Dataflow

Task Parallel Library(TPL) provides dataflow components to help increase the robustness of concurrency-enabled applications. These dataflow components are collectively referred to as the TPL Dataflow Library. This dataflow model promotes actor-based programming, because it provides in-process message passing for coarse-grained dataflow and pipelining tasks.

Dataflow components are useful when you have multiple operations that must communicate with one another asynchronously or when you want to process data as it becomes available. For example, consider an application that processes image data from a web camera. By using the dataflow model, the application can process image frames as they become available. If the application enhances image frames, by performing light correction or red-eye reduction, you can create a pipeline of dataflow components. Each stage of the pipeline might use more coarse-grained parallelism functionality, such as the functionality that is provided by the TPL, to transform the image.

Programming Model:

TPL Dataflow Lib provides a foundation for message passing and parallelizing CPU-intensive and I/O-intensive applications that have high throughput and low latency. Under dataflow model, you declare how data is handled when it becomes available, and also any dependencies between data. Because the runtime manages dependencies between data, you can often avoid the requirement to synchronize access to shared data. Also rutime schedules work based on the asynchronous arrival of data, dataflow can improve responsiveness and thrgouhput by efficiently managing the underlying threads.

Source and Targets:

TPL Dataflow library consists of dataflow blocks, which are data structures that buffer and process data. There are three kind of dataflow blocks: source, target and progagator blocks. Source is to be read from, target is to be written to, and propagator is both.

The TPL defines the [System.Threading.Tasks.Dataflow.ISourceBlock<TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.isourceblock-1) interface to represent sources, [System.Threading.Tasks.Dataflow.ITargetBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.itargetblock-1) to represent targets, and [System.Threading.Tasks.Dataflow.IPropagatorBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.ipropagatorblock-2) to represent propagators. [IPropagatorBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.ipropagatorblock-2)inherits from both [ISourceBlock<TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.isourceblock-1), and [ITargetBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.itargetblock-1).

There are a few predefined dataflow block types in TPL like

Buffering Blocks:

[System.Threading.Tasks.Dataflow.BufferBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.bufferblock-1), [System.Threading.Tasks.Dataflow.BroadcastBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.broadcastblock-1), and [System.Threading.Tasks.Dataflow.WriteOnceBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.writeonceblock-1).

BufferBlock<T>

These are general purpos asynchronous messaging structure. Stores a first in first out queue of messages that can be written to by multiple sources or read from by multiple targes. When target receives a message from a BufferBlock<T>, that message is removed from the message queue. So even if a BufferBlock<T> object can have multiple targets, only one target will receive each message.

BroadcastBlock

The [BroadcastBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.broadcastblock-1) class is useful when you must pass multiple messages to another component, but that component needs only the most recent value. This class is also useful when you want to broadcast a message to multiple components. Values are not removed from BroadcastBlock<T> objects after they are read, the same value is available every time.

WriteOnceBlock<T>

The [WriteOnceBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.writeonceblock-1) class resembles the [BroadcastBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.broadcastblock-1) class, except that a [WriteOnceBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.writeonceblock-1) object can be written to one time only. You can think of [WriteOnceBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.writeonceblock-1) as being similar to the C# [readonly](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/readonly) ([ReadOnly](https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/readonly) in Visual Basic) keyword, except that a [WriteOnceBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.writeonceblock-1) object becomes immutable after it receives a value instead of at construction. Like the [BroadcastBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.broadcastblock-1)class, when a target receives a message from a [WriteOnceBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.writeonceblock-1) object, that message is not removed from that object. Therefore, multiple targets receive a copy of the message. The [WriteOnceBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.writeonceblock-1) class is useful when you want to propagate only the first of multiple messages.

Execution Blocks:

Execution blocks call a user-provided delegate for each piece of received data. The TPL Dataflow Library provides three execution block types: [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1), [System.Threading.Tasks.Dataflow.TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2), and [System.Threading.Tasks.Dataflow.TransformManyBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformmanyblock-2).

ActionBlock<T>

The [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1) class is a target block that calls a delegate when it receives data. Think of a [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1) object as a delegate that runs asynchronously when data becomes available. The delegate that you provide to an [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1) object can be of type [Action<T>](https://docs.microsoft.com/en-us/dotnet/api/system.action-1) or type System.Func<TInput, Task>. When you use an [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1) object with [Action<T>](https://docs.microsoft.com/en-us/dotnet/api/system.action-1), processing of each input element is considered completed when the delegate returns. When you use an [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1) object with System.Func<TInput, Task>, processing of each input element is considered completed only when the returned [Task](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task) object is completed. By using these two mechanisms, you can use [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1) for both synchronous and asynchronous processing of each input element.

TransormBlock<TInput, TOutput>

The [TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2) class resembles the [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1) class, except that it acts as both a source and as a target. The delegate that you pass to a [TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2) object returns a value of type TOutput. The delegate that you provide to a [TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2) object can be of type System.Func<TInput, TOutput> or type System.Func<TInput, Task<TOutput>>. When you use a [TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2) object with System.Func<TInput, TOutput>, processing of each input element is considered completed when the delegate returns.

**TransformManyBlock(TInput, TOutput)**

The [TransformManyBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformmanyblock-2) class resembles the [TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2) class, except that [TransformManyBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformmanyblock-2) produces zero or more output values for each input value, instead of only one output value for each input value.

Every [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1), [TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2), and [TransformManyBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformmanyblock-2) object buffers input messages until the block is ready to process them. By default, these classes process messages in the order in which they are received, one message at a time. You can also specify the degree of parallelism to enable [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1),  [TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2) and [TransformManyBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformmanyblock-2) objects to process multiple messages concurrently.

| **Type** | **Synchronous Delegate Type** | **Asynchronous Delegate Type** |
| --- | --- | --- |
| [ActionBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.actionblock-1) | System.Action | System.Func<TInput, Task> |
| [TransformBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformblock-2) | System.Func<TInput, TOutput> | System.Func<TInput, Task<TOutput>> |
| [TransformManyBlock<TInput,TOutput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.transformmanyblock-2) | System.Func<TInput, IEnumerable<TOutput>> | System.Func<TInput, Task<IEnumerable<TOutput>>> |

Grouping Blocks

Grouping blocks combine data from one or more sources and under various constraints. The TPL Dataflow Library provides three join block types: BatchBlock<T>, JoinBlock<T1, t2>, AND BatchedJoinBlock<T1, T2)

#### BatchBlock(T)

The [BatchBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchblock-1) class combines sets of input data, which are known as batches, into arrays of output data. After specifying the size of each batch when creating a BatchBlock<T> object. When the BatchBlock<T> object receives the specified count of input elements, it asynchronously propagates out an array that contains those elements. If a BatchBlock<T> object is set to the completed state but does not contain enough elements to form a batch, it propagates out a final array that contains the remaining input elements.

#### JoinBlock(T1, T2, ...)

The [JoinBlock<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.joinblock-2) and [JoinBlock<T1,T2,T3>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.joinblock-3) classes collect input elements and propagate out [System.Tuple<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.tuple-2) or [System.Tuple<T1,T2,T3>](https://docs.microsoft.com/en-us/dotnet/api/system.tuple-3) objects that contain those elements. The [JoinBlock<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.joinblock-2) and [JoinBlock<T1,T2,T3>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.joinblock-3) classes do not inherit from [ITargetBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.itargetblock-1). Instead, they provide properties, [Target1](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.joinblock-2.target1), [Target2](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.joinblock-2.target2), and [Target3](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.joinblock-3.target3), that implement [ITargetBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.itargetblock-1).

#### BatchedJoinBlock(T1, T2, ...)

The [BatchedJoinBlock<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchedjoinblock-2) and [BatchedJoinBlock<T1,T2,T3>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchedjoinblock-3) classes collect batches of input elements and propagate out System.Tuple(IList(T1), IList(T2)) or System.Tuple(IList(T1), IList(T2), IList(T3)) objects that contain those elements. Think of [BatchedJoinBlock<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchedjoinblock-2) as a combination of [BatchBlock<T>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchblock-1) and [JoinBlock<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.joinblock-2). Specify the size of each batch when you create a [BatchedJoinBlock<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchedjoinblock-2) object. [BatchedJoinBlock<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchedjoinblock-2) also provides properties, [Target1](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchedjoinblock-2.target1) and [Target2](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchedjoinblock-2.target2), that implement [ITargetBlock<TInput>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.itargetblock-1). When the specified count of input elements are received from across all targets, the [BatchedJoinBlock<T1,T2>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.batchedjoinblock-2)object asynchronously propagates out a System.Tuple(IList(T1), IList(T2)) object that contains those elements.

Connecting Blocks

We can connect dataflow blocks to form pipelines, which are linear sequences of dataflow blocks, or networks, which are graphs of dataflow blocks. A pipeline is one form of network. In a pipeline or network, sources asynchronously propagate data to targets as that data becomes available. The ISourceBlock<TOutput>.LinkTo method links a source dataflow block to a target block. A source can be linked to zero or more targets;targets can be linked from zero or more sources. You can add or remove dataflow blocks to or from a pipeline or network concurrently. The predefined dataflow block types handl all thread safety aspects of linking and unlinking.

Filtering

When we call ISourceBlock>TOutput>.LinkTo method to link a source to a target, we can supply a delegate that determins whether the target block accepts or rejects a message based on the value of that message.

Message Passing

The dataflow programming model is related to the concept of message passing, where independent components of a program communicate with one another by sending messages. One way to propagate messages among application components is to call the [Post](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.dataflowblock.post) and [DataflowBlock.SendAsync](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.dataflowblock.sendasync) methods to send messages to target dataflow blocks post ([Post](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.dataflowblock.post) acts synchronously; [SendAsync](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.dataflowblock.sendasync)acts asynchronously) and the [Receive](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.dataflowblock.receive), [ReceiveAsync](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.dataflowblock.receiveasync), and [TryReceive](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.dataflow.dataflowblock.tryreceive) methods to receive messages from source blocks.

### Dataflow Block Completion

Dataflow blocks also support the concept of completion. A dataflow block that is in the completed state does not perform any further work. Each dataflow block has an associated [System.Threading.Tasks.Task](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task) object, known as a completion task, that represents the completion status of the block. Because you can wait for a [Task](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task) object to finish, by using completion tasks, you can wait for one or more terminal nodes of a dataflow network to finish.