedentResult)>)>)>
    The action delegate to execute when one task in the tasks array completes.
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAny(Of TAntecedentResult, TResult) ( _
tasks As Task(Of TAntecedentResult)(), _
continuationFunction As Func(Of Task(Of TAntecedentResult), _
) As Task(Of TResult)
```

### C#

```csharp
public TResult ContinueWhenAny<TAntecedentResult, TResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>, TResult> continuationFunction
)
```

### Parameters

**tasks**

- **Type:** array< `System.Threading.Tasks.Task<`TAntecedentResult>`)[]>
- The array of tasks from which to continue when one task completes.

**continuationFunction**

- **Type:** `System.Func<`Task<`TAntecedentResult>)$, TResult>`)`
- The function delegate to execute when one task in the tasks array completes.
Type Parameters

TAntecedentResult
The type of the result of the antecedent tasks.

TResult
The type of the result that is returned by the continuationFunction delegate and associated with the created Task<Of <(TResult)>>.

Return Value

The new continuation Task<Of <(TResult)>>.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny ( _
    tasks As Task(), _
    continuationAction As Action(Of Task), _
    cancellationToken As CancellationToken _
) As Task

C#

public Task ContinueWhenAny(  
    Task[] tasks,
    Action<Task> continuationAction,
    CancellationToken cancellationToken
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task >[]
    The array of tasks from which to continue when one task completes.

tympinationAction
    Type: System..::.Action(Of <(Task)>)
    The action delegate to execute when one task in the tasks array completes.

cancellationToken
    Type: System.Threading..::.CancellationToken
    The CancellationToken that will be assigned to the new continuation task.

Return Value

The new continuation Task.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::ContinueWhenAny Method (array<Task>[][], Action<Of (Task[])>, TaskContinuationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAny ( _
    tasks As Task(), _
    continuationAction As Action(Of Task), _
    continuationOptions As TaskContinuationOptions _
) As Task
```

#### C#

```csharp
public Task ContinueWhenAny(
    Task[] tasks,
    Action<Task> continuationAction,
    TaskContinuationOptions continuationOptions
)
```

### Parameters

- **tasks**
  - Type: array<object><System.Threading.Tasks.Task>[]
  - The array of tasks from which to continue when one task completes.

- **continuationAction**
  - Type: System..::.Action<Object, (Of System..::.Task)>)
  - The action delegate to execute when one task in the tasks array completes.

- **continuationOptions**
  - Type: System.Threading.Tasks..::.TaskContinuationOptions
  - The TaskContinuationOptions value that controls the behavior of the created continuation Task.

### Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid <code>TaskContinuationOptions</code> value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....::ContinueWhenAny<(Of <(TResult)>)> Method (array<Task>[] [], Func<(Of <(Task, TResult)>), CancellationToken)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:**  System.Threading.Tasks

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TResult) ( _
  tasks As Task(), _
  continuationFunction As Func(Of Task, TResult), _
  cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny<TResult>(
  Task[] tasks,
  Func<Task, TResult> continuationFunction,
  CancellationToken cancellationToken
)

Parameters

tasks
  Type: array< System.Threading.Tasks.Task >[]
  The array of tasks from which to continue when one task completes.

continuationFunction
  Type: System.Func(Of (Task, TResult))
  The function delegate to execute when one task in the tasks array completes.

cancellationToken
  Type: System.Threading.CancellationToken
  The CancellationToken that will be assigned to the new continuation task.
**Type Parameters**

**TResult**

The type of the result that is returned by the `continuationFunction` delegate and associated with the created `Task<Of <(TResult)<>)>`.

**Return Value**

The new continuation `Task<Of <(TResult)<>)>`. 
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskFactory...::ContinueWhenAny(Of (TResult)>) Method (array<Task>[] [], Func(Of (Task, TResult)>, TaskContinuationOptions)
TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWhenAny(Of TResult)(
    _
    tasks As Task(), _
    continuationFunction As Func(Of Task, TResult), _
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)
```

**C#**

```csharp
public Task<TResult> ContinueWhenAny<TResult>(
    Task[] tasks,
    Func<Task, TResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)
```

**Parameters**

**tasks**

Type: array< System.Threading.Tasks:::Task >[]

The array of tasks from which to continue when one task completes.

**continuationFunction**

Type: System:::Func<Of (Task, TResult)>)

The function delegate to execute when one task in the tasks array completes.

**continuationOptions**

Type: System.Threading.Tasks:::TaskContinuationOptions

The TaskContinuationOptions value that controls the behavior of the created continuation Task.
**Type Parameters**

**TResult**

The type of the result that is returned by the `continuationFunction` delegate and associated with the created `Task<Of<TResult>>`.

**Return Value**

The new continuation `Task<Of<TResult>>`).
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::ContinueWhenAny<(<TAntecedentResult>)> Method
(array<Task<(<TAntecedentResult>)>[]>[], Action<(<Task<(<TAntecedentResult>)>)>), CancellationToken)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```
Public Function ContinueWhenAny(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationAction As Action(Of Task(Of TAntecedentResult)),
    cancellationToken As CancellationToken _
) As Task
```

**C#**

```
public Task ContinueWhenAny<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Action<Task<TAntecedentResult>> continuationAction,
    CancellationToken cancellationToken
)
```

**Parameters**

- **tasks**
  Type: array< `System.Threading.Tasks.Task`<Of `TAntecedentResult`>>][[]
  The array of tasks from which to continue when one task completes.

- **continuationAction**
  Type: `System.Func<Task<Of `(Task<Of `TAntecedentResult`>)>>>
  The action delegate to execute when one task in the tasks array completes.

- **cancellationToken**
  Type: `System.Threading.CancellationToken`
  The `CancellationToken` that will be assigned to the new continuation task.
**Type Parameters**

TAntecedentResult
   The type of the result of the antecedent tasks.

**Return Value**

The new continuation Task.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....:ContinueWhenAny<(Of <(TAntecedentResult)>)> Method
(array<Task<(Of <(TAntecedentResult)>)>[][]), Action<(Of <(Task<(Of
<(TAntecedentResult)>)>>)>), TaskContinuationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task
in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function ContinueWhenAny(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationAction As Action(Of Task(Of TAntecedentResult)),
    continuationOptions As TaskContinuationOptions _) As Task
```

**C#**

```csharp
public Task ContinueWhenAny<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Action<Task<TAntecedentResult>> continuationAction,
    TaskContinuationOptions continuationOptions
)
```

### Parameters

**tasks**
Type: array< System.Threading.Tasks:::Task<Of <(TAntecedentResult)> >>[]
The array of tasks from which to continue when one task completes.

**continuationAction**
Type: System:::Action<Of <(Task<Of <(TAntecedentResult)>)>>)
The action delegate to execute when one task in the tasks array completes.

**continuationOptions**
Type: System.Threading.Tasks:::TaskContinuationOptions
The TaskContinuationOptions value that controls the behavior of the created continuation Task.
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Create a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TAntecedentResult, TResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult),
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny<TAntecedentResult, TResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>, TResult> continuationFunction,
    CancellationToken cancellationToken
)

Parameters

tasks
    Type: array< System.Threading.Tasks...:::Task<(Of TAntecedentResult)>> >[]]
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System...:::Func<(Of (Task<(Of (TAntecedentResult)>)),
    TResult>)>
    The function delegate to execute when one task in the tasks array
    completes.

cancellationToken
    Type: System.Threading...:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.
Type Parameters

TAntecedentResult
The type of the result of the antecedent tasks.

TResult
The type of the result that is returned by the continuationFunction delegate and associated with the created Task<Of<TResult>>.

ReturnValue

The new continuation Task<Of<TResult>>.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....::ContinueWhenAny<(Of <(TAntecedentResult, TResult)>)>
Method (array<TTask<(Of <(TAntecedentResult)>)>[]>[], Func<(Of <(Task<(Of <(TAntecedentResult)>)>, TResult)>)>, TaskContinuationOptions)

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TAntecedentResult, TResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult),
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny<TAntecedentResult, TResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>, TResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)

Parameters

tasks
    Type: array< System.Threading.Tasks::*:Task>(Of <(TAntecedentResult)>[])[]>
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System::.Func<Of <(Task<(Of <(TAntecedentResult)>)>),
    TResult>>
    The function delegate to execute when one task in the tasks array
    completes.

continuationOptions
    Type: System.Threading.Tasks::*:TaskContinuationOptions
    The TaskContinuationOptions value that controls the behavior of the
    created continuation Task.
**Type Parameters**

TAntecedentResult
   The type of the result of the antecedent tasks.

TResult
   The type of the result that is returned by the continuationFunction delegate and associated with the created Task<((TResult)>)

**Return Value**

The new continuation Task<((TResult)>)
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System...:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System...:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...:::ContinueWhenAny Method (array<Task>[][], Action<(Of <(Task)>)>, CancellationToken, TaskContinuationOptions, TaskScheduler)

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny ( _
    tasks As Task(), _
    continuationAction As Action(Of Task), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task

C#

public Task ContinueWhenAny(  
    Task[] tasks,
    Action<Task> continuationAction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task >[]()
    The array of tasks from which to continue when one task completes.

continuationAction
    Type: System...:..Action<(Of (Task)>)
    The action delegate to execute when one task in the tasks array completes.

cancellationToken
    Type: System.Threading.CancellationToken
    The CancellationToken that will be assigned to the new continuation task.

continuationOptions
    Type: System.Threading.Tasks.TaskContinuationOptions
    The TaskContinuationOptions value that controls the behavior of the
created continuation Task.

scheduler
Type: System.Threading.Tasks.TaskScheduler
The TaskScheduler that is used to schedule the created continuation Task.

Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null. The exception that is thrown when the continuationAction argument is null. The exception that is thrown when the scheduler argument is null. The exception that is thrown when the tasks array contains a null value. The exception that is thrown when the tasks array is empty. The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value. The exception that is thrown when one of the elements in the tasks array has been disposed. The provided CancellationToken has already been disposed.</td>
</tr>
<tr>
<td>System...:::.ArgumentOutOfRangeException</td>
<td></td>
</tr>
<tr>
<td>System...:::.ObjectDisposedException</td>
<td></td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
.NET Framework Class Library
TaskFactory...::ContinueWhenAny<Of <(TResult>>>) Method (array<Task>[] [], Func<Of <(Task, TResult)>>), CancellationToken, TaskContinuationOptions, TaskScheduler)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TResult) ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task, TResult), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny<TResult>((
    Task[] tasks,
    Func<Task, TResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

tasks
    Type: array<System.Threading.Tasks:::Task>[][]
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System:::Func(Of (Task, TResult)>)
    The function delegate to execute when one task in the tasks array completes.

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.

continuationOptions
    Type: System.Threading.Tasks:::TaskContinuationOptions
The **TaskContinuationOptions** value that controls the behavior of the created continuation Task.

**scheduler**

Type: **System.Threading.Tasks::TaskScheduler**

The **TaskScheduler** that is used to schedule the created continuation Task<Of <(TResult)>>.
Type Parameters

TResult
The type of the result that is returned by the continuationFunction delegate and associated with the created Task<Of <(TResult)>).

Return Value

The new continuation Task<Of <(TResult)>).
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory.::.ContinueWhenAny(Of <(TAntecedentResult)>) Method
(array<Task<Of <(TAntecedentResult)>>>[], Action<Of <(Task<Of <(TAntecedentResult)>>)>>), CancellationToken, TaskContinuationOptions, TaskScheduler)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAny(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationAction As Action(Of Task(Of TAntecedentResult)), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _) As Task
```

#### C#

```csharp
public Task ContinueWhenAny<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Action<TTask<TAntecedentResult>> continuationAction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)
```

### Parameters

**tasks**
- Type: array< System.Threading.Tasks..:::Task<Of <(TAntecedentResult)>>() >]0[[]]
- The array of tasks from which to continue when one task completes.

**continuationAction**
- Type: System...:::Action<Of <(Task<Of <(TAntecedentResult)>)>>>()
- The action delegate to execute when one task in the tasks array completes.

**cancellationToken**
- Type: System.Threading..:::CancellationToken
- The CancellationToken that will be assigned to the new continuation task.

**continuationOptions**
- Type: System.Threading.Tasks..:::TaskContinuationOptions
The `TaskContinuationOptions` value that controls the behavior of the created continuation `Task`.

scheduler

Type: `System.Threading.Tasks.TaskScheduler`

The `TaskScheduler` that is used to schedule the created continuation `Task<Of <(TResult)>).`
Type Parameters

TAntecedentResult

The type of the result of the antecedent tasks.

Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationAction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...:::ContinueWhenAny<(Of <(TAntecedentResult, TResult)>)> Method (array<Task<(Of <(TAntecedentResult)>)>>[][], Func<(Of <(Task<(Of <(TAntecedentResult)>), TResult)>), CancellationToken, TaskContinuationOptions, TaskScheduler)

TaskFactory Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TAntecedentResult, TResult) (_
  tasks As Task(Of TAntecedentResult)(), _
  continuationFunction As Func(Of Task(Of TAntecedentResult),
  cancellationToken As CancellationToken, _
  continuationOptions As TaskContinuationOptions, _
  scheduler As TaskScheduler _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny<TAntecedentResult, TResult>(
  Task<TAntecedentResult>[] tasks,
  Func<Task<TAntecedentResult>, TResult> continuationFunction,
  CancellationToken cancellationToken,
  TaskContinuationOptions continuationOptions,
  TaskScheduler scheduler
)

Parameters

tasks
  Type: array< System.Threading.Tasks.Task<Of
  (Of TAntecedentResult)> >[]
  The array of tasks from which to continue when one task completes.

continuationFunction
  Type: System.Func<Of (Of TAntecedentResult)>>
  The function delegate to execute when one task in the tasks array
  completes.

cancellationToken
  Type: System.Threading.CancellationToken
  The CancellationToken that will be assigned to the new continuation task.
continuationOptions
Type: System.Threading.Tasks::TaskContinuationOptions
The TaskContinuationOptions value that controls the behavior of the created continuation Task.

scheduler
Type: System.Threading.Tasks::TaskScheduler
The TaskScheduler that is used to schedule the created continuation Task<Of <(TResult)>).
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

TResult
   The type of the result that is returned by the continuationFunction delegate and associated with the created Task<Of <(TResult)>).

Return Value

The new continuation Task<Of <(TResult)>>.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System...:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System...:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System...:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...FromAsync Method

TaskFactory Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromAsync(IAsyncResult, Action&lt;Of &lt;(IAsyncResult)&gt;))</td>
<td>Creates a Task that executes an end method action when a specified IAsyncResult completes.</td>
</tr>
<tr>
<td>FromAsync&lt;Of &lt;(TResult)&gt;) (IAsyncResult, Func&lt;Of &lt;(IAsyncResult, TResult)&gt;&gt;)</td>
<td>Creates a Task that executes an end method function when a specified IAsyncResult completes.</td>
</tr>
<tr>
<td>FromAsync&lt;Of &lt;(TResult)&gt;) (Func&lt;Of &lt;(AsyncCallback, Object, IAsyncResult)&gt;, Action&lt;Of &lt;(IAsyncResult)&gt;, Object&gt;)</td>
<td>Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.</td>
</tr>
<tr>
<td>FromAsync&lt;Of &lt;(TResult)&gt;&gt;)(Func&lt;Of &lt;(AsyncCallback, Object, IAsyncResult)&gt;&gt;, Func&lt;Of &lt;(IAsyncResult, TResult)&gt;&gt;, Object)</td>
<td>Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.</td>
</tr>
<tr>
<td>FromAsync(IAsyncResult, Action&lt;Of &lt;(IAsyncResult)&gt;&gt;, TaskCreationOptions)</td>
<td>Creates a Task that executes an end method action when a specified IAsyncResult completes.</td>
</tr>
<tr>
<td>FromAsync&lt;Of &lt;(TResult)&gt;&gt;)(IAsyncResult, Func&lt;Of &lt;(IAsyncResult, TResult)&gt;&gt;, TaskCreationOptions)</td>
<td>Creates a Task that executes an end method function when a specified IAsyncResult completes.</td>
</tr>
</tbody>
</table>
FromAsync(Func<Of <(AsyncCallback, Object, IAsyncResult)>>, Action<Of <(IAsyncResult)>>, Object, TaskCreationOptions)

FromAsync<Of <(TResult)>>(Func<Of <(AsyncCallback, Object, IAsyncResult)>>, Func<Of <(IAsyncResult, TResult)>>, Object, TaskCreationOptions)

FromAsync<Of <(TArg1)>>(Func<Of <(TArg1, AsyncCallback, Object, IAsyncResult)>>, Action<Of <(IAsyncResult)>>, TArg1, Object)

FromAsync<Of <(TArg1, TResult)>)(Func<Of <(TArg1, AsyncCallback, Object, IAsyncResult)>>, Func<Of <(IAsyncResult, TResult)>>, TArg1, Object)

FromAsync(IAsyncResult, Action<Of <(IAsyncResult)>>, TaskCreationOptions, TaskScheduler)

FromAsync<Of <(TResult)>>(IAsyncResult, Func<Of <(IAsyncResult, TResult)>>, TaskCreationOptions, TaskScheduler)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Creates a Task that executes an end method action when a specified IAsyncResult completes.

Creates a Task that executes an end method function when a specified IAsyncResult completes.
FromAsync<(Of <(TArg1)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, Object, TaskCreationOptions)

FromAsync<(Of <(TArg1, TResult)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), TArg1, Object, TaskCreationOptions)

FromAsync<(Of <(TArg1, TArg2)>))(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, TArg2, Object)

FromAsync<(Of <(TArg1, TArg2, TResult)>))(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), TArg1, TArg2, Object)

FromAsync<(Of <(TArg1, TArg2, TArg3)>))(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), Action<(Of <(IAsyncResult)>)), TArg1, TArg2, TArg3, Object)

FromAsync<(Of <(TArg1, TArg2, TArg3, TResult)>))(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), TaskCreationOptions)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync<(Of <(TArg1, TArg2)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, Object, TaskCreationOptions)

FromAsync<(Of <(TArg1, TArg2, TResult)>))(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), TArg1, TArg2, Object)

FromAsync<(Of <(TArg1, TArg2, TArg3)>))(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), Action<(Of <(IAsyncResult)>)), TArg1, TArg2, TArg3, Object)

FromAsync<(Of <(TArg1, TArg2)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, Object, TaskCreationOptions)

FromAsync<(Of <(TArg1, TArg2, TResult)>))(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), TArg1, TArg2, Object)

FromAsync<(Of <(TArg1, TArg2, TArg3)>))(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), TaskCreationOptions)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync<(Of <(TArg1, TArg2)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, Object, TaskCreationOptions)

FromAsync<(Of <(TArg1, TArg2, TResult)>))(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), TArg1, TArg2, Object)

FromAsync<(Of <(TArg1, TArg2, TArg3)>))(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), TaskCreationOptions)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync<(Of <(TArg1, TArg2)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, Object, TaskCreationOptions)

FromAsync<(Of <(TArg1, TArg2, TResult)>))(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), TArg1, TArg2, Object)

FromAsync<(Of <(TArg1, TArg2, TArg3)>))(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), TaskCreationOptions)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync<(Of <(TArg1, TArg2)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, Object, TaskCreationOptions)

FromAsync<(Of <(TArg1, TArg2, TResult)>))(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), TArg1, TArg2, Object)

FromAsync<(Of <(TArg1, TArg2, TArg3)>))(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), TaskCreationOptions)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync<(Of <(TArg1, TArg2)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, Object, TaskCreationOptions)

FromAsync<(Of <(TArg1, TArg2, TResult)>))(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), TArg1, TArg2, Object)

FromAsync<(Of <(TArg1, TArg2, TArg3)>))(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), TaskCreationOptions)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync<(Of <(TArg1, TArg2)>))(Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Action<(Of <(IAsyncResult)>)), TArg1, Object, TaskCreationOptions)
conform to the Asynchronous Programming Model pattern.

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::FromAsync Method (IAsyncResult, Action(Of IAsyncResult)>)

TaskFactory Class  See Also  Send Feedback

Creates a Task that executes an end method action when a specified IAsyncResult completes.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync ( _
    asyncResult As IAsyncResult, _
    endMethod As Action(Of IAsyncResult) _
) As Task

C#

public Task FromAsync(
    IAsyncResult asyncResult,
    Action<IAsyncResult> endMethod
)

Parameters

asyncResult
    Type: System...:::IAsyncResult
    The IAsyncResult whose completion should trigger the processing of the endMethod.

demendMethod
    Type: System...:::Action(Of (Of IAsyncResult>)>
    The action delegate that processes the completed asyncResult.

Return Value

A Task that represents the asynchronous operation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::: FromAsync<Of <(TResult)>>() Method (IAsyncResult, Func<Of <(IAsyncResult, TResult)>>())

TaskFactory Class  See Also  Send Feedback

Creates a Task that executes an end method function when a specified IAsyncResult completes.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function FromAsync(Of TResult) (_
    asyncResult As IAsyncResult, _
    endMethod As Func(Of IAsyncResult, TResult) _) _
) As Task(Of TResult)
```

### C#

```csharp
public Task<TResult> FromAsync<TResult>(
    IAsyncResult asyncResult,
    Func<IAsyncResult, TResult> endMethod
)
```

## Parameters

**asyncResult**
- Type: System:::IAsyncResult
- The IAsyncResult whose completion should trigger the processing of the `endMethod`.

**endMethod**
- Type: System:::Func(Of <(IAsyncResult, TResult)>)
- The function delegate that processes the completed asyncResult.
Type Parameters

TResult
    The type of the result available through the Task.

Return Value

A Task that represents the asynchronous operation.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class  
FromAsync Overload  
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::FromAsync Method (Func<(Of <(AsyncCallback, Object, IAsyncResult)>)), Action<(Of <(IAsyncResult)>)), Object)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync ( _
    beginMethod As Func(Of AsyncCallback, Object, IAsyncResult),
    endMethod As Action(Of IAsyncResult), _
    state As Object _
) As Task

C#

public Task FromAsync(
    Func<AsyncCallback, Object, IAsyncResult> beginMethod,
    Action<IAsyncResult> endMethod,
    Object state
)

Parameters

beginMethod
    Type: System..::.Func<(Of <(AsyncCallback, Object, IAsyncResult)>)>)
    The delegate that begins the asynchronous operation.

endMethod
    Type: System..::.Action<(Of <IAsyncResult>)>)
    The delegate that ends the asynchronous operation.

state
    Type: System..::.Object
    An object containing data to be used by the beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...:::FromAsync(Of <(TResult)>()) Method (Func(Of (AsyncCallback, Object, IAsyncResult)<>), Func(Of <(IAsyncResult, TResult)>()), Object)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TResult) ( _
    beginMethod As Func(Of AsyncCallback, Object, IAsyncResult),
    endMethod As Func(Of IAsyncResult, TResult), _
    state As Object _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TResult>(
    Func<AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    Object state
)

Parameters

beginMethod
    Type: System...::Func<(Of <(AsyncCallback, Object, IAsyncResult)>)>)
    The delegate that begins the asynchronous operation.

dendMethod
    Type: System...::Func<(Of <(IAsyncResult, TResult)>)>)
    The delegate that ends the asynchronous operation.

state
    Type: System...::Object
    An object containing data to be used by the beginMethod delegate.
### Type Parameters

**TResult**

The type of the result available through the Task.

### Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System...::ArgumentException</code></td>
<td>The exception that is thrown when the <code>beginMethod</code> argument is null.</td>
</tr>
<tr>
<td><code>System...::ArgumentException</code></td>
<td>The exception that is thrown when the <code>endMethod</code> argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::FromAsync Method (IAsyncResult, Action<Of (IAsyncResult>>), TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that executes an end method action when a specified IAsyncResult completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

```vbnet
Public Function FromAsync (_
    asyncResult As IAsyncResult, _
    endMethod As Action(Of IAsyncResult), _
    creationOptions As TaskCreationOptions _
) As Task
```

C#

```csharp
public Task FromAsync(
    IAsyncResult asyncResult,
    Action<IAsyncResult> endMethod,
    TaskCreationOptions creationOptions
)
```

Parameters

asyncResult
Type: System.IAsyncResult
The IAsyncResult whose completion should trigger the processing of the endMethod.

endMethod
Type: System.Action(Of (Of IAsyncResult))>
The action delegate that processes the completed asyncResult.

creationOptions
Type: System.Threading.Tasks.TaskCreationOptions
The TaskCreationOptions value that controls the behavior of the created Task.

Return Value

A Task that represents the asynchronous operation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::FromAsync(Of (TResult)>) Method (IAasyncResult, Func(Of ((IAasyncResult, TResult)>)>, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that executes an end method function when a specified IAsyncResult completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
# Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function FromAsync(Of TResult) ( _
    asyncResult As IAsyncResult, _
    endMethod As Func(Of IAsyncResult, TResult), _
    creationOptions As TaskCreationOptions _) _
) As Task(Of TResult)
```

**C#**

```csharp
public Task<TResult> FromAsync<TResult>(
    IAsyncResult asyncResult, 
    Func<IAsyncResult, TResult> endMethod,
    TaskCreationOptions creationOptions
)
```

## Parameters

**asyncResult**

- **Type:** System:::IAsyncResult
- The IAsyncResult whose completion should trigger the processing of the endMethod.

**endMethod**

- **Type:** System:::Func(Of (Of IAsyncResult, TResult)>)
- The function delegate that processes the completed asyncResult.

**creationOptions**

- **Type:** System.Threading.Tasks:::TaskCreationOptions
- The TaskCreationOptions value that controls the behavior of the created Task.
Type Parameters

TResult
    The type of the result available through the Task.

Return Value

A Task that represents the asynchronous operation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....::FromAsync Method (Func<(Of <(AsyncCallback, Object, IAsyncResult)>)>, Action<(Of <(IAsyncResult)>)>, Object, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function FromAsync ( _
    beginMethod As Func(Of AsyncCallback, Object, IAsyncResult),
    endMethod As Action(Of IAsyncResult), _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task
```

### C#

```csharp
public Task FromAsync(
    Func<AsyncCallback, Object, IAsyncResult> beginMethod,
    Action<IAsyncResult> endMethod,
    Object state,
    TaskCreationOptions creationOptions
)
```

## Parameters

**beginMethod**
- Type: `System ::= Func<(Of (AsyncCallback, Object, IAsyncResult)>)`
- The delegate that begins the asynchronous operation.

**endMethod**
- Type: `System ::= Action<(Of (IAsyncResult)>)`
- The delegate that ends the asynchronous operation.

**state**
- Type: `System ::= Object`
- An object containing data to be used by the beginMethod delegate.

**creationOptions**
- Type: `System.Threading.Tasks ::= TaskCreationOptions`
- The TaskCreationOptions value that controls the behavior of the created `Task`. 
Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory......:FromAsync<(Of <(TResult)>)> Method (Func<(Of <(AsyncCallback, Object, IAsyncResult)>)), Func<(Of <(IAsyncResult, TResult)>)), Object, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function FromAsync(Of TResult) ( _
    beginMethod As Func(Of AsyncCallback, Object, IAsyncResult), _
    endMethod As Func(Of IAsyncResult, TResult), _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)
```

### C#

```csharp
public Task<TResult> FromAsync<TResult>(
    Func<AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    Object state,
    TaskCreationOptions creationOptions
)
```

## Parameters

**beginMethod**
Type: `System..::.Func(Of (AsyncCallback, Object, IAsyncResult)>)`
The delegate that begins the asynchronous operation.

**endMethod**
Type: `System..::.Func(Of (IAsyncResult, TResult)>)`
The delegate that ends the asynchronous operation.

**state**
Type: `System..::.Object`
An object containing data to be used by the `beginMethod` delegate.

**creationOptions**
Type: `System.Threading.Tasks..::.TaskCreationOptions`
The `TaskCreationOptions` value that controls the behavior of the created `Task`. 

---

Note: The `TaskCreationOptions` value is used to control the behavior of the `Task` created by the `FromAsync` method. It can be used to set the `Task`'s priority, abortability, and other properties.

---

Example usage:
```csharp
var resultTask = myTaskSource.FromAsync(
    async (callback, state) => { /* async block */ },
    result => { /* callback */ },
    /* state */
    TaskCreationOptions.LongRunning
);
```
Type Parameters

TResult
   The type of the result available through the Task.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...FromAsync<Of <(TArg1)>>) Method (Func<Of <(TArg1, AsyncCallback, Object, IAsyncResult)>>, Action<Of <(IAsyncResult)>>, TArg1, Object)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1) ( _
    beginMethod As Func(Of TArg1, AsyncCallback, Object, IAsyncResult), _
    endMethod As Action(Of IAsyncResult), _
    arg1 As TArg1, _
    state As Object _
) As Task

C#

public Task FromAsync<TArg1>(
    Func<TArg1, AsyncCallback, Object, IAsyncResult> beginMethod,
    Action<IAsyncResult> endMethod,
    TArg1 arg1,
    Object state
)

Parameters

beginMethod
    Type: System..::.Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>))
    The delegate that begins the asynchronous operation.

endMethod
    Type: System..::.Action<(Of <(IAsyncResult)>))
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.

state
    Type: System..::.Object
    An object containing data to be used by the beginMethod delegate.
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the `beginMethod`. 
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>beginMethod</code> argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>endMethod</code> argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory.FromAsync<...(Of <(TArg1, TResult)>)> Method (Func<...(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>), Func<...(Of <(IAsyncResult, TResult)>)), TArg1, Object)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function FromAsync(Of TArg1, TResult) ( _
    beginMethod As Func(Of TArg1, AsyncCallback, Object, IAsyncResult, TResult), _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    state As Object _
) As Task(Of TResult)
```

**C#**

```csharp
public Task<TResult> FromAsync<TArg1, TResult>(
    Func<TArg1, AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    Object state
)
```

**Parameters**

**beginMethod**
- Type: System::.Func<(Of<(TArg1, AsyncCallback, Object, IAsyncResult)>)>)
- The delegate that begins the asynchronous operation.

**endMethod**
- Type: System::.Func<(Of<(IAsyncResult, TResult)>)>)
- The delegate that ends the asynchronous operation.

**arg1**
- Type: TArg1
- The first argument passed to the beginMethod delegate.

**state**
- Type: System::.Object
- An object containing data to be used by the beginMethod delegate.
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

TResult
The type of the result available through the Task.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>beginMethod</code> argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>endMethod</code> argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...:::FromAsync Method (IAsyncResult, Action(Of IAsyncResult)), TaskCreationOptions, TaskScheduler)

TaskFactory Class  See Also  Send Feedback

Creates a Task that executes an end method action when a specified IAsyncResult completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync ( _
    asyncResult As IAsyncResult, _
    endMethod As Action(Of IAsyncResult), _
    creationOptions As TaskCreationOptions, _
    scheduler As TaskScheduler _
) As Task

C#

public Task FromAsync(
    IAsyncResult asyncResult,
    Action<IAsyncResult> endMethod,
    TaskCreationOptions creationOptions,
    TaskScheduler scheduler
)

Parameters

asyncResult
    Type: System:::IAsyncResult
    The IAsyncResult whose completion should trigger the processing of the endMethod.

endMethod
    Type: System:::Action(Of (IAsyncResult))>
    The action delegate that processes the completed asyncResult.

creationOptions
    Type: System.Threading.Tasks:::TaskCreationOptions
    The TaskCreationOptions value that controls the behavior of the created Task.

scheduler
    Type: System.Threading.Tasks:::TaskScheduler
The `TaskScheduler` that is used to schedule the task that executes the end method.

**Return Value**

A `Task` that represents the asynchronous operation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a Task that executes an end method function when a specified IAsyncResult completes.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TResult) ( _
    asyncResult As IAsyncResult, _
    endMethod As Func(Of IAsyncResult, TResult), _
    creationOptions As TaskCreationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TResult>(
    IAsyncResult asyncResult,
    Func<IAsyncResult, TResult> endMethod,
    TaskCreationOptions creationOptions,
    TaskScheduler scheduler
)

Parameters

asyncResult
    Type: System.IAsyncResult
    The IAsyncResult whose completion should trigger the processing of the endMethod.

dendMethod
    Type: System.Func<(Of (IAsyncResult, TResult))>
    The function delegate that processes the completed asyncResult.

creationOptions
    Type: System.Threading.Tasks.TaskCreationOptions
    The TaskCreationOptions value that controls the behavior of the created Task.

scheduler
    Type: System.Threading.Tasks.TaskScheduler
The TaskScheduler that is used to schedule the task that executes the end method.
Type Parameters

TResult
   The type of the result available through the Task.

ReturnValue

A Task that represents the asynchronous operation.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...:.FromAsync<(Of <(TArg1)>)> Method (Func<(Of newend<,(TArg1, AsyncCallback, Object, IAsyncResult)>)>, Action<(Of newend<,(IAsyncResult)>)>, TArg1, Object, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1) ( _
    beginMethod As Func(Of TArg1, AsyncCallback, Object, IAsyncResult), _
    endMethod As Action(Of IAsyncResult), _
    arg1 As TArg1, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task

C#

public Task FromAsync<TArg1>(
    Func<TArg1, AsyncCallback, Object, IAsyncResult> beginMethod,
    Action<IAsyncResult> endMethod,
    TArg1 arg1,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
    Type: System....Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>)
    The delegate that begins the asynchronous operation.

demethod
    Type: System....Action<(Of <(IAsyncResult)>)
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.

state
    Type: System....Object
An object containing data to be used by the beginMethod delegate.

creationOptions
Type: `System.Threading.Tasks.TaskCreationOptions`
The TaskCreationOptions value that controls the behavior of the created Task.
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the `beginMethod`.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::FromAsync<(Of <(T1, TResult)>)> Method (Func<(Of <(T1, AsyncCallback, Object, IAsyncResult)>), Func<(Of <(IAsyncResult, TResult)>)), T1, Object, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TResult) (
    beginMethod As Func(Of TArg1, AsyncCallback, Object, IAsyncResult, TResult), _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TResult>(
    Func<TArg1, AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
Type: System....:Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>)
The delegate that begins the asynchronous operation.

endMethod
Type: System....:Func<(Of <(IAsyncResult, TResult)>))
The delegate that ends the asynchronous operation.

arg1
Type: TArg1
The first argument passed to the beginMethod delegate.

state
Type: System....:Object
An object containing data to be used by the beginMethod delegate.

creationOptions
Type: System.Threading.Tasks:::TaskCreationOptions
The TaskCreationOptions value that controls the behavior of the created Task.
Type Parameters

TArg1
   The type of the first argument passed to the beginMethod delegate.

TResult
   The type of the result available through the Task.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...:FromAsync<(Of <(TArg1, TArg2)>)> Method (Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)>, Action<(Of <(IAsyncResult)>)>, TArg1, TArg2, Object)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2) ( _
    beginMethod As Func(Of TArg1, TArg2, AsyncCallback, Object, _
    endMethod As Action(Of IAsyncResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    state As Object _
) As Task

C#

public Task FromAsync<TArg1, TArg2>(
    Func<TArg1, TArg2, AsyncCallback, Object, IAsyncResult> beginMethod,
    Action<IAsyncResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    Object state
)

Parameters

beginMethod
    Type: System....Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)
    The delegate that begins the asynchronous operation.

endMethod
    Type: System....Action<(Of <(IAsyncResult)>)
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.

arg2
    Type: TArg2
The second argument passed to the beginMethod delegate.

state
Type: System::Object
An object containing data to be used by the beginMethod delegate.
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

TArg2
The type of the second argument passed to beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
**Exceptions**

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....:FromAsync<(Of <(TArg1, TArg2, TResult)>)> Method
(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)>,
Func<(Of <(IAsyncResult, TResult)>)>), TArg1, TArg2, Object)
TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to
the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2, TResult) ( _
    beginMethod As Func(Of TArg1, TArg2, AsyncCallback, Object, _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    state As Object _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TArg2, TResult>(
    Func<TArg1, TArg2, AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    Object state
)

Parameters

beginMethod
    Type: System...: Func(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)
    The delegate that begins the asynchronous operation.

endMethod
    Type: System...: Func(Of <(IAsyncResult, TResult)>)
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.

arg2
    Type: TArg2
The second argument passed to the beginMethod delegate.

state
  Type: System::Object
  An object containing data to be used by the beginMethod delegate.
**Type Parameters**

TArg1  
The type of the first argument passed to the beginMethod delegate.

TArg2  
The type of the second argument passed to beginMethod delegate.

TResult  
The type of the result available through the Task.

**Return Value**

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library
TaskFactory.::.FromAsync<(Of <(TArg1, TArg2, TArg3)>)> Method
(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)>), Action<(Of <(IAsyncResult)>)>), TArg1, TArg2, TArg3, Object)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2, TArg3) ( _
    beginMethod As Func(Of TArg1, TArg2, TArg3, AsyncCallback, _
    endMethod As Action(Of IAsyncResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    arg3 As TArg3, _
    state As Object _) As Task

C#

public Task FromAsync<TArg1, TArg2, TArg3>(
    Func<TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    TArg3 arg3,
    Object state
)

Parameters

beginMethod

Type: System...:: Func(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)

The delegate that begins the asynchronous operation.

dendMethod

Type: System...:: Action(Of <(IAsyncResult)>)

The delegate that ends the asynchronous operation.

arg1

Type: TArg1

The first argument passed to the beginMethod delegate.
arg2
   Type: TArg2
   The second argument passed to the beginMethod delegate.

arg3
   Type: TArg3
   The third argument passed to the beginMethod delegate.

state
   Type: System..::.Object
   An object containing data to be used by the beginMethod delegate.
**Type Parameters**

TArg1  
The type of the first argument passed to the `beginMethod` delegate.

TArg2  
The type of the second argument passed to `beginMethod` delegate.

TArg3  
The type of the third argument passed to `beginMethod` delegate.

**Return Value**

The created `Task` that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>beginMethod</code> argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>endMethod</code> argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library

TaskFactory......:FromAsync(Of <(TArg1, TArg2, TArg3, TResult)>) Method
(Func_(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>) Func(Of <(IAsyncResult, TResult)>), TArg1, TArg2, TArg3, Object)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2, TArg3, TResult) ( _
    beginMethod As Func(Of TArg1, TArg2, TArg3, AsyncCallback, _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    arg3 As TArg3, _
    state As Object _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TArg2, TArg3, TResult>(
    Func<TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult, Func<IAsyncResult, TResult>> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    TArg3 arg3,
    Object state
)

Parameters

beginMethod
    Type: System..::.Func<(Of (Of TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)
    The delegate that begins the asynchronous operation.

endMethod
    Type: System..::.Func<(Of (IAsyncResult, TResult)>)
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.
arg2
   Type: TArg2
   The second argument passed to the beginMethod delegate.

arg3
   Type: TArg3
   The third argument passed to the beginMethod delegate.

state
   Type: System::Object
   An object containing data to be used by the beginMethod delegate.
**Type Parameters**

TArg1  
The type of the first argument passed to the `beginMethod` delegate.

TArg2  
The type of the second argument passed to `beginMethod` delegate.

TArg3  
The type of the third argument passed to `beginMethod` delegate.

TResult  
The type of the result available through the Task.

**Return Value**

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System...:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....:FromAsync<(Of <(TArg1, TArg2)>)> Method (Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)>, Action<(Of <(IAsyncResult)>)>, TArg1, TArg2, Object, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2) ( _
    beginMethod As Func(Of TArg1, TArg2, AsyncCallback, Object, _
    endMethod As Action(Of IAsyncResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task

C#

public Task FromAsync<TArg1, TArg2>(
    Func<TArg1, TArg2, AsyncCallback, Object, IAsyncResult> beginMethod,
    Action<IAsyncResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
    Type: System....Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult>)>)
    The delegate that begins the asynchronous operation.

endMethod
    Type: System....Action<(Of <(IAsyncResult)>)>
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.
arg2
  Type: TArg2
  The second argument passed to the beginMethod delegate.

state
  Type: System::Object
  An object containing data to be used by the beginMethod delegate.

creationOptions
  Type: System.Threading.Tasks::TaskCreationOptions
  The TaskCreationOptions value that controls the behavior of the created Task.
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

TArg2
The type of the second argument passed to beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::FromAsync<(Of <(TArg1, TArg2, TResult)>)> Method
(Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)>,
Func<(Of <(IAsyncResult, TResult)>)>), TArg1, TArg2, Object,
TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to
the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2, TResult) (_
    beginMethod As Func(Of TArg1, TArg2, AsyncCallback, Object,
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TArg2, TResult>(
    Func<TArg1, TArg2, AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
    Type: System...:Func(Of <(TArg1, TArg2, AsyncCallback, Object,
    IAsyncResult)>)
    The delegate that begins the asynchronous operation.

endMethod
    Type: System...:Func(Of <(IAsyncResult, TResult)>)
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.
arg2
   Type: TArg2
   The second argument passed to the beginMethod delegate.

state
   Type: System::Object
   An object containing data to be used by the beginMethod delegate.

creationOptions
   Type: System.Threading.Tasks::TaskCreationOptions
   The TaskCreationOptions value that controls the behavior of the created Task.
**Type Parameters**

TArg1
  The type of the first argument passed to the beginMethod delegate.
TArg2
  The type of the second argument passed to beginMethod delegate.
TResult
  The type of the result available through the Task.

**Return Value**

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....:FromAsync<(Of <(TArg1, TArg2, TArg3)>)> Method
(Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), Action<(Of <(IAsyncResult)>), TArg1, TArg2, TArg3, Object, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2, TArg3) ( _
    beginMethod As Func(Of TArg1, TArg2, TArg3, AsyncCallback, _
    endMethod As Action(Of IAsyncResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    arg3 As TArg3, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task

C#

public Task FromAsync<TArg1, TArg2, TArg3>(
    Func<TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    TArg3 arg3,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
    Type: System...:: Func <(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)>)
    The delegate that begins the asynchronous operation.

eendMethod
    Type: System...:: Action<(Of <(IAsyncResult)>))
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
The first argument passed to the `beginMethod` delegate.

arg2
Type: TArg2
The second argument passed to the `beginMethod` delegate.

arg3
Type: TArg3
The third argument passed to the `beginMethod` delegate.

state
Type: System...::Object
An object containing data to be used by the `beginMethod` delegate.

creationOptions
Type: System.Threading.Tasks...::TaskCreationOptions
The `TaskCreationOptions` value that controls the behavior of the created `Task`
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

TArg2
The type of the second argument passed to beginMethod delegate.

TArg3
The type of the third argument passed to beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
.NET Framework Class Library
TaskFactory......: FromAsync<(Of <(TArg1, TArg2, TArg3, TResult)>)> Method (Func_(<Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)>, Func<(Of <(IAsyncResult, TResult)>)>, TArg1, TArg2, TArg3, Object, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2, TArg3, TResult) ( _
    beginMethod As Func(Of TArg1, TArg2, TArg3, AsyncCallback, _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    arg3 As TArg3, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TArg2, TArg3, TResult>(
    Func<TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult, Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    TArg3 arg3,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
Type: System:::Func <(Of <(TArg1, TArg2, TArg3, AsyncCallback,
    Object, IAsyncResult)>)>)
The delegate that begins the asynchronous operation.

endMethod
Type: System:::Func<(Of <(IAsyncResult, TResult)>)>)
The delegate that ends the asynchronous operation.

arg1
Type: TArg1
The first argument passed to the beginMethod delegate.

arg2
  Type: TArg2
  The second argument passed to the beginMethod delegate.

arg3
  Type: TArg3
  The third argument passed to the beginMethod delegate.

state
  Type: System::Object
  An object containing data to be used by the beginMethod delegate.

creationOptions
  Type: System.Threading.Tasks::TaskCreationOptions
  The TaskCreationOptions value that controls the behavior of the created Task.
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

TArg2
The type of the second argument passed to beginMethod delegate.

TArg3
The type of the third argument passed to beginMethod delegate.

TResult
The type of the result available through the Task.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic    C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
TaskFactory...::StartNew Method
TaskFactory Class    See Also    Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartNew(Action)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(TResult)&gt;)((Func(Of &lt;(TResult)&gt;)))</td>
<td>Creates and starts a Task&lt;Of (Of (TResult))).</td>
</tr>
<tr>
<td>StartNew(Action, CancellationToken)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
<tr>
<td>StartNew(Action, TaskCreationOptions)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object)&gt;), Object)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object)&gt;), CancellationToken)</td>
<td>Creates and starts a Task&lt;Of (&lt;TResult&gt;)).</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object)&gt;), TaskCreationOptions)</td>
<td>Creates and starts a Task&lt;Of (&lt;TResult&gt;)).</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object)&gt;, Object)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object)&gt;, Object, CancellationToken)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object)&gt;, Object, TaskCreationOptions)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object)&gt;, Object, CancellationToken)</td>
<td>Creates and starts a Task&lt;Of (&lt;TResult&gt;)&gt;.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object), TResult)&gt;)(Func(Of &lt;(Object, TResult)&gt;), Object)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object), TResult)&gt;)(Func(Of &lt;(Object, TResult)&gt;), CancellationToken)</td>
<td>Creates and starts a Task&lt;Of (&lt;TResult&gt;)&gt;.</td>
</tr>
<tr>
<td>StartNew(Of &lt;(Object), TResult)&gt;)(Func(Of &lt;(Object, TResult)&gt;), TaskCreationOptions)</td>
<td>Creates and starts a <code>Task</code>.</td>
</tr>
</tbody>
</table>
TaskCreationOptions, TaskScheduler) a Task.

StartNew(Of (TResult>)>(Func(Of (TResult>), CancellationToken,
TaskCreationOptions, TaskScheduler) Creates and starts a Task(Of (TResult>)).

StartNew(Action(Of (Object>)>, Object,
CancellationToken, TaskCreationOptions,
TaskScheduler) Creates and starts a Task.

StartNew(Of (TResult>)>(Func(Of (Object, 
TResult>)>, Object, CancellationToken,
TaskCreationOptions, TaskScheduler) Creates and starts a Task(Of (TResult>)).
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates and starts a Task.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function StartNew ( _
    action As Action _
) As Task
```

**C#**

```csharp
public Task StartNew(
    Action action
)
```

**Parameters**

`action`

Type: System...:::Action

The action delegate to execute asynchronously.

**Return Value**

The started `Task`.
Remarks

Calling StartNew is functionally equivalent to creating a Task using one of its constructors and then calling Start to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the action argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::StartNew<(Of <(TResult)>)> Method (Func<(Of <(TResult)>)>)

Creates and starts a Task<(Of <(TResult)>)>.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew(Of TResult) ( _
    function As Func(Of TResult) _
) As Task(Of TResult)

C#

public Task<TResult> StartNew<TResult>(
    Func<TResult> function
)

Parameters

function

    Type: System...::.Func(Of <(TResult)>)

A function delegate that returns the future result to be available through the Task<(Of <(TResult)>)).
Type Parameters

TResult
   The type of the result available through the Task.

Return Value

The started Task<Of <(TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<TResult>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, `StartNew` is the recommended approach for both simplicity and performance.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory.....:StartNew Method (Action, CancellationToken)

TaskFactory Class  See Also  Send Feedback

Creates and starts a Task.

**Namespace:**  System.Threads.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function StartNew ( _
    action As Action, _
    cancellationToken As CancellationToken _
) As Task
```

#### C#

```csharp
public Task StartNew(
    Action action,
    CancellationToken cancellationToken
)
```

### Parameters

**action**
- **Type:** System::Action
- The action delegate to execute asynchronously.

**cancellationToken**
- **Type:** System.Threading::CancellationToken
- The `CancellationToken` that will be assigned to the new task.

### Return Value

The started `Task`. 
Remarks

Calling StartNew is functionally equivalent to creating a Task using one of its constructors and then calling Start to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the action argument is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided <a href="https://docs.microsoft.com/en-us/dotnet/api/system.cancellationtoken">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...:.StartNew Method (Action, TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates and starts a Task.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
### Syntax

**Visual Basic (Declaration)**

```vbnet
Public Function StartNew ( _
    action As Action, _
    creationOptions As TaskCreationOptions _
) As Task
```

**C#**

```csharp
public Task StartNew(
    Action action,
    TaskCreationOptions creationOptions
)
```

### Parameters

**action**
- **Type:** System:::Action
  - The action delegate to execute asynchronously.

**creationOptions**
- **Type:** System.Threading.Tasks:::TaskCreationOptions
  - A TaskCreationOptions value that controls the behavior of the created Task.

### Return Value

- The started Task.
Remarks

Calling StartNew is functionally equivalent to creating a Task using one of its constructors and then calling Start to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the action argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory.....:StartNew Method (Action<Of (Object)>, Object)

TaskFactory Class  See Also  Send Feedback

Creates and starts a Task.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    action As Action(Of Object), _
    state As Object _
) As Task

C#

public Task StartNew(
    Action<Object> action,
    Object state
)

Parameters

action
    Type: System...::Action<(Of <(Object)>>)
    The action delegate to execute asynchronously.

state
    Type: System...::Object
    An object containing data to be used by the action delegate.

Return Value

The started Task.
Remarks

Calling StartNew is functionally equivalent to creating a Task using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the action argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....:StartNew(Of <(TResult)>) Method (Func(Of <(TResult)>) _, CancellationToken)

Creates and starts a Task(Of <(TResult)>)).

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew(Of TResult) ( _
    function As Func(Of TResult), _
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> StartNew<TResult>(
    Func<TResult> function,
    CancellationToken cancellationToken
)

Parameters

function
    Type: System..::.Func<(Of <(TResult)>>)
    A function delegate that returns the future result to be available through the Task<(Of <(TResult)>)>.

cancellationToken
    Type: System.Threading..::.CancellationToken
    The CancellationToken that will be assigned to the new Task
Type Parameters

**T**Result

The type of the result available through the Task.

**Return Value**

The started Task<Of <(TResult)>).
Calling StartNew is functionally equivalent to creating a `Task<(<TResult>)>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, `StartNew` is the recommended approach for both simplicity and performance.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....:StartNew<(Of <(TResult)>)> Method (Func<(Of <(TResult)>)>), TaskCreationOptions)

TaskFactory Class  See Also  Send Feedback

Creates and starts a Task<(Of <(TResult)>)>.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew(Of TResult) ( _
    function As Func(Of TResult), _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> StartNew<TResult>(
    Func<TResult> function,
    TaskCreationOptions creationOptions
)

Parameters

function
    Type: System::Func(Of (Of TResult)>)
    A function delegate that returns the future result to be available through the Task(Of (Of TResult)>).

creationOptions
    Type: System.Threading.Tasks::TaskCreationOptions
    A TaskCreationOptions value that controls the behavior of the created Task(Of (Of TResult)>).
Type Parameters

TResult

The type of the result available through the Task.

Return Value

The started Task<(<TResult>)>.
Remarks

Calling StartNew is functionally equivalent to creating a Task<
 VoorGets
> using one of its constructors and then calling Start to schedule it
for execution. However, unless creation and scheduling must be separated,
StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory.<>.StartNew<>() Method (Func<>()

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew(Of TResult) ( _
    function As Func(Of Object, TResult), _
    state As Object _
) As Task(Of TResult)

C#

public Task<TResult> StartNew<TResult>(
    Func<Object, TResult> function,
    Object state
)

Parameters

function
    Type: System:::Func<(Of <(Object, TResult)>)>)
    A function delegate that returns the future result to be available through the
    Task<(Of <(TResult)>)).

state
    Type: System:::Object
    An object containing data to be used by the function delegate.
**Type Parameters**

**TResult**

The type of the result available through the Task.

**Return Value**

The started Task<(Of <(TResult)>)>.
Remarks

Calling StartNew is functionally equivalent to creating a `Task<Of TResult>)` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....::StartNew Method (Action<Of<(Object)>>, Object, CancellationToken)

TaskFactory Class  See Also  Send Feedback

Creates and starts a Task.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    action As Action(Of Object), _
    state As Object, _
    cancellationToken As CancellationToken _
) As Task

C#

public Task StartNew(
    Action<
        Object>
        action,
    Object state,
    CancellationToken cancellationToken
)

Parameters

action
Type: System:::Action(Of (<Object>))
The action delegate to execute asynchronously.

state
Type: System:::Object
An object containing data to be used by the action delegate.

cancellationToken
Type: System.Threading:::CancellationToken
The CancellationToken that will be assigned to the new Task

Return Value

The started Task.
Remarks

Calling StartNew is functionally equivalent to creating a Task using one of its constructors and then calling Start to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the action argument is null.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided <a href="https://docs.microsoft.com/en-us/dotnet/api/system.runtime.cancellation.cancellationtoken">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...StartNew Method (Action<Of (Object)>, Object, TaskCreationOptions)

Creates and starts a Task.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function StartNew (_
    action As Action(Of Object), _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task
```

#### C#

```csharp
public Task StartNew(
    Action<Object> action,
    Object state,
    TaskCreationOptions creationOptions
)
```

### Parameters

**action**

- Type: `System:::Action<Of <(Object)>>)`
- The action delegate to execute asynchronously.

**state**

- Type: `System:::Object`
- An object containing data to be used by the action delegate.

**creationOptions**

- Type: `System.Threading.Tasks:::TaskCreationOptions`
- A TaskCreationOptions value that controls the behavior of the created `Task`.

### Return Value

The started `Task`. 
Calling `StartNew` is functionally equivalent to creating a Task using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, `StartNew` is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the action argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::StartNew(Of<TResult>) Method (Func(Of<Object,TResult>), Object, CancellationToken)

Creates and starts a Task(Of<TResult>্).}

**Namespace:**  System Threading Tasks  
**Assembly:**  System Threading (in System Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew(Of TResult) ( _
    function As Func(Of Object, TResult), _
    state As Object, _
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> StartNew<TResult>(
    Func<Object, TResult> function,
    Object state,
    CancellationToken cancellationToken
)

Parameters

function
    Type: System...:::Func(Of <(Object, TResult)>)
    A function delegate that returns the future result to be available through the Task<Of <(TResult)>).

state
    Type: System...:::Object
    An object containing data to be used by the function delegate.

cancellationToken
    Type: System.Threading...:::CancellationToken
    The CancellationToken that will be assigned to the new Task
Type Parameters

TResult
    The type of the result available through the Task.

Return Value

The started Task<Of <(TResult)>).
Calling StartNew is functionally equivalent to creating a `Task<(Of TResult)>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td></td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class  
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  Visual Basic  C#  C#
.NET Framework Class Library
TaskFactory....:StartNew<(Of <(TResult)>)> Method (Func<(Of <(Object, TResult)>)>), Object, TaskCreationOptions)
TaskFactory Class  See Also  Send Feedback

Creates and starts a Task<(Of <(TResult)>)>.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew(Of TResult) ( _
    function As Func(Of Object, TResult), _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> StartNew<TResult>(
    Func<Object, TResult> function,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

function
    Type: System:::Func(Of <*Object, TResult*>)
    A function delegate that returns the future result to be available through the Task<*Object, TResult*>.

state
    Type: System:::Object
    An object containing data to be used by the function delegate.

creationOptions
    Type: System.Threading.Tasks:::TaskCreationOptions
    A TaskCreationOptions value that controls the behavior of the created Task<*Object, TResult*>.
Type Parameters

TResult
   The type of the result available through the Task.

Return Value

The started Task<(Of <(TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<Of TResult>`) using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory...::StartNew Method (Action, CancellationToken, TaskCreationOptions, TaskScheduler)

Creates and starts a Task.

**Namespace:**  System.Threading.Tasks
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    action As Action, _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions, _
    scheduler As TaskScheduler _
) As Task

C#

public Task StartNew(
    Action action,
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions,
    TaskScheduler scheduler
)

Parameters

action
    Type: System...::.Action
    The action delegate to execute asynchronously.

cancellationToken
    Type: System.Threading...::.CancellationToken
    The CancellationToken that will be assigned to the new Task

creationOptions
    Type: System.Threading.Tasks...::.TaskCreationOptions
    A TaskCreationOptions value that controls the behavior of the created Task.

scheduler
    Type: System.Threading.Tasks...::.TaskScheduler
    The TaskScheduler that is used to schedule the created Task.
Return Value

The started Task.
Remarks

Calling StartNew is functionally equivalent to creating a Task using one of its constructors and then calling Start to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the action argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates and starts a Task<Of (TResult)>.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function StartNew(Of TResult) ( _
    function As Func(Of TResult), _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)
```

#### C#

```csharp
public Task<TResult> StartNew<TResult>(
    Func<TResult> function,
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions,
    TaskScheduler scheduler
)
```

### Parameters

- **function**
  - Type: `System.Func(Of TResult)`
  - A function delegate that returns the future result to be available through the `Task(Of TResult)`.

- **cancellationToken**
  - Type: `System.Threading.CancellationToken`
  - The `CancellationToken` that will be assigned to the new task.

- **creationOptions**
  - Type: `System.Threading.Tasks.TaskCreationOptions`
  - A `TaskCreationOptions` value that controls the behavior of the created `Task(Of TResult)`.

- **scheduler**
  - Type: `System.Threading.Tasks.TaskScheduler`
  - The `TaskScheduler` for the new task.
The TaskScheduler that is used to schedule the created Task{TResult}.
**Type Parameters**

TResult

The type of the result available through the Task.

**Return Value**

The started Task<Of <(TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<Of TResult>>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskFactory....:StartNew Method (Action<Of <(Object)>>, Object, CancellationToken, TaskCreationOptions, TaskScheduler)
TaskFactory Class  See Also  Send Feedback

Creates and starts a Task.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew (  
    action As Action(Of Object), _
    state As Object, _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions, _
    scheduler As TaskScheduler _
) As Task

C#

public Task StartNew(
    Action<
        Object
    > action,
    Object state,
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions,
    TaskScheduler scheduler
)

Parameters

action
    Type: System...::Action<
        (Of <
            Object
        >
    )>  
The action delegate to execute asynchronously.

state
    Type: System...::Object
    An object containing data to be used by the action delegate.

cancellationToken
    Type: System.Threading...::CancellationToken
    The CancellationToken that will be assigned to the new task.

creationOptions
    Type: System.Threading.Tasks...::TaskCreationOptions
    A TaskCreationOptions value that controls the behavior of the created Task.
scheduler
Type: System.Threading.Tasks::TaskScheduler
The TaskScheduler that is used to schedule the created Task.

Return Value
The started Task.
Remarks

Calling StartNew is functionally equivalent to creating a Task using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the action argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....:StartNew<Of <(TResult)>>() Method (Func<Of <(Object, TResult)>>(), Object, CancellationToken, TaskCreationOptions, TaskScheduler)

TaskFactory Class  See Also  Send Feedback

Creates and starts a Task<Of <(TResult)>>().

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew(Of TResult) (_
    function As Func(Of Object, TResult), _
    state As Object, _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)

C#

public Task<TResult> StartNew<TResult>(
    Func<Object, TResult> function,
    Object state,
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions,
    TaskScheduler scheduler
)

Parameters

function
    Type: System...:::Func<(Of <(Object, TResult)>)>)
    A function delegate that returns the future result to be available through the
    Task<(Of <(TResult)>)>.

state
    Type: System...:::Object
    An object containing data to be used by the function delegate.

cancellationToken
    Type: System.Threading...:::CancellationToken
    The CancellationToken that will be assigned to the new task.

creationOptions
    Type: System.Threading.Tasks...:::TaskCreationOptions
A TaskCreationOptions value that controls the behavior of the created Task<(Of <(TResult)>)>.

scheduler
Type: System.Threading.Tasks...:::TaskScheduler
The TaskScheduler that is used to schedule the created Task{TResult}.
**Type Parameters**

**TResult**

The type of the result available through the Task.

**Return Value**

The started Task<Of <(TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<TResult>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, `StartNew` is the recommended approach for both simplicity and performance.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::.ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System:::.ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
<tr>
<td>System:::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The **TaskFactory** type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancellationToken</td>
<td>Gets the default CancellationToken of this TaskFactory.</td>
</tr>
<tr>
<td>ContinuationOptions</td>
<td>Gets the TaskContinuationOptions value of this TaskFactory.</td>
</tr>
<tr>
<td>CreationOptions</td>
<td>Gets the TaskCreationOptions value of this TaskFactory.</td>
</tr>
<tr>
<td>Scheduler</td>
<td>Gets the TaskScheduler of this TaskFactory.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory:::CancellationToken Property

TaskFactory Class  See Also  Send Feedback

Gets the default CancellationToken of this TaskFactory.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property CancellationToken As CancellationToken

C#

public CancellationToken CancellationToken { get; }
Remarks

This property returns the default CancellationToken that will be assigned to all tasks created by this factory unless another CancellationToken value is explicitly specified during the call to the factory methods.
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the `TaskContinuationOptions` value of this `TaskFactory`.

**Namespace:**  [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property ContinuationOptions As TaskContinuationOptions

C#

public TaskContinuationOptions ContinuationOptions { get; }
Remarks

This property returns the default continuation options for this factory. They will be used to create all continuation tasks unless other options are explicitly specified during calls to this factory's methods.
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the `TaskCreationOptions` value of this TaskFactory.

**Namespace:**  [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property CreationOptions As TaskCreationOptions

C#
public TaskCreationOptions CreationOptions { get; }
Remarks

This property returns the default creation options for this factory. They will be used to create all tasks unless other options are explicitly specified during calls to this factory’s methods.
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory....Scheduler Property

TaskFactory Class  See Also  Send Feedback

Gets the TaskScheduler of this TaskFactory.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Scheduler As TaskScheduler

C#

public TaskScheduler Scheduler { get; }
Remarks

This property returns the default scheduler for this factory. It will be used to schedule all tasks unless another scheduler is explicitly specified during calls to this factory's methods. If null, TaskScheduler.Current will be used.
See Also

TaskFactory Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Provides support for creating and scheduling Task{TResult} objects.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
    ExternalThreading := True)> _
Public Class TaskFactory(Of TResult)

C#

[HostProtectionAttribute(SecurityAction.LinkDemand, Synchronization
    ExternalThreading = true)]
public class TaskFactory<TResult>
Type Parameters

TResult
The type of the results that are available though the Task{TResult} objects that are associated with the methods in this class.
Remarks

There are many common patterns for which tasks are relevant. The TaskFactory(Of TResult>) class encodes some of these patterns into methods that pick up default settings, which are configurable through its constructors.

A default instance of TaskFactory(Of TResult>) is available through the Task{TResult}.Factory property.
Inheritance Hierarchy

System..::.Object
System.Threading.Tasks..::.TaskFactory<Of <(TResult)>)}
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

TaskFactory(Of TResult) Constructor

See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt;()()()</td>
<td>Initializes a TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; instance with the default configuration.</td>
</tr>
<tr>
<td>TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; (CancellationToken)</td>
<td>Initializes a TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; instance with the default configuration.</td>
</tr>
<tr>
<td>TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; (TaskScheduler)</td>
<td>Initializes a TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; instance with the specified configuration.</td>
</tr>
<tr>
<td>TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; (TaskCreationOptions, TaskContinuationOptions)</td>
<td>Initializes a TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; instance with the specified configuration.</td>
</tr>
<tr>
<td>TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; (CancellationToken, TaskCreationOptions, TaskContinuationOptions, TaskScheduler)</td>
<td>Initializes a TaskFactory&lt;(Of &lt;(TResult)&gt;)&gt; instance with the specified configuration.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)> Constructor

Initializes a TaskFactory<(Of <(TResult)>)> instance with the default configuration.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public TaskFactory()
Remarks

This constructor creates a `TaskFactory<Of <(TResult)>>` instance with a default configuration. The `TaskCreationOptions` property is initialized to `TaskCreationOptions.None`, the `TaskContinuationOptions` property is initialized to `TaskContinuationOptions.None`, and the `TaskScheduler` property is initialized to the current scheduler (see `TaskScheduler.Current`).
See Also

TaskFactory(Of (TResult)> ) Class
TaskFactory(Of (TResult)> ) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a `TaskFactory(Of<TResult>`) instance with the default configuration.

**Namespace:**  `System.Threading.Tasks`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Sub New ( _
cancellationToken As CancellationToken _
)

C#

public TaskFactory(
    CancellationToken cancellationToken
)

Parameters

cancellationToken
Type: System.Threading.CancellationToken
The default CancellationToken that will be assigned to tasks created by this TaskFactory unless another CancellationToken is explicitly specified while calling the factory methods.
Remarks

This constructor creates a `TaskFactory<Of<TResult>>` instance with a default configuration. The `TaskCreationOptions` property is initialized to `TaskCreationOptions.None`, the `TaskContinuationOptions` property is initialized to `TaskContinuationOptions.None`, and the `TaskScheduler` property is initialized to the current scheduler (see `TaskScheduler.Current`).
See Also

TaskFactory(Of TResult) Class
TaskFactory(Of TResult) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of TResult) Constructor (TaskScheduler)

TaskFactory(Of TResult) Class

Initializes a TaskFactory(Of TResult) instance with the specified configuration.

Namespace: System.Threading.Tasks

Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    scheduler As TaskScheduler _
)

C#

public TaskFactory(
    TaskScheduler scheduler
)

Parameters

scheduler
    Type: System.Threading.Tasks:::TaskScheduler
The TaskScheduler to use to schedule any tasks created with this TaskFactory{TResult}. A null value indicates that the current TaskScheduler should be used.
Remarks

With this constructor, the `TaskCreationOptions` property is initialized to `TaskCreationOptions.None`, the `TaskContinuationOptions` property is initialized to `TaskContinuationOptions.None`, and the `TaskScheduler` property is initialized to `scheduler`, unless it’s null, in which case the property is initialized to the current scheduler (see `TaskScheduler.Current`).
See Also

TaskFactory(Of TResult) Class
TaskFactory(Of TResult) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskFactory(Of TResult) Constructor (TaskCreationOptions, TaskContinuationOptions)
TaskFactory(Of TResult) Class  See Also  Send Feedback

Initializes a TaskFactory(Of TResult) instance with the specified configuration.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
**Syntax**

### Visual Basic (Declaration)

```vbnet
Public Sub New ( _
    creationOptions As TaskCreationOptions, _
    continuationOptions As TaskContinuationOptions _
)
```

### C#

```csharp
public TaskFactory(
    TaskCreationOptions creationOptions,
    TaskContinuationOptions continuationOptions
)
```

**Parameters**

- **creationOptions**
  - Type: `System.Threading.Tasks.TaskCreationOptions`
  - The default `TaskCreationOptions` to use when creating tasks with this `TaskFactory{TResult}`.

- **continuationOptions**
  - Type: `System.Threading.Tasks.TaskContinuationOptions`
  - The default `TaskContinuationOptions` to use when creating continuation tasks with this `TaskFactory{TResult}`.
Remarks

With this constructor, the `TaskCreationOptions` property is initialized to `creationOptions`, the `TaskContinuationOptions` property is initialized to `continuationOptions`, and the `TaskScheduler` property is initialized to the current scheduler (see `TaskScheduler.Current`).
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.ArgumentOutOfRangeException</code></td>
<td>The exception that is thrown when the <code>creationOptions</code> argument or the <code>continuationOptions</code> argument specifies an invalid value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
TaskFactory(Of TResult>) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory\<(Of\<(TResult\>)\>)\> Constructor (CancellationToken, TaskCreationOptions, TaskContinuationOptions, TaskScheduler)

TaskFactory\<(Of\<(TResult\>)\>)\> Class  See Also  Send Feedback

Initializes a TaskFactory\<(Of\<(TResult\>)\>)\> instance with the specified configuration.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
)

C#

public TaskFactory(
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

cancellationToken
Type: System.Threading.CancellationToken
The default CancellationToken that will be assigned to tasks created by this TaskFactory unless another CancellationToken is explicitly specified while calling the factory methods.

creationOptions
Type: System.Threading.Tasks.TaskCreationOptions
The default TaskCreationOptions to use when creating tasks with this TaskFactory{TResult}.

continuationOptions
Type: System.Threading.Tasks.TaskContinuationOptions
The default TaskContinuationOptions to use when creating continuation tasks with this TaskFactory{TResult}. 
scheduler

Type: System.Threading.Tasks.TaskScheduler
The default TaskScheduler to use to schedule any Tasks created with this TaskFactory{TResult}. A null value indicates that TaskScheduler.Current should be used.
Remarks

With this constructor, the TaskCreationOptions property is initialized to creationOptions, the TaskContinuationOptions property is initialized to continuationOptions, and the TaskScheduler property is initialized to scheduler, unless it's null, in which case the property is initialized to the current scheduler (see TaskScheduler.Current).
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::..ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument or the continuationOptions arguments specifies an invalid value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
TaskFactory(Of TResult) Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

TaskFactory(Of TResult) Methods

The TaskFactory(Of TResult) type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContinueWhenAll</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ContinueWhenAny</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>FromAsync</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>StartNew</td>
<td>Overloaded.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □  C#
□  Include Protected Members
□  Include Inherited Members
.NET Framework Class Library
TaskFactory<Of <(TResult)>)::.::ContinueWhenAll Method
TaskFactory<Of <(TResult)>) Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ContinueWhenAll(array&lt;Task&gt;[][], Func&lt;Of (array&lt;Task&gt;[][], TResult)&gt;)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll&lt;(Of &lt;(TAntecedentResult)&gt;)&gt; (array&lt;Task&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;&gt;[][], Func&lt;(Of &lt;(array&lt;Task&gt;(Of &lt;(TAntecedentResult)&gt;)&gt;&gt;[][])[], TResult)&gt;)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll(array&lt;Task&gt;[][], Func&lt;Of (array&lt;Task&gt;[][], TResult)&gt;, CancellationToken)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll(array&lt;Task&gt;[][], Func&lt;Of (array&lt;Task&gt;[][], TResult)&gt;, TaskContinuationOptions)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll&lt;(Of &lt;(TAntecedentResult)&gt;)&gt; (array&lt;Task&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;&gt;[][], Func&lt;(Of &lt;(array&lt;Task&gt;(Of &lt;(TAntecedentResult)&gt;)&gt;]&gt;[][])[], TResult)&gt;), CancellationToken)</code></td>
<td>Creates a continuation Task that will be started upon the completion of a set of provided Tasks.</td>
</tr>
<tr>
<td><code>ContinueWhenAll&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;)</code></td>
<td>Creates a continuation Task</td>
</tr>
</tbody>
</table>
(array<Task<(Of <(TAntecedentResult>)>)>[][], Func<(Of <(array<Task<(Of <(TAntecedentResult>)>)>[][])[], TResult>)>, TaskContinuationOptions)

ContinueWhenAll(array<Task>[][], Func<(Of <(array<Task>[][])[], TResult>)>, CancellationToken, TaskContinuationOptions, TaskScheduler)

that will be started upon the completion of a set of provided Tasks.

ContinueWhenAll<(Of <(TAntecedentResult)>)>(array<Task<(Of <(TAntecedentResult)>)>[][])[], Func<(Of <(array<Task<(Of <(TAntecedentResult)>)>[][])[], TResult>)>, CancellationToken, TaskContinuationOptions, TaskScheduler)

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.
See Also

TaskFactory<(Of (TResult)>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>).::.ContinueWhenAll Method (array<Task>[][], Func<(Of <(array<Task>[][])[], TResult)>))

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

**Namespace:**  System.Threading.Tasks
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task(), TResult) _) _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll(
    Task[] tasks,
    Func<TTask[], TResult> continuationFunction
)

Parameters

tasks
    Type: array< System.Threading.Tasks..::.Task >[]()
    The array of tasks from which to continue.

continuationFunction
    Type: System..::.Func(Of(array< Task >[][], TResult>)>
    The function delegate to execute when all tasks in the tasks array have completed.

Return Value

The new continuation Task.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<Of <(TResult)?>>.::.ContinueWhenAll<Of <(TAntecedentResult)?>> Method (array<Task<Of <(TAntecedentResult)?>>>[][], Func<Of <(array<Task<Of <(TAntecedentResult)?>>>[][], TResult>>)))

TaskFactory<Of <(TResult)?>> Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult)();
) As Task(Of TResult)

C#

public TResult ContinueWhenAll<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult][]> continuationFunction
)

Parameters

tasks
    Type: array< System.Threading.Tasks::: Task <(Of <(TAntecedentResult)>) > [] >[]
The array of tasks from which to continue.

continuationFunction
    Type: System::: Func< (Of< array< Task <(Of< (TAntecedentResult)>) > > [[]] >, TResult > )>
The function delegate to execute when all tasks in the tasks array have completed.
Type Parameters

TAntecedentResult

The type of the result of the antecedent tasks.

Return Value

The new continuation Task<(Of <(TResult)>)>.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<Of (TResult)>..::.ContinueWhenAll Method (array<Task>[][], Func<Of (array<Task>[][])[], TResult[]), CancellationToken)
TaskFactory<Of (TResult)>() Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task(), TResult), _
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll(
    Task[] tasks,
    Func<Task[], TResult> continuationFunction,
    CancellationToken cancellationToken
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task >[]
    The array of tasks from which to continue.

continuationFunction
    Type: System.Func(Of (array< Task >[])[], TResult)
    The function delegate to execute when all tasks in the tasks array have completed.

cancellationToken
    Type: System.Threading.CancellationToken
    The CancellationToken that will be assigned to the new continuation task.

Return Value

The new continuation Task.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)>::ContinueWhenAll Method (array<Task>[][], Func<(Of <(array<Task>[][])[], TResult)[]>, TaskContinuationOptions)

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
## Syntax

### Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAll ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task(), TResult), _
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)
```

### C#

```csharp
public Task<TResult> ContinueWhenAll(
    Task[] tasks,
    Func<Task[], TResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)
```

## Parameters

**tasks**

Type: array< System.Threading.Tasks.Task >[]

The array of tasks from which to continue.

**continuationFunction**

Type: System..::.Func(Of <array< Task >>[](), TResult>)

The function delegate to execute when all tasks in the tasks array have completed.

**continuationOptions**

Type: System..::.TaskContinuationOptions

The TaskContinuationOptions value that controls the behavior of the created continuation Task.

## Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...:ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System...:ArgumentNullException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System...:ArgumentException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid</td>
</tr>
<tr>
<td></td>
<td>TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System...:ArgumentOutOfRangeException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<((Of<br>)>) Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of (TResult)>) ..::. ContinueWhenAll(Of (TAntecedentResult)>) Method (array<Task(Of (TAntecedentResult)>)[], Func(Of (array<Task(Of (TAntecedentResult)>)[], CancellationToken))

TaskFactory(Of (TResult)>) Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult)();
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>[], TResult> continuationFunction,
    CancellationToken cancellationToken
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task<Of
<(<TAntecedentResult>)>[]) >[][[]
    The array of tasks from which to continue.

continuationFunction
    Type: System:::Func<Of <(array< Task<Of <(<TAntecedentResult>)> >
[][[], TResult>)>)
    The function delegate to execute when all tasks in the tasks array have
completed.

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.
Type Parameters

TAntecedentResult
  The type of the result of the antecedent tasks.

Return Value

The new continuation Task<Of<(TResult)>>.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided <a href="https://learn.microsoft.com">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library

TaskFactory(Of TResult)::.ContinueWhenAll(Of (TAntecedentResult)) Method (array<Task(Of (TAntecedentResult))>[][], Func(Of (array<Task(Of (TAntecedentResult))>[][])[], TResult[][], TaskContinuationOptions))

TaskFactory(Of TResult) Class  See Also  Send Feedback

 Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult)();
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>[][], TResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)

Parameters

tasks
Type: array<System.Threading.Tasks::Task>(Of <(TAntecedentResult)>[]) >[]
The array of tasks from which to continue.

continuationFunction
Type: System::Func(Of (array<System.Threading.Tasks::Task>(Of <(TAntecedentResult)>[]) >[])[], TResult>)>
The function delegate to execute when all tasks in the tasks array have completed.

continuationOptions
Type: System.Threading.Tasks::TaskContinuationOptions
The TaskContinuationOptions value that controls the behavior of the created continuation Task.
**Type Parameters**

TAntecedentResult

The type of the result of the antecedent tasks.

**Return Value**

The new continuation Task<(Of (TResult)>).
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<Of <(TResult)>,::<ContinueWhenAll Method (array<Task>[], Func<Of <(array<Task>[][])[], TResult>>, CancellationToken, TaskContinuationOptions, TaskScheduler)
TaskFactory<Of <(TResult)>> Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task(), TResult), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll(
    Task[] tasks,
    Func<Task[], TResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task >[]()
    The array of tasks from which to continue.

continuationFunction
    Type: System:::Func(Of <(array< Task >[])[], TResult>->)
    The function delegate to execute when all tasks in the tasks array have completed.

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.

continuationOptions
    Type: System.Threading.Tasks:::TaskContinuationOptions
The `TaskContinuationOptions` value that controls the behavior of the created continuation Task.

**scheduler**
- Type: `System.Threading.Tasks::TaskScheduler`
- The `TaskScheduler` that is used to schedule the created continuation Task.

**Return Value**

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
.NET Framework Class Library
TaskFactory(Of TResult).ContinueWhenAll(Of TAntecedentResult). Method (array<Task(Of TAntecedentResult)>[], Func(Of array<Task(Of TAntecedentResult)>[])[], CancellationToken, TaskContinuationOptions, TaskScheduler)

TaskFactory(Of TResult). Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of a set of provided Tasks.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAll(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult)(), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAll<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>[], TResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

tasks
Type: array<System.Threading.Tasks.Task(Of TAntecedentResult)[]>()
The array of tasks from which to continue.

continuationFunction
Type: System.Func(Of (array<System.Threading.Tasks.Task(Of TAntecedentResult)[])>(), TResult)
The function delegate to execute when all tasks in the tasks array have completed.

cancellationToken
Type: System.Threading.CancellationToken
The CancellationToken that will be assigned to the new continuation task.
continuationOptions
Type: System.Threading.Tasks.TaskContinuationOptions
The TaskContinuationOptions value that controls the behavior of the created continuation Task.

scheduler
Type: System.Threading.Tasks.TaskScheduler
The TaskScheduler that is used to schedule the created continuation Task<Of <(TResult)>).
**Type Parameters**

**TAntecedentResult**
The type of the result of the antecedent tasks.

**Return Value**
The new continuation `Task<Of <(TResult)>`).
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAll.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td></td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
ContinueWhenAll Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

TaskFactory(Of TResult).ContinueWhenAny Method

TaskFactory(Of TResult) Class   See Also   Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ContinueWhenAny(array&lt;Task&gt;[][], Func&lt;(Task, TResult)&gt;)</code></td>
<td>Creates a continuation Task that will be started upon the completion of any Task in the provided set.</td>
</tr>
<tr>
<td>`ContinueWhenAny((Of &lt;(TAntecedentResult)&gt;)&gt;(array&lt;Task&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;[][], Func&lt;(Of &lt;(Task&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;, TResult)&gt;), CancellationToken)</td>
<td>Creates a continuation Task that will be started upon the completion of any Task in the provided set.</td>
</tr>
<tr>
<td>`ContinueWhenAny(array&lt;Task&gt;[][], Func&lt;(Of &lt;(Task, TResult)&gt;)&gt;, CancellationToken)</td>
<td>Creates a continuation Task that will be started upon the completion of any Task in the provided set.</td>
</tr>
<tr>
<td><code>ContinueWhenAny(array&lt;Task&gt;[][], Func&lt;(Of &lt;(Task, TResult)&gt;)&gt;, TaskContinuationOptions)</code></td>
<td>Creates a continuation Task that will be started upon the completion of any Task in the provided set.</td>
</tr>
<tr>
<td><code>ContinueWhenAny((Of &lt;(TAntecedentResult)&gt;)&gt;(array&lt;Task&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;[][], Func&lt;(Of &lt;(Task&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;, TResult)&gt;), CancellationToken)</code></td>
<td>Creates a continuation Task that will be started upon the completion of any Task in the provided set.</td>
</tr>
<tr>
<td><code>ContinueWhenAny&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;(array&lt;Task&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;[][], Func&lt;(Of &lt;(Task&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;, TResult)&gt;), CancellationToken)</code></td>
<td>Creates a continuation Task that will be started upon the completion of any Task in the provided set.</td>
</tr>
<tr>
<td><code>ContinueWhenAny&lt;(Of &lt;(TAntecedentResult)&gt;)&gt;</code></td>
<td>Creates a continuation Task</td>
</tr>
</tbody>
</table>
that will be started upon the completion of any Task in the provided set.

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Creates a continuation Task that will be started upon the completion of any Task in the provided set.
See Also

TaskFactory(Of TResult) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of TResult).ContinueWhenAny Method (array<Task>[] [], Func(Of (Task, TResult)>)

TaskFactory(Of TResult) Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task, TResult) _) _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny(
    Task[] tasks,
    Func<Task, TResult> continuationFunction
)

Parameters

tasks
    Type: array< System.Threading.Tasks..:::Task >[]()[]
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System...:::Func<(Of (Task, TResult)>)
    The function delegate to execute when one task in the tasks array completes.

Return Value

The new continuation Task.
<table>
<thead>
<tr>
<th><strong>Exception</strong></th>
<th><strong>Condition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TAntecedentResult) (  _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult),
    ) As Task(Of TResult)

C#

public TResult ContinueWhenAny<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>, TResult> continuationFunction
)

Parameters

tasks
Type: array< System.Threading.Tasks:::Task <(Of <(TAntecedentResult)>)> >[]([]]
The array of tasks from which to continue when one task completes.

continuationFunction
Type: System:::Func <(Of <(Task<Of <(TAntecedentResult)>)>),
TResult>)>
The function delegate to execute when one task in the tasks array completes.
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task<Of <(TResult)>).
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of TResult)::.ContinueWhenAny Method (array<Task>[] [], Func(Of (Task, TResult)[]), CancellationToken)

TaskFactory(Of TResult) Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task, TResult), _
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny(  
    Task[] tasks,  
    Func<Task, TResult> continuationFunction,  
    CancellationToken cancellationToken  
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task >[]
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System...:::Func<Of (Task, TResult)>)
    The function delegate to execute when one task in the tasks array completes.

cancellationToken
    Type: System.Threading...:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.

Return Value

The new continuation Task.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td><code>System:::ArgumentNullException</code></td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td><code>System:::ArgumentException</code></td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td><code>System:::ArgumentException</code></td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td><code>System:::ObjectDisposedException</code></td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
.NET Framework Class Library
TaskFactory<(Of (TResult)>)::.::.ContinueWhenAny Method (array<Task>[] [], Func<(Of (Task, TResult)>)>, TaskContinuationOptions)
TaskFactory<(Of (TResult)>) Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:**  System.Threading.Tasks
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task, TResult), _
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny( 
    Task[] tasks, 
    Func<Task, TResult> continuationFunction, 
    TaskContinuationOptions continuationOptions
)

Parameters

tasks
    Type: array< System.Threading.Tasks.Task >[]
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System.Func<(Of <Task, TResult>)>
    The function delegate to execute when one task in the tasks array completes.

continuationOptions
    Type: System.Threading.Tasks.TaskContinuationOptions
    The TaskContinuationOptions value that controls the behavior of the created continuation Task.

Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of (T Result)<>).ContinueWhenAny(Of (T Antecedent Result)<>)
Method (array(Of (T Antecedent Result)<>), Func(Of (Task(Of (T Antecedent Result)<>), T Result)<>), CancellationToken)

TaskFactory(Of (T Result)<>).Class
See Also
Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function ContinueWhenAny(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult),
    CancellationToken),
) As Task(Of TResult)
```

#### C#

```csharp
public Task<TResult> ContinueWhenAny<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>, TResult> continuationFunction,
    CancellationToken cancellationToken
)
```

### Parameters

- **tasks**
  - Type: array< `System.Threading.Tasks:::Task`<Of (Of `TAntecedentResult`)> >[]
  - The array of tasks from which to continue when one task completes.

- **continuationFunction**
  - Type: System...::Func<Of (Of `<Task`<Of (Of `TAntecedentResult`)>)),
    TResult>)>
  - The function delegate to execute when one task in the tasks array completes.

- **cancellationToken**
  - Type: `System.Threading:::CancellationToken`
  - The `CancellationToken` that will be assigned to the new continuation task.
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task<(Of <(TResult)>)>.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System:::ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided <code>CancellationToken</code> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskFactory(Of TResult)::.ContinueWhenAny(Of TAntecedentResult>) Method (array<Task(Of TAntecedentResult>)[], Func(Of Task(Of TAntecedentResult)>, TResult>), TaskContinuationOptions)

TaskFactory(Of TResult) Class  See Also  Send Feedback

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult),
    continuationOptions As TaskContinuationOptions _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>, TResult> continuationFunction,
    TaskContinuationOptions continuationOptions
)

Parameters

tasks
    Type: array<System.Threading.Tasks::Task<TAntecedentResult>>
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System....Func<Of (Of <Task<TAntecedentResult>>),
    TResult>)>
    The function delegate to execute when one task in the tasks array
    completes.

continuationOptions
    Type: System.Threading.Tasks::TaskContinuationOptions
    The TaskContinuationOptions value that controls the behavior of the
    created continuation Task.
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task<(Of <(TResult)>)>.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskFactory(Of TResult).ContinueWhenAny Method (array(Task)[], Func(Of Func(Task, TResult), CancellationToken, TaskContinuationOptions, TaskScheduler)

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny ( _
    tasks As Task(), _
    continuationFunction As Func(Of Task, TResult), _
    cancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny(
    Task[] tasks,
    Func<Task, TResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

tasks
    Type: array<System.Threading.Tasks:::Task>[][]
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System:::Func<(Of (Task, TResult)>)
    The function delegate to execute when one task in the tasks array completes.

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.

continuationOptions
    Type: System.Threading.Tasks:::TaskContinuationOptions
The TaskContinuationOptions value that controls the behavior of the created continuation Task.

scheduler
Type: System.Threading.Tasks.TaskScheduler
The TaskScheduler that is used to schedule the created continuation Task.

Return Value

The new continuation Task.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

TaskFactory<<(Of <(TResult)>>>..ContinueWhenAny<<(Of <(TAntecedentResult)>>> Method (array<Task<<(Of <(TAntecedentResult)>>>>[][], Func<<(Of <(Task<<(Of <(TAntecedentResult)>>>), TResult)>>>, CancellationToken, TaskContinuationOptions, TaskScheduler)

Creates a continuation Task that will be started upon the completion of any Task in the provided set.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function ContinueWhenAny(Of TAntecedentResult) ( _
    tasks As Task(Of TAntecedentResult)(), _
    continuationFunction As Func(Of Task(Of TAntecedentResult),
    CancellationToken As CancellationToken, _
    continuationOptions As TaskContinuationOptions, _
    scheduler As TaskScheduler _) As Task(Of TResult)

C#

public Task<TResult> ContinueWhenAny<TAntecedentResult>(
    Task<TAntecedentResult>[] tasks,
    Func<Task<TAntecedentResult>, TResult> continuationFunction,
    CancellationToken cancellationToken,
    TaskContinuationOptions continuationOptions,
    TaskScheduler scheduler
)

Parameters

tasks
    Type: array< System.Threading.Tasks:::Task<Of
    <(TAntecedentResult)>>>[]
    The array of tasks from which to continue when one task completes.

continuationFunction
    Type: System:::Func<Of <(Task<Of <(TAntecedentResult)>>),
    TResult>>
    The function delegate to execute when one task in the tasks array completes.

cancellationToken
    Type: System.Threading:::CancellationToken
    The CancellationToken that will be assigned to the new continuation task.
continuationOptions
Type: System.Threading.Tasks:::TaskContinuationOptions
The TaskContinuationOptions value that controls the behavior of the created continuation Task.

scheduler
Type: System.Threading.Tasks:::TaskScheduler
The TaskScheduler that is used to schedule the created continuation Task<(Of<(TResult)>)>.
Type Parameters

TAntecedentResult
   The type of the result of the antecedent tasks.

Return Value

The new continuation Task<Of<TResult>>.
Remarks

The NotOn* and OnlyOn* TaskContinuationOptions, which constrain for which TaskStatus states a continuation will be executed, are illegal with ContinueWhenAny.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the continuationFunction argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System..::.OutOfBoundsException</td>
<td>The exception that is thrown when the tasks array contains a null value.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the tasks array is empty.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the continuationOptions argument specifies an invalid TaskContinuationOptions value.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The exception that is thrown when one of the elements in the tasks array has been disposed.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of (TResult)> ) Class
ContinueWhenAny Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  □ C#
□ Include Protected Members
□ Include Inherited Members
.NET Framework Class Library
TaskFactory<Of <(TResult)>).::.FromAsync Method
TaskFactory<Of <(TResult)>>.Class  See Also  Send Feedback
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromAsync(IAsyncResult, Func&lt;Of (IAsyncResult, TResult)&gt;)</td>
<td>Creates a Task that executes an end method function when a specified IAsyncResult completes.</td>
</tr>
<tr>
<td>FromAsync(Func&lt;Of (AsyncCallback, Object, IAsyncResult)&gt;, Func&lt;Of (IAsyncResult, TResult)&gt;, Object)</td>
<td>Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.</td>
</tr>
<tr>
<td>FromAsync(IAsyncResult, Func&lt;Of (IAsyncResult, TResult)&gt;, TaskCreationOptions)</td>
<td>Creates a Task that executes an end method function when a specified IAsyncResult completes.</td>
</tr>
<tr>
<td>FromAsync(Func&lt;Of (AsyncCallback, Object, IAsyncResult)&gt;, Func&lt;Of (IAsyncResult, TResult)&gt;, Object, TaskCreationOptions)</td>
<td>Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.</td>
</tr>
<tr>
<td>FromAsync&lt;Of (TArg1)&gt;(Func&lt;Of (TArg1, AsyncCallback, Object, IAsyncResult)&gt;, Func&lt;Of (IAsyncResult, TResult)&gt;, TArg1, Object)</td>
<td>Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.</td>
</tr>
<tr>
<td>FromAsync(IAsyncResult, Func&lt;Of (IAsyncResult, TResult)&gt;, TaskCreationOptions, TaskScheduler)</td>
<td>Creates a Task that executes an end method function when a specified</td>
</tr>
</tbody>
</table>
FromAsync(Of <TArg1>)(Func(Of <TArg1, AsyncCallback, Object, IAsyncResult>), Func(Of <IAsyncResult, TResult>), TArg1, Object, TaskCreationOptions)

Starts a new task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync(Of <TArg1, TArg2>)(Func(Of <TArg1, TArg2, AsyncCallback, Object, IAsyncResult>), Func(Of <IAsyncResult, TResult>), TArg1, TArg2, Object)

Starts a new task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync(Of <TArg1, TArg2, TArg3>)(Func(Of <TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult>), Func(Of <IAsyncResult, TResult>), TArg1, TArg2, TArg3, Object)

Starts a new task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync(Of <TArg1, TArg2>(Func(Of <TArg1, TArg2, AsyncCallback, Object, IAsyncResult>), Func(Of <IAsyncResult, TResult>), TArg1, TArg2, Object, TaskCreationOptions)

Starts a new task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

FromAsync(Of <TArg1, TArg2, TArg3>)(Func(Of <TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult>), Func(Of <IAsyncResult, TResult>), TArg1, TArg2, TArg3, Object, TaskCreationOptions)

Starts a new task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

IAsyncResult completes.

Starts a new task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.
See Also

TaskFactory<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of TResult).FromAsync Method (IAsyncResult, Func(Of (IAsyncResult, TResult)))

TaskFactory(Of TResult) Class  See Also  Send Feedback

Creates a Task that executes an end method function when a specified IAsyncResult completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync ( _
    asyncResult As IAsyncResult, _
    endMethod As Func(Of IAsyncResult, TResult) _
) As Task(Of TResult)

C#

public Task&lt;TResult&gt; FromAsync(
    IAsyncResult asyncResult,
    Func&lt;IAsyncResult, TResult&gt; endMethod
)

Parameters

asyncResult
    Type: System:::IAsyncResult
    The IAsyncResult whose completion should trigger the processing of the endMethod.

endMethod
    Type: System:::Func&lt;(Of &lt;IAsyncResult, TResult&gt;)&gt;
    The function delegate that processes the completed asyncResult.

Return Value

A Task that represents the asynchronous operation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::..ArgumentNullException</td>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>System...::..ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)>:::.FromAsync Method (Func<(Of <(AsyncCallback, Object, IAsyncResult)>)>, Func<(Of <(IAsyncResult, TResult)>)>, Object)

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
### Syntax

#### Visual Basic (Declaration)

```vbnet
Public Function FromAsync ( _
    beginMethod As Func(Of AsyncCallback, Object, IAsyncResult),
    endMethod As Func(Of IAsyncResult, TResult), _
    state As Object _
) As Task(Of TResult)
```

#### C#

```csharp
public Task<TResult> FromAsync( 
    Func<AsyncCallback, Object, IAsyncResult> beginMethod, 
    Func<IAsyncResult, TResult> endMethod, 
    Object state
)
```

### Parameters

**beginMethod**
- Type: System:::Func(Of ((AsyncCallback, Object, IAsyncResult))>
- The delegate that begins the asynchronous operation.

**endMethod**
- Type: System:::Func(Of ((IAsyncResult, TResult))>
- The delegate that ends the asynchronous operation.

**state**
- Type: System:::Object
- An object containing data to be used by the beginMethod delegate.

### Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System...::.ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System...::.ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of (TResult)>)(Of (TResult)>)<-Class FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)>..::..FromAsync Method (IAsyncResult, Func<(Of <(IAsyncResult, TResult)>)>), TaskCreationOptions)

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates a Task that executes an end method function when a specified IAsyncResult completes.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync ( _
    asyncResult As IAsyncResult, _
    endMethod As Func(Of IAsyncResult, TResult), _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync(
    IAsyncResult asyncResult,
    Func<IAsyncResult, TResult> endMethod,
    TaskCreationOptions creationOptions
)

Parameters

asyncResult
    Type: System.IAsyncResult
    The IAsyncResult whose completion should trigger the processing of the endMethod.

endMethod
    Type: System.Func(Of IAsyncResult, TResult)
    The function delegate that processes the completed asyncResult.

creationOptions
    Type: System.Threading.Tasks.TaskCreationOptions
    The TaskCreationOptions value that controls the behavior of the created Task.

Return Value

A Task that represents the asynchronous operation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)>...:.FromAsync Method (Func<(Of <(AsyncCallback, Object, IAsyncResult)>)), Func<(Of <(IAsyncResult, TResult)>)), Object, TaskCreationOptions)

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync ( _
    beginMethod As Func(Of AsyncCallback, Object, IAsyncResult), _
    endMethod As Func(Of IAsyncResult, TResult), _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync(
    Func<AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
    Type: System:::Func<(Of <(AsyncCallback, Object, IAsyncResult)>)>  
The delegate that begins the asynchronous operation.

endMethod
    Type: System:::Func<(Of <(IAsyncResult, TResult)>)>  
The delegate that ends the asynchronous operation.

state
    Type: System:::Object  
    An object containing data to be used by the beginMethod delegate.

creationOptions
    Type: System.Threading.Tasks:::TaskCreationOptions  
The TaskCreationOptions value that controls the behavior of the created Task.
**Return Value**

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)::.FromAsync<(Of <(TArg1)>)> Method (Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>)>, Func<(Of <(IAsyncResult, TResult)>)>, TArg1, Object)  

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1) ( _
    beginMethod As Func(Of TArg1, AsyncCallback, Object, IAsyncResult, TResult), _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    state As Object _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1>(
    Func<TArg1, AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    Object state
)

Parameters

beginMethod
    Type: System:::Func<(Of <(TArg1, AsyncCallback, Object, IAsyncResult)>))
    The delegate that begins the asynchronous operation.

downloadMethod
    Type: System:::Func<(Of <(IAsyncResult, TResult)>))
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.

state
    Type: System:::Object
    An object containing data to be used by the beginMethod delegate.
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)>..::..FromAsync Method (IAsyncResult, Func<(Of <(IAsyncResult, TResult)>)), TaskCreationOptions, TaskScheduler)

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates a Task that executes an end method function when a specified IAsyncResult completes.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

```vbnet
Public Function FromAsync (_
    asyncResult As IAsyncResult, _
    endMethod As Func(Of IAsyncResult, TResult), _
    creationOptions As TaskCreationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)
```

C#

```csharp
public Task<TResult> FromAsync(
    IAsyncResult asyncResult,
    Func<IAsyncResult, TResult> endMethod,
    TaskCreationOptions creationOptions,
    TaskScheduler scheduler
)
```

Parameters

asyncResult
Type: System:::IAsyncResult
The IAsyncResult whose completion should trigger the processing of the endMethod.

endMethod
Type: System:::Func(Of <(IAsyncResult, TResult)>)
The function delegate that processes the completed asyncResult.

creationOptions
Type: System.Threading.Tasks:::TaskCreationOptions
The TaskCreationOptions value that controls the behavior of the created Task.

scheduler
Type: System.Threading.Tasks:::TaskScheduler
The `TaskScheduler` that is used to schedule the task that executes the end method.

**Return Value**

A Task that represents the asynchronous operation.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..:::ArgumentNullException</td>
<td>The exception that is thrown when the asyncResult argument is null.</td>
</tr>
<tr>
<td>System..:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System..:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System..:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library
TaskFactory(Of (TResult))::FromAsync(Of (TArg1)) Method
(Func(Of (TArg1, AsyncCallback, Object, IAsyncResult)), Func(Of (IAsyncResult, TResult)), TArg1, Object, TaskCreationOptions)

TaskFactory(Of (TResult)) Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Function FromAsync(Of TArg1) ( _
    beginMethod As Func(Of TArg1, AsyncCallback, Object, IAsyncResult, TResult), _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)
```

**C#**

```csharp
public Task<TResult> FromAsync<TArg1>(
    Func<TArg1, AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    Object state,
    TaskCreationOptions creationOptions
)
```

**Parameters**

**beginMethod**
- Type: System..::.Func(Of (Of TArg1, AsyncCallback, Object, IAsyncResult)>)
- The delegate that begins the asynchronous operation.

**endMethod**
- Type: System..::.Func(Of (IAsyncResult, TResult)>)
- The delegate that ends the asynchronous operation.

**arg1**
- Type: TArg1
- The first argument passed to the beginMethod delegate.

**state**
- Type: System..::.Object
An object containing data to be used by the beginMethod delegate.

creationOptions
Type: System.Threading.Tasks.TaskCreationOptions
The TaskCreationOptions value that controls the behavior of the created Task.
Type Parameters

TArg1
    The type of the first argument passed to the beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library
TaskFactory(Of (TResult>)).:::FromAsync(Of ((TArg1, TArg2)>)
Method (Func(Of ((TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>), Func(Of ((IAsyncResult, TResult)>), TArg1, TArg2, Object)

TaskFactory(Of (Result>)> Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2) ( _
    beginMethod As Func(Of TArg1, TArg2, AsyncCallback, Object, _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    state As Object _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TArg2>(
    Func<TArg1, TArg2, AsyncCallback, Object, IAsyncResult> begin,
    Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    Object state
)

Parameters

beginMethod
    Type: System....:Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)
    The delegate that begins the asynchronous operation.

eндMethod
    Type: System....:Func<(Of <(IAasyncResult, TResult)>))
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
    The first argument passed to the beginMethod delegate.

arg2
    Type: TArg2
The second argument passed to the beginMethod delegate.

state
Type: System::Object
An object containing data to be used by the beginMethod delegate.
**Type Parameters**

TArg1
The type of the first argument passed to the `beginMethod` delegate.

TArg2
The type of the second argument passed to `beginMethod` delegate.

**Return Value**

The created `Task` that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>beginMethod</code> argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>endMethod</code> argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
.NET Framework Class Library

TaskFactory<(Of <(TResult)>)>..::.FromAsync<(Of <(TArg1, TArg2, TArg3)>)> Method (Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)), Func<(Of <(IAsyncResult, TResult)>)), TArg1, TArg2, TArg3, Object)

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

**Namespace:**  `System.Threading.Tasks`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2, TArg3) (_
    beginMethod As Func(Of TArg1, TArg2, TArg3, AsyncCallback, _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    arg3 As TArg3, _
    state As Object _
  ) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TArg2, TArg3>(
    Func<TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult Func<TAsyncResult, TResult>> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    TArg3 arg3,
    Object state
  )

Parameters

beginMethod
Type: System...:: Func(Of (TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)
The delegate that begins the asynchronous operation.

dendMethod
Type: System....Func(Of (IAsyncResult, TResult)>)
The delegate that ends the asynchronous operation.

arg1
Type: TArg1
The first argument passed to the beginMethod delegate.
arg2
  Type: TArg2
  The second argument passed to the beginMethod delegate.

arg3
  Type: TArg3
  The third argument passed to the beginMethod delegate.

state
  Type: System::Object
  An object containing data to be used by the beginMethod delegate.
Type Parameters

TArg1
- The type of the first argument passed to the BeginMethod delegate.

TArg2
- The type of the second argument passed to BeginMethod delegate.

TArg3
- The type of the third argument passed to BeginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>beginMethod</code> argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the <code>endMethod</code> argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library
TaskFactory<(Of <(TResult)>).:.:.FromAsync<(Of <(TArg1, TArg2)>)> Method (Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)>, Func<(Of <(IAsyncResult, TResult)>)>, TArg1, TArg2, Object, TaskCreationOptions)

TaskFactory<(Of <(TResult)>)> Class

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2) ( _
    beginMethod As Func(Of TArg1, TArg2, AsyncCallback, Object, _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TArg2>(
    Func<TArg1, TArg2, AsyncCallback, Object, IAsyncResult> beginMethod,
    Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
    Type: System....Func<(Of <(TArg1, TArg2, AsyncCallback, Object, IAsyncResult)>)>  
    The delegate that begins the asynchronous operation.

d EndMethod
    Type: System....Func<(Of <(IAsyncResult, TResult)>)>  
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1  
    The first argument passed to the beginMethod delegate.
arg2
Type: TArg2
The second argument passed to the beginMethod delegate.

state
Type: System::Object
An object containing data to be used by the beginMethod delegate.

creationOptions
Type: System.Threading.Tasks::TaskCreationOptions
The TaskCreationOptions value that controls the behavior of the created Task.
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.
TArg2
The type of the second argument passed to beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td>System..::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)>::FromAsync<(Of <(TArg1, TArg2, TArg3)>)> Method (Func_<(Of <(TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult)>)>, Func<(Of <IAsyncResult, TResult>)>, TArg1, TArg2, TArg3, Object, TaskCreationOptions)

Creates a Task that represents a pair of begin and end methods that conform to the Asynchronous Programming Model pattern.

**Namespace**: System.Threading.Tasks
**Assembly**: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function FromAsync(Of TArg1, TArg2, TArg3) ( _
    beginMethod As Func(Of TArg1, TArg2, TArg3, AsyncCallback, _
    endMethod As Func(Of IAsyncResult, TResult), _
    arg1 As TArg1, _
    arg2 As TArg2, _
    arg3 As TArg3, _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> FromAsync<TArg1, TArg2, TArg3>(
    Func<TArg1, TArg2, TArg3, AsyncCallback, Object, IAsyncResult, Func<IAsyncResult, TResult> endMethod,
    TArg1 arg1,
    TArg2 arg2,
    TArg3 arg3,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

beginMethod
    Type: System:::Func<(Of (TArg1, TArg2, TArg3, AsyncCallback,
     Object, IAsyncResult)>)
    The delegate that begins the asynchronous operation.

endMethod
    Type: System:::Func<(Of (IAsyncResult, TResult)>)
    The delegate that ends the asynchronous operation.

arg1
    Type: TArg1
The first argument passed to the `beginMethod` delegate.

*arg2*
Type: `TArg2`
The second argument passed to the `beginMethod` delegate.

*arg3*
Type: `TArg3`
The third argument passed to the `beginMethod` delegate.

*state*
Type: `System::Object`
An object containing data to be used by the `beginMethod` delegate.

*creationOptions*
Type: `System::Threading::Tasks::TaskCreationOptions`
The `TaskCreationOptions` value that controls the behavior of the created `Task`. 
Type Parameters

TArg1
The type of the first argument passed to the beginMethod delegate.

TArg2
The type of the second argument passed to beginMethod delegate.

TArg3
The type of the third argument passed to beginMethod delegate.

Return Value

The created Task that represents the asynchronous operation.
Remarks

This method throws any exceptions thrown by the beginMethod.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the beginMethod argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the endMethod argument is null.</td>
</tr>
<tr>
<td></td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
FromAsync Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
TaskFactory(Of (TResult)>).StartNew Method
TaskFactory(Of (TResult)>) Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>StartNew(Func&lt;Of &lt;(TResult)&gt;&gt;)</code></td>
<td>Creates and starts a Task&lt;Of &lt;(TResult)&gt;&gt;.</td>
</tr>
<tr>
<td><code>StartNew(Func&lt;Of &lt;(TResult)&gt;&gt;, CancellationToken)</code></td>
<td>Creates and starts a Task&lt;Of &lt;(TResult)&gt;&gt;.</td>
</tr>
<tr>
<td><code>StartNew(Func&lt;Of &lt;(TResult)&gt;&gt;, TaskCreationOptions)</code></td>
<td>Creates and starts a Task&lt;Of &lt;(TResult)&gt;&gt;.</td>
</tr>
<tr>
<td><code>StartNew(Func&lt;Of &lt;(Object, TResult)&gt;&gt;, Object)</code></td>
<td>Creates and starts a Task&lt;Of &lt;(Object, TResult)&gt;&gt;.</td>
</tr>
<tr>
<td><code>StartNew(Func&lt;Of &lt;(Object, TResult)&gt;&gt;, Object, CancellationToken)</code></td>
<td>Creates and starts a Task&lt;Of &lt;(Object, TResult)&gt;&gt;.</td>
</tr>
<tr>
<td><code>StartNew(Func&lt;Of &lt;(Object, TResult)&gt;&gt;, Object, TaskCreationOptions)</code></td>
<td>Creates and starts a Task&lt;Of &lt;(Object, TResult)&gt;&gt;.</td>
</tr>
<tr>
<td><code>StartNew(Func&lt;Of &lt;(TResult)&gt;&gt;, CancellationToken, TaskCreationOptions, TaskScheduler)</code></td>
<td>Creates and starts a Task&lt;Of &lt;(TResult)&gt;&gt;.</td>
</tr>
<tr>
<td><code>StartNew(Func&lt;Of &lt;(Object, TResult)&gt;&gt;, Object, CancellationToken, TaskCreationOptions, TaskScheduler)</code></td>
<td>Creates and starts a Task&lt;Of &lt;(Object, TResult)&gt;&gt;.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Creates and starts a Task<Of <(TResult)>).
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    function As Func(Of TResult) _
) As Task(Of TResult)

C#

public Task<TResult> StartNew(
    Func<TResult> function
)

Parameters

function
    Type: System...::Func<Of <(TResult)>>
    A function delegate that returns the future result to be available through the Task<Of <(TResult)>>.

Return Value

The started Task<Of <(TResult)>>.
Remarks

Calling StartNew is functionally equivalent to creating a `Task<Of TResult>)` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(<TResult>)> Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
C#
.NET Framework Class Library
TaskFactory<(Of (TResult)>):::.StartNew Method (Func<(Of (TResult)>)>, CancellationToken)
TaskFactory<(Of (TResult)>) Class  See Also  Send Feedback

Creates and starts a Task<(Of (TResult)>)>.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    function As Func(Of TResult), _
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> StartNew(
    Func<TResult> function,
    CancellationToken cancellationToken
)

Parameters

function
    Type: System...:Func(Of (TResult)>)
    A function delegate that returns the future result to be available through the
    Task<(Of (TResult)>).

cancellationToken
    Type: System.Threading...:CancellationToken
    The CancellationToken that will be assigned to the new task.

Return Value

The started Task<(Of (TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<TResult>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of (TResult)>)> Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of TResult)::.StartNew Method (Func(Of TResult), TaskCreationOptions)

TaskFactory(Of TResult) Class  See Also  Send Feedback

Creates and starts a Task(Of TResult).

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    function As Func(Of TResult), _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> StartNew(
    Func<TResult> function,
    TaskCreationOptions creationOptions
)

Parameters

function
    Type: System::Func<(Of (TResult)>)
    A function delegate that returns the future result to be available through the Task<(Of (TResult)>).

creationOptions
    Type: System.Threading.Tasks::TaskCreationOptions
    A TaskCreationOptions value that controls the behavior of the created Task<(Of (TResult)>).

Return Value

The started Task<(Of (TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<TResult>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArgumentNullException</code></td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td><code>ArgumentOutOfRangeException</code></td>
<td>The exception that is thrown when the <code>creationOptions</code> argument specifies an invalid <code>TaskCreationOptions</code> value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskFactory(Of (TResult))::.StartNew Method (Func(Of (Object, TResult)), Object)

TaskFactory(Of (TResult)) Class  See Also  Send Feedback

Creates and starts a Task(Of (TResult)).

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

**Visual Basic (Declaration)**

Public Function StartNew ( _
    function As Func(Of Object, TResult), _
    state As Object _
) As Task(Of TResult)

**C#**

public Task<TResult> StartNew(
    Func<Object, TResult> function,
    Object state
)

**Parameters**

function

Type: System:::Func<(Of (Object, TResult)>)
A function delegate that returns the future result to be available through the Task<(Of (TResult)>).

state

Type: System:::Object
An object containing data to be used by the function delegate.

**Return Value**

The started Task<(Of (TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<(<TResult>)>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of TResult)::.StartNew Method (Func(Of (Object, TResult)<>), Object, CancellationToken)

TaskFactory(Of TResult) Class  See Also  Send Feedback

Creates and starts a Task(Of TResult>).

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    function As Func(Of Object, TResult), _
    state As Object, _
    cancellationToken As CancellationToken _
) As Task(Of TResult)

C#

public Task<TResult> StartNew(
    Func<Object, TResult> function,
    Object state,
    CancellationToken cancellationToken
)

Parameters

function
    Type: System::.::Func<(Of <(Object, TResult)>)
    A function delegate that returns the future result to be available through the Task<(Of <(TResult)>).

state
    Type: System::.::Object
    An object containing data to be used by the function delegate.

cancellationToken
    Type: System.Threading::.::CancellationToken
    The CancellationToken that will be assigned to the new task.

Return Value

The started Task<(Of <(TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<(<TResult>)>` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System..::.ObjectDisposedException</td>
<td>The provided <a href="#">CancellationToken</a> has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)>...::StartNew Method (Func<(Of <(Object, TResult)>>), Object, TaskCreationOptions)

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates and starts a Task<(Of <(TResult)>)>.

Namespace:  System.Threading.Tasks
Assembly:  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    function As Func(Of Object, TResult), _
    state As Object, _
    creationOptions As TaskCreationOptions _
) As Task(Of TResult)

C#

public Task<TResult> StartNew(
    Func<Object, TResult> function,
    Object state,
    TaskCreationOptions creationOptions
)

Parameters

function
    Type: System...::Func<(Of <(Object, TResult)>)
    A function delegate that returns the future result to be available through the Task<(Of <(TResult)>).

state
    Type: System...::Object
    An object containing data to be used by the function delegate.

creationOptions
    Type: System.Threading.Tasks...::TaskCreationOptions
    A TaskCreationOptions value that controls the behavior of the created Task<(Of <(TResult)>).

Return Value

The started Task<(Of <(TResult)>).
Remarks

Calling StartNew is functionally equivalent to creating a `Task<((TResult)>)` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
# Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory<(Of <(TResult)>)>::StartNew Method (Func<(Of <(TResult)>)>, CancellationToken, TaskCreationOptions, TaskScheduler)

TaskFactory<(Of <(TResult)>)> Class  See Also  Send Feedback

Creates and starts a Task<(Of <(TResult)>)>.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( function As Func(Of TResult), cancellationToken As CancellationToken, creationOptions As TaskCreationOptions, scheduler As TaskScheduler ) As Task(Of TResult)

C#

public Task<TResult> StartNew( Func<TResult> function, CancellationToken cancellationToken, TaskCreationOptions creationOptions, TaskScheduler scheduler )

Parameters

function
Type: System...:::Func(Of ((TResult)>))
A function delegate that returns the future result to be available through the Task(Of ((TResult)>)).

cancellationToken
Type: System.Threading...:::CancellationToken
The CancellationToken that will be assigned to the new task.

creationOptions
Type: System.Threading.Tasks...:::TaskCreationOptions
A TaskCreationOptions value that controls the behavior of the created Task(Of ((TResult)>)).

scheduler
Type: System.Threading.Tasks...:::TaskScheduler
The TaskScheduler that is used to schedule the created Task<TResult>.

Return Value

The started Task<Of (TResult)>.
Remarks

Calling StartNew is functionally equivalent to creating a `Task(Of TResult)` using one of its constructors and then calling `Start` to schedule it for execution. However, unless creation and scheduling must be separated, `StartNew` is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System::.ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System::.ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System::.ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
<tr>
<td>System::.ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult>) Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskFactory(Of TResult)::.StartNew Method (Func(Of (Object, TResult)), Object, CancellationToken, TaskCreationOptions, TaskScheduler)

**TaskFactory(Of TResult) Class**  
See Also  
Send Feedback

Creates and starts a Task(Of TResult).  

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Function StartNew ( _
    function As Func(Of Object, TResult), _
    state As Object, _
    cancellationToken As CancellationToken, _
    creationOptions As TaskCreationOptions, _
    scheduler As TaskScheduler _
) As Task(Of TResult)

C#

public Task<TResult> StartNew(
    Func<
        Object,
        TResult>
    function,
    Object state,
    CancellationToken cancellationToken,
    TaskCreationOptions creationOptions,
    TaskScheduler scheduler
)

Parameters

function
    Type: System...:Func<
        Of
        <
        Object,
        TResult>
    )>
    A function delegate that returns the future result to be available through the
    Task<
        Of
        <
        TResult>
    >).

state
    Type: System...:Object
    An object containing data to be used by the function delegate.

cancellationToken
    Type: System.Threading...:CancellationToken
    The CancellationToken that will be assigned to the new task.

creationOptions
    Type: System.Threading.Tasks...:TaskCreationOptions
A TaskCreationOptions value that controls the behavior of the created Task<(Of <(TResult)>)).

scheduler
  Type: System.Threading.Tasks::__::TaskScheduler
  The TaskScheduler that is used to schedule the created Task{TResult}.

Return Value

The started Task<(Of <(TResult)>)).
Remarks

Calling StartNew is functionally equivalent to creating a Task<TResult> using one of its constructors and then calling Start to schedule it for execution. However, unless creation and scheduling must be separated, StartNew is the recommended approach for both simplicity and performance.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the function argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentNullException</td>
<td>The exception that is thrown when the scheduler argument is null.</td>
</tr>
<tr>
<td>System:::ArgumentOutOfRangeException</td>
<td>The exception that is thrown when the creationOptions argument specifies an invalid TaskCreationOptions value.</td>
</tr>
<tr>
<td>System:::ObjectDisposedException</td>
<td>The provided CancellationToken has already been disposed.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory<(Of <(TResult)>)> Class
StartNew Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `TaskFactory(Of TResult)`) type exposes the following members.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancellationToken</td>
<td>Gets the default <a href="#">CancellationToken</a> of this TaskFactory.</td>
</tr>
<tr>
<td>ContinuationOptions</td>
<td>Gets the <a href="#">TaskContinuationOptions</a> value of this TaskFactory{TResult}.</td>
</tr>
<tr>
<td>CreationOptions</td>
<td>Gets the <a href="#">TaskCreationOptions</a> value of this TaskFactory{TResult}.</td>
</tr>
<tr>
<td>Scheduler</td>
<td>Gets the <a href="#">TaskScheduler</a> of this TaskFactory{TResult}.</td>
</tr>
</tbody>
</table>
See Also

TaskFactory(Of TResult) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the default `CancellationToken` of this `TaskFactory`.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property CancellationToken As CancellationToken

C#

public CancellationToken CancellationToken { get; }
Remarks

This property returns the default CancellationToken that will be assigned to all tasks created by this factory unless another CancellationToken value is explicitly specified during the call to the factory methods.
See Also

TaskFactory<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the `TaskContinuationOptions` value of this `TaskFactory{TResult}`.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property ContinuationOptions As TaskContinuationOptions

C#

public TaskContinuationOptions ContinuationOptions { get; }
Remarks

This property returns the default continuation options for this factory. They will be used to create all continuation tasks unless other options are explicitly specified during calls to this factory's methods.
See Also

TaskFactory<T>(Of <(TResult)>>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the `TaskCreationOptions` value of this `TaskFactory{TResult}`.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property CreationOptions As TaskCreationOptions

C#

public TaskCreationOptions CreationOptions { get; }
Remarks

This property returns the default creation options for this factory. They will be used to create all tasks unless other options are explicitly specified during calls to this factory's methods.
See Also

TaskFactory(Of TResult>) Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the `TaskScheduler` of this `TaskFactory{TResult}`.

**Namespace:** [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Scheduler As TaskScheduler

C#

public TaskScheduler Scheduler { get; }
Remarks

This property returns the default scheduler for this factory. It will be used to schedule all tasks unless another scheduler is explicitly specified during calls to this factory's methods. If null, TaskScheduler.Current will be used.
See Also

TaskFactory<(Of <(TResult)>)> Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Represents an abstract scheduler for tasks.

**Namespace:**  [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<ServiceProtectionAttribute(SecurityAction.InheritanceDemand, Unrestricted=
ServiceProtectionAttribute(SecurityAction.LinkDemand, Synchronization=
ExternalThreading := True)>
Public MustInherit Class TaskScheduler

C#

[ServiceProtectionAttribute(SecurityAction.InheritanceDemand, Unrestricted=
[ServiceProtectionAttribute(SecurityAction.LinkDemand, Synchronization=
ExternalThreading = true)]
public abstract class TaskScheduler
Remarks

TaskScheduler acts as the extension point for all pluggable scheduling logic. This includes mechanisms such as how to schedule a task for execution, and how scheduled tasks should be exposed to debuggers.

All members of the abstract TaskScheduler type are thread-safe and may be used from multiple threads concurrently.
Inheritance Hierarchy

System..::.Object
System.Threading.Tasks..::.TaskScheduler
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes the TaskScheduler.

**Namespace:** System.Threading.Tasks

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Sub New

C#

protected TaskScheduler()
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `TaskScheduler` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.) Frees all resources associated with this scheduler.</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Overides Object...::Finalize())</td>
</tr>
<tr>
<td>FromCurrentSynchronizationContext</td>
<td>Creates a TaskScheduler associated with the current SynchronizationContext.</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetScheduledTasks</td>
<td>Generates an enumerable of Task instances currently queued to the scheduler waiting to be executed.</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>QueueTask</td>
<td>Queues a Task to the scheduler.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>TryDequeue</td>
<td>Attempts to dequeue a Task that was previously queued to this scheduler.</td>
</tr>
<tr>
<td>TryExecuteTask</td>
<td>Attempts to execute the provided Task on this scheduler.</td>
</tr>
<tr>
<td>TryExecuteTaskInline</td>
<td>Determines whether the provided Task can be executed synchronously in this call, and if it can, executes it.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Frees all resources associated with this scheduler.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Protected Overrides Sub Finalize

C#
protected override void Finalize()
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#

.NET Framework Class Library

TaskScheduler...:FromCurrentSynchronizationContext Method

**TaskScheduler Class**  See Also  Send Feedback

Creates a **TaskScheduler** associated with the current SynchronizationContext.

**Namespace:**  **System.Threading.Tasks**  
**Assembly:**  **System.Threading** (in System.Threading.dll)**
Syntax

Visual Basic (Declaration)

Public Shared Function FromCurrentSynchronizationContext As TaskScheduler

C#

public static TaskScheduler FromCurrentSynchronizationContext()

Return Value

A TaskScheduler associated with the current SynchronizationContext, as determined by SynchronizationContext.Current.
Remarks

All Task instances queued to the returned scheduler will be executed through a call to the Post method on that context.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System..::.InvalidOperationException</code></td>
<td>The current <code>SynchronizationContext</code> may not be used as a <code>TaskScheduler</code>.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Generates an enumerable of Task instances currently queued to the scheduler waiting to be executed.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected MustOverride Function GetScheduledTasks As IEnumerable(Of C#

protected abstract IEnumerable<

Return Value

An enumerable that allows traversal of tasks currently queued to this scheduler.
Remarks

A class derived from TaskScheduler implements this method in order to support integration with debuggers. This method will only be invoked by the .NET Framework when the debugger requests access to the data. The enumerable returned will be traversed by debugging utilities to access the tasks currently queued to this scheduler, enabling the debugger to provide a representation of this information in the user interface.

It is important to note that, when this method is called, all other threads in the process will be frozen. Therefore, it's important to avoid synchronization with other threads that may lead to blocking. If synchronization is necessary, the method should prefer to throw a NotSupportedException than to block, which could cause a debugger to experience delays. Additionally, this method and the enumerable returned must not modify any globally visible state.

The returned enumerable should never be null. If there are currently no queued tasks, an empty enumerable should be returned instead.

For developers implementing a custom debugger, this method shouldn't be called directly, but rather this functionality should be accessed through the internal wrapper method GetScheduledTasksForDebugger: internal Task[] GetScheduledTasksForDebugger(). This method returns an array of tasks, rather than an enumerable. In order to retrieve a list of active schedulers, a debugger may use another internal method: internal static TaskScheduler[] GetTaskSchedulersForDebugger(). This static method returns an array of all active TaskScheduler instances. GetScheduledTasksForDebugger then may be used on each of these scheduler instances to retrieve the list of scheduled tasks for each.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System....:NotSupportedException</td>
<td>This scheduler is unable to generate a list of queued tasks at this time.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Queues a Task to the scheduler.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Friend MustOverride Sub QueueTask ( _
  task As Task _
)

C#

protected internal abstract void QueueTask(_
  Task task
)

Parameters

task
  Type: System.Threading.Tasks.Task
  The Task to be queued.
Remarks

A class derived from TaskScheduler implements this method to accept tasks being scheduled on the scheduler. A typical implementation would store the task in an internal data structure, which would be serviced by threads that would execute those tasks at some time in the future.

This method is only meant to be called by the .NET Framework and should not be called directly by the derived class. This is necessary for maintaining the consistency of the system.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System..::.ArgumentNullException</td>
<td>The task argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Attempts to dequeue a Task that was previously queued to this scheduler.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Friend Overridable Function TryDequeue ( _
    task As Task _
) As Boolean

C#

protected internal virtual bool TryDequeue(
    Task task
)

Parameters

task
    Type: System.Threading.Tasks:::Task
    The Task to be dequeued.

Return Value

A Boolean denoting whether the task argument was successfully dequeued.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentException</td>
<td>The task argument is null.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Attempts to execute the provided Task on this scheduler.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Function TryExecuteTask ( _
    task As Task _
) As Boolean

C#

protected bool TryExecuteTask(
    Task task
)

Parameters

task
    Type: System.Threading.Tasks.Task
    A Task object to be executed.

Return Value

A Boolean that is true if task was successfully executed, false if it was not. A common reason for execution failure is that the task had previously been executed or is in the process of being executed by another thread.
Remarks

Scheduler implementations are provided with Task instances to be executed through either the QueueTask(Task) method or the TryExecuteTaskInline(Task, Boolean) method. When the scheduler deems it appropriate to run the provided task, TryExecuteTask(Task) should be used to do so. TryExecuteTask handles all aspects of executing a task, including action invocation, exception handling, state management, and lifecycle control.

TryExecuteTask(Task) must only be used for tasks provided to this scheduler by the .NET Framework infrastructure. It should not be used to execute arbitrary tasks obtained through custom mechanisms.
<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.::.InvalidOperationException</td>
<td>The task is not associated with this scheduler.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Determines whether the provided `Task` can be executed synchronously in this call, and if it can, executes it.

**Namespace:**  `System.Threading.Tasks`  
**Assembly:**  `System.Threading (in System.Threading.dll)`
Syntax

Visual Basic (Declaration)

Protected MustOverride Function TryExecuteTaskInline ( _
    task As Task, _
    taskWasPreviouslyQueued As Boolean _
) As Boolean

C#

protected abstract bool TryExecuteTaskInline(
    Task task,
    bool taskWasPreviouslyQueued
)

Parameters

task
    Type: System.Threading.Tasks.Task
    The Task to be executed.

taskWasPreviouslyQueued
    Type: System.Boolean
    A Boolean denoting whether or not task has previously been queued. If this parameter is True, then the task may have been previously queued (scheduled); if False, then the task is known not to have been queued, and this call is being made in order to execute the task inline without queueing it.

Return Value

A Boolean value indicating whether the task was executed inline.
Remarks

A class derived from `TaskScheduler` implements this function to support inline execution of a task on a thread that initiates a wait on that task object. Inline execution is optional, and the request may be rejected by returning false. However, better scalability typically results the more tasks that can be inlined, and in fact a scheduler that inlines too little may be prone to deadlocks. A proper implementation should ensure that a request executing under the policies guaranteed by the scheduler can successfully inline. For example, if a scheduler uses a dedicated thread to execute tasks, any inlining requests from that thread should succeed.

If a scheduler decides to perform the inline execution, it should do so by calling to the base TaskScheduler's `TryExecuteTask` method with the provided task object, propagating the return value. It may also be appropriate for the scheduler to remove an inlined task from its internal data structures if it decides to honor the inlining request. Note, however, that under some circumstances a scheduler may be asked to inline a task that was not previously provided to it with the `QueueTask(Task)` method.

The derived scheduler is responsible for making sure that the calling thread is suitable for executing the given task as far as its own scheduling and execution policies are concerned.
## Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System:::ArgumentNullException</td>
<td>The task argument is null.</td>
</tr>
<tr>
<td>System:::InvalidOperationException</td>
<td>The task was already executed.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `TaskScheduler` type exposes the following members.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>Gets the <a href="#">TaskScheduler</a> associated with the currently executing task.</td>
</tr>
<tr>
<td>Default</td>
<td>Gets the default <a href="#">TaskScheduler</a> instance.</td>
</tr>
<tr>
<td>Id</td>
<td>Gets the unique ID for this <a href="#">TaskScheduler</a>.</td>
</tr>
<tr>
<td>MaximumConcurrencyLevel</td>
<td>Indicates the maximum concurrency level this <a href="#">TaskScheduler</a> is able to support.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  ▪  C#
.NET Framework Class Library
TaskScheduler...:..Current Property
TaskScheduler Class  See Also  Send Feedback

Gets the TaskScheduler associated with the currently executing task.

**Namespace:**  System.Threading.Tasks  
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared ReadOnly Property Current As TaskScheduler

C#

public static TaskScheduler Current { get; }
Remarks

When not called from within a task, Current will return the Default scheduler.
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets the default `TaskScheduler` instance.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared ReadOnly Property Default As TaskScheduler

C#

public static TaskScheduler Default { get; }
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskScheduler....Id Property

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public ReadOnly Property Id As Integer

C#

customal int Id { get; }
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Indicates the maximum concurrency level this TaskScheduler is able to support.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)
Public Overridable ReadOnly Property MaximumConcurrencyLevel As Integer

C#
public virtual int MaximumConcurrencyLevel { get; }
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The TaskScheduler type exposes the following members.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnobservedTaskException</td>
<td>Occurs when a faulted Task's unobserved exception is about to trigger exception escalation policy, which, by default, would terminate the process.</td>
</tr>
</tbody>
</table>
See Also

TaskScheduler Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskScheduler.UnobservedTaskException Event

Occurs when a faulted Task's unobserved exception is about to trigger exception escalation policy, which, by default, would terminate the process.

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Shared Event UnobservedTaskException As EventHandler(Of UnobsedTaskExceptionEventArgs)

C#

public static event EventHandler<UnobservedTaskExceptionEventArgs> \
Remarks

This AppDomain-wide event provides a mechanism to prevent exception escalation policy (which, by default, terminates the process) from triggering. Each handler is passed a `UnobservedTaskExceptionEventArgs` instance, which may be used to examine the exception and to mark it as observed.
See Also

**TaskScheduler Class**  
**System.Threading.Tasks Namespace**

Send [feedback](mailto:feedback@microsoft.com) on this topic to Microsoft.
Represents an exception used to communicate an invalid operation by a TaskScheduler.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

<SerializableAttribute> _
Public Class TaskSchedulerException _
    Inherits Exception

C#

[SerializableAttribute]
public class TaskSchedulerException : Exception
Inheritance Hierarchy

System..::.Object
   System..::.Exception
      System.Threading.Tasks..::.TaskSchedulerException
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Visual Basic  C#
Include Protected Members
Include Inherited Members
.NET Framework Class Library
TaskSchedulerException Constructor

TaskSchedulerException Class  See Also  Send Feedback
## Overload List

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskSchedulerException()()</td>
<td>Initializes a new instance of the TaskSchedulerException class.</td>
</tr>
<tr>
<td>TaskSchedulerException(Exception)</td>
<td>Initializes a new instance of the TaskSchedulerException class using the default error message and a reference to the inner exception that is the cause of this exception.</td>
</tr>
<tr>
<td>TaskSchedulerException(String)</td>
<td>Initializes a new instance of the TaskSchedulerException class with a specified error message.</td>
</tr>
<tr>
<td>TaskSchedulerException(SerializationInfo, StreamingContext)</td>
<td>Initializes a new instance of the TaskSchedulerException class with serialized data.</td>
</tr>
<tr>
<td>TaskSchedulerException(String, Exception)</td>
<td>Initializes a new instance of the TaskSchedulerException class with a specified error message and a reference to the inner exception that is the cause of this exception.</td>
</tr>
</tbody>
</table>
See Also

TaskSchedulerException Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initialized a new instance of the **TaskSchedulerException** class.

**Namespace:**  [System.Threading.Tasks]

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New

C#

public TaskSchedulerException()
See Also

TaskSchedulerException Class
TaskSchedulerException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskSchedulerException Constructor (Exception)

Initializes a new instance of the TaskSchedulerException class using the default error message and a reference to the inner exception that is the cause of this exception.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
    innerException As Exception _
)

C#

public TaskSchedulerException(
    Exception innerException
)

Parameters

innerException
    Type: System..::.Exception
    The exception that is the cause of the current exception.
See Also

TaskSchedulerException Class
TaskSchedulerException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `TaskSchedulerException` class with a specified error message.

**Namespace:**  `System.Threading.Tasks`

**Assembly:**  `System.Threading` (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New (_
    message As String _
)

C#

public TaskSchedulerException(
    string message
)

Parameters

message
  Type: System..::.String
  The error message that explains the reason for the exception.
See Also

TaskSchedulerException Class
TaskSchedulerException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `TaskSchedulerException` class with serialized data.

**Namespace:** [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks?view=netframework-4.8)

**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Protected Sub New (_
    info As SerializationInfo, _
    context As StreamingContext _
)

C#

protected TaskSchedulerException(
    SerializationInfo info,
    StreamingContext context
)

Parameters

info
    Type: System.Runtime.Serialization.SerializationInfo
    The SerializationInfo that holds the serialized object data about the exception being thrown.

context
    Type: System.Runtime.Serialization.StreamingContext
    The StreamingContext that contains contextual information about the source or destination.
See Also

TaskSchedulerException Class
TaskSchedulerException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the TaskSchedulerException class with a specified error message and a reference to the inner exception that is the cause of this exception.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
**Syntax**

**Visual Basic (Declaration)**

```vbnet
Public Sub New ( _
    message As String, _
    innerException As Exception _
)
```

**C#**

```csharp
public TaskSchedulerException(
    string message,
    Exception innerException
)
```

**Parameters**

- **message**
  - Type: System.String
  - The error message that explains the reason for the exception.

- **innerException**
  - Type: System.Exception
  - The exception that is the cause of the current exception.
See Also

TaskSchedulerException Class
TaskSchedulerException Overload
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `TaskSchedulerException` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetBaseException</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetObjectData</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Exception.)</td>
</tr>
</tbody>
</table>
See Also

TaskSchedulerException Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `TaskSchedulerException` type exposes the following members.
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>HelpLink</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>HResult</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>InnerException</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Message</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>Source</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>StackTrace</td>
<td>(Inherited from Exception.)</td>
</tr>
<tr>
<td>TargetSite</td>
<td>(Inherited from Exception.)</td>
</tr>
</tbody>
</table>
See Also

TaskSchedulerException Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
TaskStatus Enumeration

Represents the current stage in the lifecycle of a Task.

**Namespace**: System.Threading.Tasks

**Assembly**: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Enumeration TaskStatus

C#

public enum TaskStatus
## Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created</td>
<td>The task has been initialized but has not yet been scheduled.</td>
</tr>
<tr>
<td></td>
<td>The task is waiting to be activated and scheduled internally by the .NET</td>
</tr>
<tr>
<td></td>
<td>Framework infrastructure.</td>
</tr>
<tr>
<td>WaitingForActivation</td>
<td>The task has been scheduled for execution but has not yet begun executing.</td>
</tr>
<tr>
<td>WaitingToRun</td>
<td>The task is running but has not yet completed.</td>
</tr>
<tr>
<td>Running</td>
<td>The task has finished executing and is implicitly waiting for attached child</td>
</tr>
<tr>
<td></td>
<td>tasks to complete.</td>
</tr>
<tr>
<td>RanToCompletion</td>
<td>The task completed execution successfully.</td>
</tr>
<tr>
<td>Canceled</td>
<td>The task acknowledged cancellation by throwing an OperationCanceledException</td>
</tr>
<tr>
<td></td>
<td>with its own CancellationToken while the token was in signaled state, or the</td>
</tr>
<tr>
<td></td>
<td>task's CancellationToken was already signaled before the task started</td>
</tr>
<tr>
<td></td>
<td>executing.</td>
</tr>
<tr>
<td>Faulted</td>
<td>The task completed due to an unhandled exception.</td>
</tr>
</tbody>
</table>
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Provides data for the event that is raised when a faulted Task’s exception goes unobserved.

**Namespace:** System.Threading.Tasks
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Class UnobservedTaskExceptionEventArgs
    Inherits EventArgs

C#

public class UnobservedTaskExceptionEventArgs : EventArgs
Remarks

The Exception property is used to examine the exception without marking it as observed, whereas the `SetObserved()` method is used to mark the exception as observed. Marking the exception as observed prevents it from triggering exception escalation policy which, by default, terminates the process.
Inheritance Hierarchy

System..::.Object
  System..::.EventArgs
    System.Threading.Tasks..::.UnobservedTaskExceptionEventArgs
See Also

System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Initializes a new instance of the `UnobservedTaskExceptionEventArgs` class with the unobserved exception.

**Namespace:** System.Threading.Tasks  
**Assembly:** System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub New ( _
   exception As AggregateException _
)

C#

public UnobservedTaskExceptionEventArgs(
   AggregateException exception
)

Parameters

exception
   Type: System.::.AggregateException
   The Exception that has gone unobserved.
See Also

UnobservedTaskExceptionEventArgs Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The `UnobservedTaskExceptionEventArgs` type exposes the following members.
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>Finalize</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>GetType</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>MemberwiseClone</td>
<td>(Inherited from Object.)</td>
</tr>
<tr>
<td>SetObserved</td>
<td>Marks the <strong>Exception</strong> as &quot;observed,&quot; thus preventing it from triggering exception escalation policy which, by default, terminates the process.</td>
</tr>
<tr>
<td>ToString</td>
<td>(Inherited from Object.)</td>
</tr>
</tbody>
</table>
See Also

UnobservedTaskExceptionEventArgs Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Marks the Exception as "observed," thus preventing it from triggering exception escalation policy which, by default, terminates the process.

**Namespace:**  System.Threading.Tasks
**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public Sub SetObserved

C#

public void SetObserved()
See Also

UnobservedTaskExceptionEventArgs Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The **UnobservedTaskExceptionEventArgs** type exposes the following members.
## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
<td>The Exception that went unobserved.</td>
</tr>
<tr>
<td>Observed</td>
<td>Gets whether this exception has been marked as &quot;observed.&quot;</td>
</tr>
</tbody>
</table>
See Also

UnobservedTaskExceptionEventArgs Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
The Exception that went unobserved.

**Namespace:**  [System.Threading.Tasks](#)

**Assembly:**  System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Exception As AggregateException

C#

public AggregateException Exception { get; }
See Also

UnobservedTaskExceptionEventArgs Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.
Gets whether this exception has been marked as "observed."

Namespace: System.Threading.Tasks
Assembly: System.Threading (in System.Threading.dll)
Syntax

Visual Basic (Declaration)

Public ReadOnly Property Observed As Boolean

C#

public bool Observed { get; }
See Also

UnobservedTaskExceptionEventArgs Class
System.Threading.Tasks Namespace

Send feedback on this topic to Microsoft.