Namespace Biocs

Classes

Location

Represents the region of the biological sequence.

StringResourceUsageAttribute

Specifies the usage of string resources.

Structs

Codon

Represents a nucleotide triplet.

DnaBase

Represents nucleotides for DNA.

Struct Codon

Namespace: Biocs

Assembly: Biocs.Core.dll

Represents a nucleotide triplet.

```
public readonly struct Codon : IEquatable<Codon>
```

Implements
IEquatable <Codon>

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

The default constructor creates an object whose value is Gap.

Constructors

Codon(DnaBase, DnaBase, DnaBase)

Represents a nucleotide triplet.

```
public Codon(DnaBase first, DnaBase second, DnaBase third)
```

Parameters

`first` DnaBase

The nucleotide in the first position.

`second` DnaBase

The nucleotide in the second position.

`third` DnaBase

The nucleotide in the third position.

Remarks

The default constructor creates an object whose value is Gap.

Properties

Any

Gets a codon that is filled with unknown bases.

```
public static Codon Any { get; }
```

Property Value

Codon

First

Gets the nucleotide in the first position of this codon.

```
public DnaBase First { get; }
```

Property Value

DnaBase

Gap

Gets a codon that is filled with gaps.

```
public static Codon Gap { get; }
```

Property Value

Codon

IsAtomic

Gets a value indicating whether this codon is completely specified.

```
public bool IsAtomic { get; }
```

Property Value

bool

Second

Gets the nucleotide in the second position of this codon.

```
public DnaBase Second { get; }
```

Property Value

DnaBase

Symbols

Gets the string representation of this codon.

```
public string Symbols { get; }
```

Property Value

string

Third

Gets the nucleotide in the third position of this codon.

```
public DnaBase Third { get; }
```

Property Value

DnaBase

Methods

Equals(Codon)

Determines whether the current Codon instance is equal to a specified Codon instance.

```
public bool Equals(Codon other)
```

Parameters

`other` Codon

The codon to compare to this instance.

Returns

bool

true if the two instances are equal; otherwise, false.

Equals(object?)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object? obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

Parse(ReadOnlySpan<char>)

Converts the string representation of a codon to an equivalent Codon instance.

```
public static Codon Parse(ReadOnlySpan<char> value)
```

Parameters

`value` ReadOnlySpan<char>

A string to convert.

Returns

Codon

A Codon instance whose symbol is represented by `value`.

Exceptions

ArgumentException

`value` contains an unknown character in a certain position.

ToLower()

Converts this codon to its lowercase equivalent.

```
public Codon ToLower()
```

Returns

Codon

The lowercase equivalent of this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToUpper()

Converts this codon to its uppercase equivalent.

```
public Codon ToUpper()
```

Returns

Codon

The uppercase equivalent of this instance.

TryParse(ReadOnlySpan<char>, out Codon)

Tries to convert the string representation of a codon to an equivalent Codon instance, and returns a value that indicates whether the conversion succeeded.

```
public static bool TryParse(ReadOnlySpan<char> value, out Codon result)
```

Parameters

`value` ReadOnlySpan<char>

A string with a length of 3 characters to convert.

`result` Codon

When this method returns, `result` contains a Codon instance that is represented by `value` if the conversion succeeded, or Gap if the conversion failed.

Returns

bool

true if `value` was converted successfully; otherwise, false.

Operators

operator ==(Codon, Codon)

Compares two Codon structures for equality.

```
public static bool operator ==(Codon one, Codon other)
```

Parameters

`one` Codon

The first instance of Codon to compare.

`other` Codon

The second instance of Codon to compare.

Returns

bool

true if the two instances are equal; otherwise, false.

operator !=(Codon, Codon)

Compares two Codon structures for inequality.

```
public static bool operator !=(Codon one, Codon other)
```

Parameters

`one` Codon

The first instance of Codon to compare.

`other` Codon

The second instance of Codon to compare.

Returns

bool

false if the two instances are equal; otherwise, true.

Struct DnaBase

Namespace: Biocs

Assembly: Biocs.Core.dll

Represents nucleotides for DNA.

```
public readonly struct DnaBase : IEquatable<DnaBase>
```

Implements
IEquatable <DnaBase>

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

Each member other than Name property and EqualsCaseInsensitive(DnaBase) method performs a case-sensitive operation. By default, each instance is uppercase except gaps.

The default constructor creates an object whose value is Gap.

Properties

Adenine

Gets the DnaBase instance for adenine.

```
public static DnaBase Adenine { get; }
```

Property Value

DnaBase

Any

Gets the DnaBase instance for an unknown base.

```
public static DnaBase Any { get; }
```

Property Value

DnaBase

Cytosine

Gets the DnaBase instance for cytosine.

```
public static DnaBase Cytosine { get; }
```

Property Value

DnaBase

Gap

Gets the DnaBase instance for a gap.

```
public static DnaBase Gap { get; }
```

Property Value

DnaBase

Guanine

Gets the DnaBase instance for guanine.

```
public static DnaBase Guanine { get; }
```

Property Value

DnaBase

IsAtomic

Gets a value indicating whether this nucleotide is completely specified.

```
public bool IsAtomic { get; }
```

Property Value

bool

IsGap

Gets a value indicating whether this instance represents a gap.

```
public bool IsGap { get; }
```

Property Value

bool

IsLower

Gets a value indicating whether this nucleotide has a lowercase alphabetic symbol and is not a gap.

```
public bool IsLower { get; }
```

Property Value

bool

IsUpper

Gets a value indicating whether this nucleotide has an uppercase alphabetic symbol and is not a gap.

```
public bool IsUpper { get; }
```

Property Value

bool

Name

Gets the description of this nucleotide.

```
public string Name { get; }
```

Property Value

string

Remarks

This property doesn't distinguish between uppercase and lowercase.

Symbol

Gets the character representation of this nucleotide.

```
public char Symbol { get; }
```

Property Value

char

Thymine

Gets the DnaBase instance for thymine.

```
public static DnaBase Thymine { get; }
```

Property Value

DnaBase

Methods

Complement()

Returns a complentary nucleotide of this nucleotide.

```
public DnaBase Complement()
```

Returns

DnaBase

A complementary nucleotide.

Equals(DnaBase)

Determines whether the current DnaBase instance is equal to a specified DnaBase instance.

```
public bool Equals(DnaBase other)
```

Parameters

`other` DnaBase

The nucleotide to compare to this instance.

Returns

bool

true if the two instances are equal; otherwise, false.

Equals(object?)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object? obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

EqualsCaseInsensitive(DnaBase)

Compares two DnaBase structures ignoring case for equality.

```
public bool EqualsCaseInsensitive(DnaBase other)
```

Parameters

`other` DnaBase

The nucleotide to compare to this instance.

Returns

bool

true if the two instances are equal; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

Parse(char)

Converts the character representation of a nucleotide to an equivalent DnaBase instance.

```
public static DnaBase Parse(char value)
```

Parameters

`value` char

A character to convert.

Returns

DnaBase

A DnaBase instance whose symbol is represented by `value`.

Exceptions

ArgumentException

`value` is not one of the symbols defined for DnaBase.

ToLower()

Converts the value of a nucleotide to its lowercase equivalent.

```
public DnaBase ToLower()
```

Returns

DnaBase

The lowercase equivalent of this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToUpper()

Converts the value of a nucleotide to its uppercase equivalent.

```
public DnaBase ToUpper()
```

Returns

DnaBase

The uppercase equivalent of this instance.

TryParse(char, out DnaBase)

Tries to convert the character representation of a nucleotide to an equivalent DnaBase instance, and returns a value that indicates whether the conversion succeeded.

```
public static bool TryParse(char value, out DnaBase result)
```

Parameters

`value` char

A character to convert.

`result` DnaBase

When this method returns, `result` contains a DnaBase instance whose symbol is represented by `value` if the conversion succeeded, or Gap if the conversion failed.

Returns

bool

true if `value` was converted successfully; otherwise, false.

Operators

operator ==(DnaBase, DnaBase)

Compares two DnaBase structures for equality.

```
public static bool operator ==(DnaBase one, DnaBase other)
```

Parameters

`one` DnaBase

The first instance of DnaBase to compare.

`other` DnaBase

The second instance of DnaBase to compare.

Returns

bool

true if the two instances are equal; otherwise, false.

operator !=(DnaBase, DnaBase)

Compares two DnaBase structures for inequality.

```
public static bool operator !=(DnaBase one, DnaBase other)
```

Parameters

`one` DnaBase

The first instance of DnaBase to compare.

`other` DnaBase

The second instance of DnaBase to compare.

Returns

bool

false if the two instances are equal; otherwise, true.

Class Location

Namespace: Biocs

Assembly: Biocs.Core.dll

Represents the region of the biological sequence.

```
public class Location : IEquatable<Location?>, IComparable<Location?>,
ISpanParsable<Location>, IParsable<Location>
```

Inheritance

object ← Location

Implements

IEquatable<Location>, IComparable<Location>, ISpanParsable<Location>, IParsable<Location>

Inherited Members

object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

Each base numbering is one-based indexing.

Compliant with [The DDBJ/ENA/GenBank Feature Table Definition] (http://www.insdc.org/files/feature_table.html)

Properties

Elements

```
public IReadOnlyList<Location> Elements { get; }
```

Property Value

IReadOnlyList<Location>

End

```
public int End { get; }
```

Property Value

int

IsComplement

Gets a value that indicates whether the current Location object represents the complementary strand of the specified sequence.

```
public bool IsComplement { get; }
```

Property Value

bool

IsExactEnd

Gets a value that indicates whether the exact ending base number is known.

```
public bool IsExactEnd { get; }
```

Property Value

bool

IsExactStart

Gets a value that indicates whether the exact starting base number is known.

```
public bool IsExactStart { get; }
```

Property Value

bool

IsSpan

Gets a value that indicates whether the current Location object represents a continuous range.

```
public bool IsSpan { get; }
```

Property Value

bool

Length

```
public int Length { get; }
```

Property Value

int

SequenceID

```
public string? SequenceID { get; }
```

Property Value

string

Start

```
public int Start { get; }
```

Property Value

int

Methods

CompareTo(Location?)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

```
public int CompareTo(Location? other)
```

Parameters

`other` Location

An object to compare with this instance.

Returns

int

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Value	Meaning
Less than zero	This instance precedes `other` in the sort order.
Zero	This instance occurs in the same position in the sort order as `other`.
Greater than zero	This instance follows `other` in the sort order.

Equals(Location?)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Location? other)
```

Parameters

`other` Location

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object?)

Determines whether the specified object is equal to the current object.

```
public override bool Equals(object? obj)
```

Parameters

`obj` object

The object to compare with the current object.

Returns

bool

true if the specified object is equal to the current object; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer hash code.

Parse(ReadOnlySpan<char>, IFormatProvider?)

Parses a span of characters into a value.

```
public static Location Parse(ReadOnlySpan<char> s, IFormatProvider? provider)
```

Parameters

`s` ReadOnlySpan<char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Location

The result of parsing `s`.

Exceptions

FormatException

`s` is not in the correct format.

OverflowException

`s` is not representable by Location.

Parse(string, IFormatProvider?)

Parses a string into a value.

```
public static Location Parse(string s, IFormatProvider? provider)
```

Parameters

`s` string

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

Returns

Location

The result of parsing `s`.

Exceptions

ArgumentNullException

`s` is null.

FormatException

`s` is not in the correct format.

OverflowException

`s` is not representable by Location.

ToString()

Converts the current Location object to its equivalent string representation.

```
public override string ToString()
```

Returns

string

The string representation of the current Location object.

TryParse(ReadOnlySpan<char>, IFormatProvider?, out Location)

Tries to parse a span of characters into a value.

```
public static bool TryParse(ReadOnlySpan<char> s, IFormatProvider? provider, out
Location result)
```

Parameters

`s` ReadOnlySpan<char>

The span of characters to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Location

When this method returns, contains the result of successfully parsing `s`, or an undefined value on failure.

Returns

bool

true if `s` was successfully parsed; otherwise, false.

TryParse(string?, IFormatProvider?, out Location)

Tries to parse a string into a value.

```
public static bool TryParse(string? s, IFormatProvider? provider, out Location result)
```

Parameters

`s` string

The string to parse.

`provider` IFormatProvider

An object that provides culture-specific formatting information about `s`.

`result` Location

When this method returns, contains the result of successfully parsing `s` or an undefined value on failure.

Returns

bool

true if `s` was successfully parsed; otherwise, false.

Operators

operator ==(Location?, Location?)

```
public static bool operator ==(Location? left, Location? right)
```

Parameters

`left` Location

`right` Location

Returns

bool

operator >(Location?, Location?)

```
public static bool operator >(Location? left, Location? right)
```

Parameters

`left` Location

`right` Location

Returns

bool

operator >=(Location?, Location?)

Determines whether the first specified Location object is greater than or equal to the second specified Location object.

```
public static bool operator >=(Location? left, Location? right)
```

Parameters

`left` Location

The first Location object.

`right` Location

The second Location object.

Returns

bool

true if `left` is greater than or equal to `right`; otherwise, false.

operator !=(Location?, Location?)

```
public static bool operator !=(Location? left, Location? right)
```

Parameters

`left` Location

`right` Location

Returns

bool

operator <(Location?, Location?)

```
public static bool operator <(Location? left, Location? right)
```

Parameters

`left` Location

`right` Location

Returns

bool

operator <=(Location?, Location?)

```
public static bool operator <=(Location? left, Location? right)
```

Parameters

`left` Location

`right` Location

Returns

bool

Class StringResourceUsageAttribute

Namespace: Biocs

Assembly: Biocs.Core.dll

Specifies the usage of string resources.

```
[AttributeUsage(AttributeTargets.Constructor|AttributeTargets.Method, AllowMultiple = true,
Inherited = false)]
[Conditional("DEBUG")]
public sealed class StringResourceUsageAttribute : Attribute
```

Inheritance

object ← Attribute ← StringResourceUsageAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) , Attribute.GetCustomAttribute(Assembly, Type, bool) , Attribute.GetCustomAttribute(MemberInfo, Type) , Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) , Attribute.GetCustomAttribute(ParameterInfo, Type) , Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) , Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) , Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) , Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) , Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) , Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) , Attribute.GetCustomAttributes(ParameterInfo, bool) , Attribute.GetCustomAttributes(ParameterInfo, Type) , Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() , Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) , Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) , Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) , Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This API is not intended to be used directly from your code.

Constructors

StringResourceUsageAttribute(string, int)

Initializes a new instance of the StringResourceUsageAttribute class with the number of format items.

```
public StringResourceUsageAttribute(string name, int formatItemCount = 0)
```

Parameters

`name` string

The name of the string resource to be used.

`formatItemCount` int

The number of format items contained in the value of the string resource.

Exceptions

ArgumentNullException

`name` is null.

ArgumentOutOfRangeException

`formatItemCount` is less than 0.

Properties

FormatItemCount

Gets the number of format items contained in the value of the string resource.

```
public int FormatItemCount { get; }
```

Property Value

int

Name

Gets the name of the string resource to be used.

```
public string Name { get; }
```

Property Value

string

ResourceCheckOnly

Gets or sets a value indicating whether the name and the value in the resource should only be checked.

```
public bool ResourceCheckOnly { get; set; }
```

Property Value

bool

Remarks

If any element is generated from the applied method by a compiler, or the local resource class is not used for the formatting operation, the value of this property is set to true. In that case, a tester will not check the body of the applied method.

Namespace Biocs.Collections

Classes

CollectionTools

Provides static methods for collections.

Counter<T>

Represents a tally counter to count the frequency of items.

Deque<T>

Represents a double-ended queue with a dynamic array.

Class CollectionTools

Namespace: Biocs.Collections

Assembly: Biocs.Core.dll

Provides static methods for collections.

```
public static class CollectionTools
```

Inheritance

object ← CollectionTools

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

AllItemsAreEqual<T>(IEnumerable<T>, IEqualityComparer<T>?)

Determines whether all items in the specified collection are equal.

```
public static bool AllItemsAreEqual<T>(this IEnumerable<T> collection, IEqualityComparer<T>?
comparer = null)
```

Parameters

`collection` IEnumerable<T>

The IEnumerable<T> to check equality between items.

`comparer` IEqualityComparer<T>

An IEqualityComparer<T> to use to compare items. The default value is Default.

Returns

bool

true if `collection` is not empty and all items are equal; otherwise, false.

Type Parameters

`T`

The type of items of `collection`.

Exceptions

ArgumentNullException

`collection` is null.

AllItemsAreEqual<T>(IEnumerable<T>, IEqualityComparer<T>?, out T)

Determines whether all items in the specified collection are equal, and tries to get the unique item.

```
public static bool AllItemsAreEqual<T>(this IEnumerable<T> collection, IEqualityComparer<T>?
comparer, out T value)
```

Parameters

`collection` IEnumerable<T>

The IEnumerable<T> to check equality between items.

`comparer` IEqualityComparer<T>

An IEqualityComparer<T> to use to compare items, or null to use the default IEqualityComparer<T>.

`value` T

When this method returns, `value` contains the first item of `collection` if all items are equal, or the default value for the `T` type if `collection` is empty or contains different items.

Returns

bool

true if `collection` is not empty and all items are equal; otherwise, false.

Type Parameters

`T`

The type of items of `collection`.

Exceptions

ArgumentNullException

`collection` is null.

Class Counter<T>

Namespace: Biocs.Collections

Assembly: Biocs.Core.dll

Represents a tally counter to count the frequency of items.

```
public class Counter<T>
```

Type Parameters

`T`

The type of items to count.

Inheritance

object ← Counter<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Counter<T> accepts null as a valid value for reference types.

Constructors

Counter(Counter<T>)

Initializes a new instance of the Counter<T> class that contains unique items and counts copied from the specified Counter<T> and uses the same equality comparer.

```
public Counter(Counter<T> other)
```

Parameters

`other` Counter<T>

The Counter<T> whose unique items and counts are copied to the new Counter<T>.

Exceptions

ArgumentNullException

`other` is null.

Counter(IEqualityComparer<T>?)

Initializes a new instance of the Counter<T> class that is empty, has zero capacity, and uses the specified equality comparer.

```
public Counter(IEqualityComparer<T>? comparer)
```

Parameters

`comparer` IEqualityComparer<T>

The IEqualityComparer<T> implementation to use when comparing items, or null to use the default IEqualityComparer<T> for the type of the item.

Counter(int, IEqualityComparer<T>?)

Initializes a new instance of the Counter<T> class that is empty, has the specified initial capacity, and uses the specified equality comparer.

```
public Counter(int capacity = 0, IEqualityComparer<T>? comparer = null)
```

Parameters

`capacity` int

The initial number of items that the Counter<T> can contain.

`comparer` IEqualityComparer<T>

The IEqualityComparer<T> implementation to use when comparing items. The default value is Default.

Exceptions

ArgumentOutOfRangeException

`capacity` is less than 0.

Properties

Comparer

Gets the IEqualityComparer<T> that is used to determine equality of items for the Counter<T>.

```
public IEqualityComparer<T> Comparer { get; }
```

Property Value

IEqualityComparer<T>

NumberOfItems

Gets the number of the kinds of items that the Counter<T> contains.

```
public int NumberOfItems { get; }
```

Property Value

int

RepeatedItems

Gets an enumerable collection that contains items repeated by each count.

```
public IEnumerable<T> RepeatedItems { get; }
```

Property Value

IEnumerable<T>

TotalCount

Gets the total count of items.

```
public int TotalCount { get; }
```

Property Value

int

UniqueItems

Gets an enumerable collection of unique items that the Counter<T> has counted before now.

```
public IEnumerable<T> UniqueItems { get; }
```

Property Value

IEnumerable <T>

Remarks

This enumerable collection also contains items whose the count is 0.

Enumerators retured by this enumerable collection cannot be used to modify the Counter<T>. For example, the following enumeration raises an InvalidOperationException.

```
var counter = new Counter<int>();
counter.AddRange(new[] { 1, 2, 3 });
foreach (int item in counter.UniqueItems)
{
  counter.Reset(item);
}
```

Methods

Add(T)

Counts an object once.

```
public void Add(T item)
```

Parameters

`item` T

The object to be counted to the Counter<T>.

Add(T, int)

Counts an object a specified number of times.

```
public void Add(T item, int times)
```

Parameters

`item` T

The object to be counted to the Counter<T>.

`times` int

The number of times to count `item`.

Exceptions

ArgumentOutOfRangeException

`times` is less than 0.

AddRange(IEnumerable<T>)

Counts the items of the specified collection.

```
public void AddRange(IEnumerable<T> items)
```

Parameters

`items` IEnumerable<T>

The collection whose items should be counted.

Exceptions

ArgumentNullException

`items` is null.

Clear()

Removes all items from the Counter<T>.

```
public void Clear()
```

Contains(T)

Determines whether the Counter<T> contains the specified object.

```
public bool Contains(T item)
```

Parameters

`item` T

The object to locate in the Counter<T>.

Returns

bool

true if `item` is found in the Counter<T>; otherwise, false.

CopyTo(T[], int)

Copies the Counter<T> unique items to an existing one-dimensional Array, starting at the specified array index.

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

Parameters

`array` T[]

The one-dimensional Array that is the destination of the unique items copied from Counter<T>.

`arrayIndex` int

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

Exceptions

ArgumentNullException

`array` is null.

ArgumentOutOfRangeException

`arrayIndex` is less than 0.

ArgumentException

The number of items in the Counter<T> is greater than the available space from `arrayIndex` to the end of the destination `array`.

GetCount(T)

Gets the number of times that the item occurs in the Counter<T>.

```
public int GetCount(T item)
```

Parameters

`item` T

The object to get the count.

Returns

int

The number of times that `item` occurs in the Counter<T>.

Remarks

If `item` is not contained in the Counter<T>, this method returns 0.

Remove(T)

Decreases the count of the specified item by one.

```
public bool Remove(T item)
```

Parameters

`item` T

The item to decrement the count value.

Returns

bool

true if the count of `item` was successfully decremented; otherwise, false.

Remove(T, int)

Decreases the count of the specified item by the specified amount.

```
public int Remove(T item, int times)
```

Parameters

`item` T

The item to decrement the count value.

`times` int

The amount by which to decrement the counter value.

Returns

int

The amount of the count to be decreased acutually.

ResetCount(T)

Sets the number of times that the specified item occurs in the Counter<T> to zero.

```
public void ResetCount(T item)
```

Parameters

`item` T

The item to reset the count.

ResetCounts()

Sets the number of times that each item occurs in the Counter<T> to zero. The collection of items is preserved.

```
public void ResetCounts()
```

Class Deque<T>

Namespace: Biocs.Collections

Assembly: Biocs.Core.dll

Represents a double-ended queue with a dynamic array.

```
public sealed class Deque<T> : IList<T>, ICollection<T>, IReadOnlyList<T>,
IReadOnlyCollection<T>, IEnumerable<T>, IEnumerable
```

Type Parameters

`T`

The element type of the double-ended queue.

Inheritance

object ← Deque<T>

Implements

IList<T>, ICollection<T>, IReadOnlyList<T>, IReadOnlyCollection<T>, IEnumerable<T>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Extension Methods

CollectionTools.AllItemsAreEqual<T>(IEnumerable<T>, IEqualityComparer<T>?) , CollectionTools.AllItemsAreEqual<T>(IEnumerable<T>, IEqualityComparer<T>?, out T)

Constructors

Deque(IEnumerable<T>)

Initializes a new instance of the Deque<T> class that contains elements copied from the specified IEnumerable<T>.

```
public Deque(IEnumerable<T> collection)
```

Parameters

`collection` IEnumerable<T>

The IEnumerable<T> whose elements are copied to the new Deque<T>.

Remarks

The elements are copied onto the Deque<T> in the same order they are read by the enumerator of `collection`. If the type of `collection` implements ICollection<T>, CopyTo(T[], int) is used to copy elements.

Exceptions

ArgumentNullException

`collection` is null.

Deque(int)

Initializes a new instance of the Deque<T> class that is empty and has the specified initial capacity.

```
public Deque(int capacity = 0)
```

Parameters

`capacity` int

The initial number of elements that the Deque<T> can contain.

Exceptions

ArgumentOutOfRangeException

`capacity` is less than 0.

Properties

Capacity

Gets or sets the total number of elements the internal data structure can hold without resizing.

```
public int Capacity { get; set; }
```

Property Value

int

Exceptions

ArgumentOutOfRangeException

The value in a set operation is less than Count.

Count

Gets the number of elements actually contained in the Deque<T>.

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

Property Value

int

First

Gets or sets the first element of the Deque<T>.

```
public T First { get; set; }
```

Property Value

T

Exceptions

InvalidOperationException

The Deque<T> is empty.

this[int]

Gets or sets the element at the specified index.

```
public T this[int index] { get; set; }
```

Parameters

`index` int

The zero-based index of the element to get or set.

Property Value

T

The element at the specified index.

Exceptions

ArgumentOutOfRangeException

`index` is less than 0.

-or-
`index` is equal to or greater than Count.

Last

Gets or sets the last element of the Deque<T>.

```
public T Last { get; set; }
```

Property Value

T

Exceptions

InvalidOperationException

The Deque<T> is empty.

Methods

AddFirst(T)

Adds a new element at the start of the Deque<T>.

```
public void AddFirst(T item)
```

Parameters

`item` T

The value to add at the start of the Deque<T>.

AddLast(T)

Adds a new element at the end of the Deque<T>.

```
public void AddLast(T item)
```

Parameters

`item` T

The value to add at the end of the Deque<T>.

Clear()

Removes all elements from the Deque<T>.

```
public void Clear()
```

Contains(T)

Determines whether an element is in the Deque<T>.

```
public bool Contains(T item)
```

Parameters

`item` T

The value to locate in the Deque<T>.

Returns

bool

true if `item` is found in the Deque<T>; otherwise, false.

CopyTo(int, T[], int, int)

Copies a range of elements from the Deque<T> to an existing one-dimensional Array.

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

Parameters

`index` int

The zero-based index in the Deque<T> at which copying begins.

`array` T[]

The one-dimensional Array that is the destination of the elements copied from Deque<T>.

`arrayIndex` int

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

`count` int

The number of elements to copy.

Exceptions

ArgumentNullException

`array` is null.

ArgumentOutOfRangeException

`index`, `arrayIndex` or `count` is less than 0.

ArgumentException

`count` is greater than the number of elements from `index` to the end of the Deque<T>.

-or-
`count` is greater than the available space from `arrayIndex` to the end of the destination `array`.

CopyTo(T[], int)

Copies the Deque<T> elements to an existing one-dimensional Array.

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

Parameters

`array` T[]

The one-dimensional Array that is the destination of the elements copied from Deque<T>.

`arrayIndex` int

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

Exceptions

ArgumentNullException

`array` is null.

ArgumentOutOfRangeException

`arrayIndex` is less than 0.

ArgumentException

The number of elements in the Deque<T> is greater than the available space from `arrayIndex` to the end of the destination `array`.

GetEnumerator()

Returns an enumerator that iterates through the Deque<T>.

```
public IEnumerator<T> GetEnumerator()
```

Returns

IEnumerator <T>

An IEnumerator<T> for the Deque<T>.

Remarks

If changes are made to the collection, the next call to MoveNext() throws an InvalidOperationException.

IndexOf(T)

Searches for the specified value and returns the zero-based index of the first occurrence within the Deque<T>.

```
public int IndexOf(T item)
```

Parameters

`item` T

The value to locate in the Deque<T>.

Returns

int

The zero-based index of the first occurrence of `item` within the Deque<T>, if found; otherwise, -1.

Remarks

This method determines equality using the default equality comparer Default.

Insert(int, T)

Inserts an element into the Deque<T> at the specified index.

```
public void Insert(int index, T item)
```

Parameters

`index` int

The zero-based index at which `item` should be inserted.

`item` T

The value to insert.

Exceptions

ArgumentOutOfRangeException

`index` is less than 0.

-or-
`index` is greater than Count.

InsertRange(int, IEnumerable<T>)

Inserts the elements of a collection into the Deque<T> at the specified index.

```
public void InsertRange(int index, IEnumerable<T> collection)
```

Parameters

`index` int

The zero-based index at which the new elements should be inserted.

`collection` IEnumerable<T>

The collection whose elements should be inserted into the Deque<T>.

Exceptions

ArgumentNullException

`collection` is null.

ArgumentOutOfRangeException

`index` is less than 0.

-or-
`index` is greater than Count.

Remove(T)

Removes the first occurrence of a specific element from the Deque<T>.

```
public bool Remove(T item)
```

Parameters

`item` T

The element to remove from the Deque<T>.

Returns

bool

true if `item` is successfully removed; otherwise, false.

RemoveAt(int)

Removes the element at the specified index of the Deque<T>.

```
public void RemoveAt(int index)
```

Parameters

`index` int

The zero-based index of the element to remove.

Exceptions

ArgumentOutOfRangeException

`index` is less than 0.

-or-
`index` is equal to or greater than Count.

RemoveFirst()

Removes the element at the start of the Deque<T>.

```
public void RemoveFirst()
```

Exceptions

InvalidOperationException

The Deque<T> is empty.

RemoveLast()

Removes the element at the end of the Deque<T>.

```
public void RemoveLast()
```

Exceptions

InvalidOperationException

The Deque<T> is empty.

RemoveRange(int, int)

Removes a range of elements from the Deque<T>.

```
public void RemoveRange(int index, int count)
```

Parameters

`index` int

The zero-based starting index of the range of elements to remove.

`count` int

The number of elements to remove.

Exceptions

ArgumentOutOfRangeException

`index` is less than 0.

-or-
`count` is less than 0.

ArgumentException

`index` and `count` do not denote a valid range of elements in the Deque<T>.

See Also

LinkedList <T>
Queue <T>

Namespace Biocs.IO

Classes

BgzfStream

Provides access to streams in the BGZF compression format.

Class BgzfStream

Namespace: Biocs.IO

Assembly: Biocs.Core.dll

Provides access to streams in the BGZF compression format.

```
public class BgzfStream : Stream, IAsyncDisposable, IDisposable
```

Inheritance

object ← MarshalByRefObject ← Stream ← BgzfStream

Implements

IAsyncDisposable, IDisposable

Inherited Members

Stream.Null , Stream.BeginRead(byte[], int, int, AsyncCallback, object) , Stream.BeginWrite(byte[], int, int, AsyncCallback, object) , Stream.Close() , Stream.CopyTo(Stream) , Stream.CopyTo(Stream, int) , Stream.CopyToAsync(Stream) , Stream.CopyToAsync(Stream, int) , Stream.CopyToAsync(Stream, int, CancellationToken) , Stream.CopyToAsync(Stream, CancellationToken) , Stream.CreateWaitHandle() , Stream.Dispose() , Stream.DisposeAsync() , Stream.EndRead(IAsyncResult) , Stream.EndWrite(IAsyncResult) , Stream.FlushAsync() , Stream.FlushAsync(CancellationToken) , Stream.ObjectInvariant() , Stream.ReadAsync(byte[], int, int) , Stream.ReadAsync(byte[], int, int, CancellationToken) , Stream.ReadAsync(Memory<byte>, CancellationToken) , Stream.ReadAtLeast(Span<byte>, int, bool) , Stream.ReadAtLeastAsync(Memory<byte>, int, bool, CancellationToken) , Stream.ReadByte() , Stream.ReadExactly(byte[], int, int) , Stream.ReadExactly(Span<byte>) , Stream.ReadExactlyAsync(byte[], int, int, CancellationToken) , Stream.ReadExactlyAsync(Memory<byte>, CancellationToken) , Stream.Synchronized(Stream) , Stream.ValidateBufferArguments(byte[], int, int) , Stream.ValidateCopyToArguments(Stream, int) , Stream.WriteAsync(byte[], int, int) , Stream.WriteAsync(byte[], int, int, CancellationToken) , Stream.WriteAsync(ReadOnlyMemory<byte>, CancellationToken) , Stream.WriteByte(byte) , Stream.CanTimeout , Stream.ReadTimeout , Stream.WriteTimeout , MarshalByRefObject.GetLifetimeService() , MarshalByRefObject.InitializeLifetimeService() , MarshalByRefObject.MemberwiseClone(bool) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

BgzfStream(Stream, CompressionLevel, bool)

Initializes a new instance of the BgzfStream class with the specified stream and compression level, and a value that specifies whether to leave the stream open.

```
public BgzfStream(Stream stream, CompressionLevel level, bool leaveOpen = false)
```

Parameters

`stream` Stream

The stream to compress.

`level` CompressionLevel

One of the CompressionLevel values that indicates whether to emphasize speed or compression size.

`leaveOpen` bool

true to leave the stream open; otherwise, false.

Exceptions

ArgumentNullException

`stream` is null.

ArgumentException

`level` is not a valid CompressionLevel enumeration value.

BgzfStream(Stream, CompressionMode, bool)

Initializes a new instance of the BgzfStream class with the specified stream and compression mode, and a value that specifies whether to leave the stream open.

```
public BgzfStream(Stream stream, CompressionMode mode, bool leaveOpen = false)
```

Parameters

`stream` Stream

The stream to compress or decompress.

`mode` CompressionMode

One of the CompressionMode values that indicates the action to take.

`leaveOpen` bool

true to leave the stream open; otherwise, false.

Remarks

The compression level is set to Optimal when the compression mode is Compress.

Exceptions

ArgumentNullException

`stream` is null.

ArgumentException

`mode` is not a valid CompressionMode enumeration value.

Properties

CanRead

Gets a value indicating whether the current stream supports reading.

```
public override bool CanRead { get; }
```

Property Value

bool

CanSeek

Gets a value indicating whether the current stream supports seeking.

```
public override bool CanSeek { get; }
```

Property Value

bool

CanWrite

Gets a value indicating whether the current stream supports writing.

```
public override bool CanWrite { get; }
```

Property Value

bool

Length

This property is not supported and always throws a NotSupportedException.

```
public override long Length { get; }
```

Property Value

long

Exceptions

NotSupportedException

This property is not supported on this stream.

Position

This property is not supported and always throws a NotSupportedException.

```
public override long Position { get; set; }
```

Property Value

long

Exceptions

NotSupportedException

This property is not supported on this stream.

Methods

Dispose(bool)

Releases the unmanaged resources used by the Stream and optionally releases the managed resources.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

true to release both managed and unmanaged resources; false to release only unmanaged resources.

Flush()

Writes any buffered data to the underlying stream.

```
public override void Flush()
```

Exceptions

IOException

An I/O error occurs.

NotSupportedException

The size of compressed bytes for a BGZF block exceeds about 64 KB.

IsBgzfFile(string?)

Determines whether the specified file is in the BGZF format.

```
public static bool IsBgzfFile(string? path)
```

Parameters

`path` string

The file to check.

Returns

bool

true if the specified file has the regular BGZF header; otherwise, false.

Read(byte[], int, int)

Reads a number of decompressed bytes from the underlying stream into the specified byte array.

```
public override int Read(byte[] buffer, int offset, int count)
```

Parameters

`buffer` byte[]

An array of bytes used to store decompressed bytes.

`offset` int

The zero-based byte offset in `buffer` at which to begin storing decompressed bytes.

`count` int

The maximum number of decompressed bytes to be read.

Returns

int

The total number of decompressed bytes read into the buffer. This can be less than `count` or zero if the end of the stream has been reached.

Exceptions

ArgumentNullException

`buffer` is null.

ArgumentOutOfRangeException

`offset` or `count` is negative.

-or-
The sum of `offset` and `count` is larger than the length of `buffer`.

InvalidDataException

The stream data is in an invalid BGZF format.

IOException

An I/O error occurs.

NotSupportedException

The stream does not support reading.

ObjectDisposedException

The method were called after the stream was closed.

Read(Span<byte>)

Reads a number of decompressed bytes from the underlying stream into the specified byte span.

```
public override int Read(Span<byte> buffer)
```

Parameters

`buffer` Span<byte>

A region of memory.

Returns

int

The total number of decompressed bytes read into the buffer. This can be less than the length of `buffer` or zero if the end of the stream has been reached.

Exceptions

InvalidDataException

The stream data is in an invalid BGZF format.

IOException

An I/O error occurs.

NotSupportedException

The stream does not support reading.

ObjectDisposedException

The method were called after the stream was closed.

Seek(long, SeekOrigin)

This method is not supported and always throws a NotSupportedException.

```
public override long Seek(long offset, SeekOrigin origin)
```

Parameters

`offset` long

`origin` SeekOrigin

Returns

long

Exceptions

NotSupportedException

This method is not supported on this stream.

SetLength(long)

This method is not supported and always throws a NotSupportedException.

```
public override void SetLength(long value)
```

Parameters

`value` long

Exceptions

NotSupportedException

This method is not supported on this stream.

Write(byte[], int, int)

Writes a sequence of compressed bytes to the underlying stream.

```
public override void Write(byte[] buffer, int offset, int count)
```

Parameters

`buffer` byte[]

An array of bytes to compress.

`offset` int

The zero-based byte offset in `buffer` at which to begin compressing.

`count` int

The number of bytes to be compress.

Exceptions

ArgumentNullException

`buffer` is null.

ArgumentOutOfRangeException

`offset` or `count` is negative.

-or-
The sum of `offset` and `count` is larger than the length of `buffer`.

IOException

An I/O error occurs.

NotSupportedException

The stream does not support writing.

-or-
The size of compressed bytes for a BGZF block exceeds about 64 KB.

ObjectDisposedException

The method were called after the stream was closed.

Write(ReadOnlySpan<byte>)

Writes a sequence of compressed bytes to the underlying stream.

```
public override void Write(ReadOnlySpan<byte> buffer)
```

Parameters

`buffer` ReadOnlySpan<byte>

A region of memory.

Exceptions

IOException

An I/O error occurs.

NotSupportedException

The stream does not support writing.

-or-
The size of compressed bytes for a BGZF block exceeds about 64 KB.

ObjectDisposedException

The method were called after the stream was closed.

Namespace Biocs.Numerics

Classes

DoubleMersenneTwister

Represents double-precision Mersenne Twister pseudorandom number generator based on IEEE 754 format.

Class DoubleMersenneTwister

Namespace: Biocs.Numerics

Assembly: Biocs.Core.dll

Represents double-precision Mersenne Twister pseudorandom number generator based on IEEE 754 format.

```
public class DoubleMersenneTwister
```

Inheritance

object ← DoubleMersenneTwister

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

For details about Mersenne Twister, see http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/.

Currently, the environment where the architecture is big-endian is not supported.

Constructors

DoubleMersenneTwister()

Initializes a new instance of the DoubleMersenneTwister class, using a time-dependent default seed value.

```
public DoubleMersenneTwister()
```

DoubleMersenneTwister(int)

Initializes a new instance of the DoubleMersenneTwister class, using the specified seed value.

```
public DoubleMersenneTwister(int seed)
```

Parameters

`seed` int

A 32-bit integer used as the seed.

DoubleMersenneTwister(int[])

Initializes a new instance of the DoubleMersenneTwister class, using the specified seed array.

```
public DoubleMersenneTwister(int[] seeds)
```

Parameters

`seeds` int[]

An array of 32-bit integers used as the seed.

Exceptions

ArgumentNullException

`seeds` is null.

Methods

Next()

Returns a double-precision pseudorandom number that distributes uniformly in the range [0, 1).

```
public double Next()
```

Returns

double

A random floating-point number that is greater than or equal to 0.0, and less than 1.0.

NextOpen()

Returns a double-precision pseudorandom number that distributes uniformly in the range (0, 1).

```
public double NextOpen()
```

Returns

double

A random floating-point number that is greater than 0.0, and less than 1.0.