Namespace BelNytheraSeiche.TrieDictionary

Classes

AATreeRecordStore

A concrete implementation of BinaryTreeRecordStore that uses a self-balancing AA tree.

AVLTreeRecordStore

A concrete implementation of BinaryTreeRecordStore that uses a self-balancing AVL tree.

BasicRecordStore

An abstract base class for key-value record stores.

BasicRecordStore.Record

Represents a single node in a linked list of records.

BasicRecordStore.RecordAccess

Provides access to and manages a linked list of BasicRecordStore.Record objects.

BasicRecordStore.SerializationOptions

Provides options to control the serialization process for key-value record stores.

BinaryTreeRecordStore

An abstract base class for record stores that use a binary search tree to organize records.

BitSet

Provides a mutable bit set for efficiently building large bit vectors.

BitwiseVectorDictionary

A concrete implementation of KeyRecordDictionary that uses a high-performance, array-based trie structure.

DirectedAcyclicGraphDictionary

A concrete implementation of KeyRecordDictionary that uses a Directed Acyclic Word Graph (DAWG).

DoubleArrayDictionary

A concrete implementation of KeyRecordDictionary that uses a mutable Double-Array trie structure.

HashMapRecordStore

A concrete implementation of BasicRecordStore that uses a hash map as its underlying data structure.

ImmutableBitSet

Represents a read-only, immutable bit set.

KeyRecordDictionary

An abstract base class for advanced key-record dictionaries.

KeyRecordDictionary.SerializationOptions
: Provides options to control the serialization process for dictionary data structures.

KeyRecordDictionary.StringSpecialized
: Provides a string-specialized wrapper around an KeyRecordDictionary.IKeyAccess instance, simplifying operations by handling string-to-byte encoding and decoding.

LevelOrderBitsDictionary
: An abstract base class for read-only, trie-based dictionaries that are constructed in level-order.

LoudsDictionary
: A concrete implementation of KeyRecordDictionary that uses a memory-efficient LOUDS (Level-Order Unary Degree Sequence) trie.

PrimitiveRecordStore
: Provides a low-level, memory-efficient data store for variable-length byte records.

PrimitiveRecordStore.Record
: Represents a handle to a single record within the store.

PrimitiveRecordStore.RecordAccess
: Provides access to a linked list of records within a PrimitiveRecordStore.

PrimitiveRecordStore.SerializationOptions
: Provides options to control the serialization process for data store.

RankSelectBitSet
: An immutable bit set optimized for high-performance Rank and Select operations.

ScapegoatTreeRecordStore
: A concrete implementation of BinaryTreeRecordStore that uses a self-balancing Scapegoat tree.

TreapRecordStore
: A concrete implementation of BinaryTreeRecordStore that uses a Treap.

ValueBuffer<T>
: Represents a buffer for value types, implemented using a list of array chunks.

Xoroshiro128PlusPlus
: Represents a 64-bit pseudo-random number generator based on the xoroshiro128++ algorithm.

Xoshiro256PlusPlus
: Represents a 64-bit pseudo-random number generator based on the xoshiro256++ algorithm.

Interfaces

KeyRecordDictionary.IKeyAccess

Defines the primary public contract for all key-based operations within the dictionary.

KeyRecordDictionary.IRecord

Defines a public contract for a single data record.

KeyRecordDictionary.IRecordAccess

Defines a public contract for accessing and managing a list of records associated with a single key.

Enums

KeyRecordDictionary.SearchDirectionType

Specifies the direction for key matching and sorting operations.

Class AATreeRecordStore

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of BinaryTreeRecordStore that uses a self-balancing AA tree.

```
public sealed class AATreeRecordStore : BinaryTreeRecordStore, ICloneable
```

Inheritance

object ← BasicRecordStore ← BinaryTreeRecordStore ← AATreeRecordStore

Implements

ICloneable

Inherited Members

BinaryTreeRecordStore.Clear() , BinaryTreeRecordStore.Contains(int) , BinaryTreeRecordStore.GetRecordAccess(int, out bool) , BinaryTreeRecordStore.TryGetRecordAccess(int, out BasicRecordStore.RecordAccess) , BinaryTreeRecordStore.Enumerate() , BinaryTreeRecordStore.PrintTree(TextWriter) , BasicRecordStore.Serialize<T>(T, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, string, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, Stream, BasicRecordStore.SerializationOptions) , BasicRecordStore.Deserialize<T>(byte[]) , BasicRecordStore.Deserialize<T>(string) , BasicRecordStore.Deserialize<T>(Stream) , BasicRecordStore.Add(int) , BasicRecordStore.GetRecordAccess(int) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class uses an AA tree, a variation of a Red-Black tree, to maintain balance. It simplifies the rebalancing logic by using two core operations, Skew and Split, to handle insertions and deletions efficiently.

Serialization Limits: The binary serialization format for this class imposes certain constraints on the data. Exceeding these limits will result in an InvalidDataException during serialization.

- **Max Records per List:** A single identifier can have a maximum of 16,777,215 records in its linked list.
- **Max Record Content Size:** The `Content` byte array of any single record cannot exceed 65,535 bytes.

Methods

Clone()

Creates a deep copy of the AATreeRecordStore.

```
public object Clone()
```

Returns

object

A new AATreeRecordStore instance with the same structure and record data as the original.

Remarks

The method creates a new tree and copies all records, ensuring that the new store is independent of the original.

Deserialize(Stream)

Deserializes an AATreeRecordStore from a stream.

```
public static AATreeRecordStore Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

AATreeRecordStore

A new instance of AATreeRecordStore reconstructed from the stream.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The stream data is corrupted, in an unsupported format, or contains invalid values.

IsBalanced()

Overrides the base method to check if the tree conforms to the specific balancing rules of an AA tree.

```
public override bool IsBalanced()
```

Returns

bool

true if the tree satisfies all tree invariants; otherwise, false.

Remarks

This method validates the level-based properties that define a valid tree.

Remove(int)

Removes the node with the specified identifier from the tree, performing rebalancing operations as necessary.

```
public override void Remove(int identifier)
```

Parameters

`identifier` int

The identifier of the node to remove.

Exceptions

ArgumentOutOfRangeException

`identifier` is MinValue.

Serialize(AATreeRecordStore, Stream, SerializationOptions?)

Serializes the entire state of the tree store into a stream.

```
public static void Serialize(AATreeRecordStore store, Stream stream,
BasicRecordStore.SerializationOptions? options = null)
```

Parameters

`store` AATreeRecordStore

The AATreeRecordStore instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` BasicRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is specific to this tree implementation, using Brotli compression and an XxHash32 checksum for data integrity.

Exceptions

ArgumentNullException

`store` or `stream` is null.

Class AVLTreeRecordStore

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of BinaryTreeRecordStore that uses a self-balancing AVL tree.

```
public sealed class AVLTreeRecordStore : BinaryTreeRecordStore, ICloneable
```

Inheritance

object ← BasicRecordStore ← BinaryTreeRecordStore ← AVLTreeRecordStore

Implements

ICloneable

Inherited Members

BinaryTreeRecordStore.Clear() , BinaryTreeRecordStore.Contains(int) , BinaryTreeRecordStore.GetRecordAccess(int, out bool) , BinaryTreeRecordStore.TryGetRecordAccess(int, out BasicRecordStore.RecordAccess) , BinaryTreeRecordStore.Enumerate() , BinaryTreeRecordStore.IsBalanced() , BinaryTreeRecordStore.PrintTree(TextWriter) , BasicRecordStore.Serialize<T>(T, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, string, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, Stream, BasicRecordStore.SerializationOptions) , BasicRecordStore.Deserialize<T>(byte[]) , BasicRecordStore.Deserialize<T>(string) , BasicRecordStore.Deserialize<T>(Stream) , BasicRecordStore.Add(int) , BasicRecordStore.GetRecordAccess(int) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class provides efficient key-based record lookups, insertions, and deletions by maintaining the balance of the tree according to AVL rules, ensuring logarithmic time complexity for these operations.

Serialization Limits: The binary serialization format for this class imposes certain constraints on the data. Exceeding these limits will result in an InvalidDataException during serialization.

- **Max Records per List:** A single identifier can have a maximum of 16,777,215 records in its linked list.
- **Max Record Content Size:** The `Content` byte array of any single record cannot exceed 65,535 bytes.

Methods

Clone()

Creates a deep copy of the AVLTreeRecordStore.

```
public object Clone()
```

Returns

object

A new AVLTreeRecordStore instance with the same structure and record data as the original.

Remarks

The method creates a new tree and copies all records, ensuring that the new store is independent of the original.

Deserialize(Stream)

Deserializes an AVLTreeRecordStore from a stream.

```
public static AVLTreeRecordStore Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

AVLTreeRecordStore

A new instance of AVLTreeRecordStore reconstructed from the stream.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The stream data is corrupted, in an unsupported format, or contains invalid values.

Remove(int)

Removes the node with the specified identifier from the tree, performing rebalancing rotations as necessary to maintain the tree's balance.

```
public override void Remove(int identifier)
```

Parameters

`identifier` int

The identifier of the node to remove.

Exceptions

ArgumentOutOfRangeException

`identifier` is MinValue.

Serialize(AVLTreeRecordStore, Stream, SerializationOptions?)

Serializes the entire state of the tree store into a stream.

```
public static void Serialize(AVLTreeRecordStore store, Stream stream,
BasicRecordStore.SerializationOptions? options = null)
```

Parameters

`store` AVLTreeRecordStore

The AVLTreeRecordStore instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` BasicRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is specific to this tree implementation, using Brotli compression and an XxHash32 checksum for data integrity.

Exceptions

ArgumentNullException

`store` or `stream` is null.

Class BasicRecordStore

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

An abstract base class for key-value record stores.

```
public abstract class BasicRecordStore
```

Inheritance

object ← BasicRecordStore

Derived

BinaryTreeRecordStore, HashMapRecordStore

Inherited Members

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

Remarks

This class defines a common interface for record manipulation (add, remove, enumerate, etc.) and provides a generic, reflection-based serialization mechanism that dispatches to the static methods of the concrete derived class.

Methods

Add(int)

Adds a record with the specified identifier, or returns false if it already exists.

```
public virtual bool Add(int identifier)
```

Parameters

`identifier` int

The identifier of the record to add.

Returns

bool

true if a new record was added; false if a record with that identifier already existed.

Clear()

When overridden in a derived class, removes all records from the store.

```
public virtual void Clear()
```

Contains(int)

When overridden in a derived class, determines whether a record with the specified identifier exists.

```
public virtual bool Contains(int identifier)
```

Parameters

`identifier` int

The identifier to locate.

Returns

bool

true if a record with the specified identifier is found; otherwise, false.

Deserialize<T>(byte[])

Deserializes a store from a byte array.

```
public static T Deserialize<T>(byte[] data) where T : BasicRecordStore
```

Parameters

`data` byte[]

The byte array containing the serialized data.

Returns

T

A new instance of `T`.

Type Parameters

`T`

The concrete type of the record store to deserialize, which must have a public static `Deserialize(Stream)` method.

Remarks

This static method uses reflection to invoke the static `Deserialize(Stream)` method on the specified type `T`.

Exceptions

ArgumentNullException

`data` is null.

Deserialize<T>(Stream)

Deserializes a store from a stream using reflection.

```
public static T Deserialize<T>(Stream stream) where T : BasicRecordStore
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

T

A new instance of `T`.

Type Parameters

`T`

The concrete type of the record store to deserialize, which must have a public static `Deserialize(Stream)` method.

Exceptions

ArgumentNullException

`stream` is null.

NotSupportedException

The type `T` does not have a public static `Deserialize(Stream)` method.

Deserialize<T>(string)

Deserializes a store from a file.

```
public static T Deserialize<T>(string file) where T : BasicRecordStore
```

Parameters

`file` string

The path of the file to read from.

Returns

T

A new instance of `T`.

Type Parameters

`T`

The concrete type of the record store to deserialize, which must have a public static `Deserialize(Stream)` method.

Remarks

This static method uses reflection to invoke the static `Deserialize(Stream)` method on the specified type `T`.

Exceptions

ArgumentNullException

`file` is null.

ArgumentException

`file` is empty or whitespace.

Enumerate()

When overridden in a derived class, returns an enumerator that iterates through all records in the store.

```
public virtual IEnumerable<(int, BasicRecordStore.RecordAccess)> Enumerate()
```

Returns

IEnumerable<(int, BasicRecordStore.RecordAccess)>

An IEnumerable<T> of tuples, each containing an identifier and its corresponding BasicRecordStore.RecordAccess.

GetRecordAccess(int)

Gets the accessor for a record with the specified identifier. If the record does not exist, it is created.

```
public virtual BasicRecordStore.RecordAccess GetRecordAccess(int identifier)
```

Parameters

`identifier` int

The identifier of the record to get or create.

Returns

BasicRecordStore.RecordAccess

A BasicRecordStore.RecordAccess for the found or newly created record.

GetRecordAccess(int, out bool)

When overridden in a derived class, gets the accessor for a record with the specified identifier, indicating if it was newly created.

```
public virtual BasicRecordStore.RecordAccess GetRecordAccess(int identifier, out
bool createNew)
```

Parameters

`identifier` int

The identifier of the record to get or create.

`createNew` bool

When this method returns, contains true if a new record was created; otherwise, false.

Returns

BasicRecordStore.RecordAccess

A BasicRecordStore.RecordAccess for the found or newly created record.

Remove(int)

When overridden in a derived class, removes a record with the specified identifier.

```
public virtual void Remove(int identifier)
```

Parameters

`identifier` int

The identifier of the record to remove.

Serialize<T>(T, SerializationOptions?)

Serializes the specified store instance into a byte array.

```
public static byte[] Serialize<T>(T obj, BasicRecordStore.SerializationOptions? options =
null) where T : BasicRecordStore
```

Parameters

`obj` T

The record store instance to serialize.

`options` BasicRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Returns

byte[]

A byte array containing the serialized data.

Type Parameters

`T`

The concrete type of the record store, which must have a public static `Serialize(T, Stream)` method.

Remarks

This static method uses reflection to invoke the static `Serialize(T, Stream)` method on the actual type of `obj`.

Exceptions

ArgumentNullException

`obj` is null.

Serialize<T>(T, Stream, SerializationOptions?)

Serializes the specified store instance into a stream using reflection.

```
public static void Serialize<T>(T obj, Stream stream, BasicRecordStore.SerializationOptions?
options = null) where T : BasicRecordStore
```

Parameters

`obj` T

The record store instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` BasicRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Type Parameters

`T`

The concrete type of the record store, which must have a public static `Serialize(T, Stream)` method.

Exceptions

ArgumentNullException

`obj` or `stream` is null.

NotSupportedException

The type `T` does not have a public static `Serialize(T, Stream)` method.

Serialize<T>(T, string, SerializationOptions?)

Serializes the specified store instance to a file.

```
public static void Serialize<T>(T obj, string file, BasicRecordStore.SerializationOptions?
options = null) where T : BasicRecordStore
```

Parameters

`obj` T

The record store instance to serialize.

`file` string

The path of the file to create.

`options` BasicRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Type Parameters

`T`

The concrete type of the record store, which must have a public static `Serialize(T, Stream)` method.

Remarks

This static method uses reflection to invoke the static `Serialize(T, Stream)` method on the actual type of `obj`.

Exceptions

ArgumentNullException

`obj` or `file` is null.

ArgumentException

`file` is empty or whitespace.

TryGetRecordAccess(int, out RecordAccess?)

When overridden in a derived class, tries to get the accessor for a record with the specified identifier.

```
public virtual bool TryGetRecordAccess(int identifier, out BasicRecordStore.RecordAccess?
access)
```

Parameters

`identifier` int

The identifier to locate.

`access` BasicRecordStore.RecordAccess

When this method returns, contains the BasicRecordStore.RecordAccess object if the identifier was found; otherwise, null.

Returns

bool

true if a record with the specified identifier was found; otherwise, false.

Class BasicRecordStore.Record

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Represents a single node in a linked list of records.

```
public sealed class BasicRecordStore.Record
```

Inheritance

object ← BasicRecordStore.Record

Inherited Members

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

Properties

Content

Gets or sets the byte array content of this record.

```
public byte[] Content { get; set; }
```

Property Value

byte[]

Exceptions

ArgumentNullException

The assigned value is null.

Next

Gets the next record in the list, or null if this is the last record.

```
public BasicRecordStore.Record? Next { get; }
```

Property Value

BasicRecordStore.Record

Methods

InsertAfter(ReadOnlySpan<byte>)

Inserts a new record with the specified content immediately after this record in the list.

```
public BasicRecordStore.Record InsertAfter(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record to insert.

Returns

BasicRecordStore.Record

The newly inserted BasicRecordStore.Record.

InsertBefore(ReadOnlySpan<byte>)

Inserts a new record with the specified content immediately before this record in the list.

```
public BasicRecordStore.Record InsertBefore(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record to insert.

Returns

BasicRecordStore.Record

The newly inserted BasicRecordStore.Record.

Remove()

Removes this record from the list it belongs to.

```
public void Remove()
```

Class BasicRecordStore.RecordAccess

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Provides access to and manages a linked list of BasicRecordStore.Record objects.

```
public sealed class BasicRecordStore.RecordAccess : IEnumerable<BasicRecordStore.Record>,
IEnumerable
```

Inheritance

object ← BasicRecordStore.RecordAccess

Implements

IEnumerable<BasicRecordStore.Record>, IEnumerable

Inherited Members

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

Remarks

This class implements IEnumerable<T> to allow easy iteration. The collection will throw an InvalidOperationException if modified during enumeration.

Properties

Identifier

Gets the identifier for the head of the linked list that this object accesses.

```
public int Identifier { get; }
```

Property Value

int

Methods

Add(ReadOnlySpan<byte>)

Adds a new record with the specified content to the end of the list.

```
public BasicRecordStore.Record Add(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record.

Returns

BasicRecordStore.Record

The newly added BasicRecordStore.Record.

Clear()

Removes all records from the list.

```
public void Clear()
```

GetEnumerator()

Returns an enumerator that iterates through the records in the list.

```
public IEnumerator<BasicRecordStore.Record> GetEnumerator()
```

Returns

IEnumerator<BasicRecordStore.Record>

An enumerator for this record list.

Class BasicRecordStore.SerializationOptions

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Provides options to control the serialization process for key-value record stores.

```
public class BasicRecordStore.SerializationOptions
```

Inheritance

object ← BasicRecordStore.SerializationOptions

Inherited Members

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

Properties

CompressionLevel

Gets or sets the compression level to use for serialization.

```
public CompressionLevel CompressionLevel { get; set; }
```

Property Value

CompressionLevel

Remarks

Higher compression levels can result in smaller file sizes but may take longer to process. Defaults to Fastest for a balance of speed and size.

Default

Gets a singleton instance of BasicRecordStore.SerializationOptions with default values.

```
public static BasicRecordStore.SerializationOptions Default { get; }
```

Property Value

BasicRecordStore.SerializationOptions

Remarks

Use this property to avoid creating a new options object when default settings are sufficient.

Class BinaryTreeRecordStore

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

An abstract base class for record stores that use a binary search tree to organize records.

```
public abstract class BinaryTreeRecordStore : BasicRecordStore
```

Inheritance

object ← BasicRecordStore ← BinaryTreeRecordStore

Derived

AATreeRecordStore, AVLTreeRecordStore, ScapegoatTreeRecordStore, TreapRecordStore

Inherited Members

BasicRecordStore.Serialize<T>(T, BasicRecordStore.SerializationOptions) ,
BasicRecordStore.Serialize<T>(T, string, BasicRecordStore.SerializationOptions) ,
BasicRecordStore.Serialize<T>(T, Stream, BasicRecordStore.SerializationOptions) ,
BasicRecordStore.Deserialize<T>(byte[]) , BasicRecordStore.Deserialize<T>(string) ,
BasicRecordStore.Deserialize<T>(Stream) , BasicRecordStore.Remove(int) , BasicRecordStore.Add(int) ,
BasicRecordStore.GetRecordAccess(int) , object.Equals(object) , object.Equals(object, object) ,
object.GetHashCode() , object.GetType() , object.MemberwiseClone() ,
object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class provides the core logic for searching, enumerating, and manipulating a binary tree structure where nodes are keyed by an integer identifier. Derived classes are expected to implement the specific insertion and balancing logic (e.g., for an AVL or Red-Black tree).

Methods

Clear()

Removes all nodes from the tree, effectively clearing the store.

```
public override void Clear()
```

Contains(int)

Determines whether a record with the specified identifier exists in the store.

```
public override bool Contains(int identifier)
```

Parameters

`identifier` int

The identifier to locate in the tree.

Returns

bool

true if a record with the specified identifier is found; otherwise, false.

Exceptions

ArgumentOutOfRangeException

`identifier` is equal to MinValue, which is reserved.

Enumerate()

Returns an enumerator that iterates through all records in the store in ascending order of their identifiers.

```
public override IEnumerable<(int, BasicRecordStore.RecordAccess)> Enumerate()
```

Returns

IEnumerable<(int, BasicRecordStore.RecordAccess)>

An IEnumerable<T> of tuples, each containing the identifier and BasicRecordStore.RecordAccess for a record.

Remarks

The enumeration is performed using an in-order traversal of the binary tree.

GetRecordAccess(int, out bool)

Gets the accessor for a record with the specified identifier. If the record does not exist, it is created.

```
public override BasicRecordStore.RecordAccess GetRecordAccess(int identifier, out
bool createNew)
```

Parameters

`identifier` int

The identifier of the record to get or create.

`createNew` bool

When this method returns, contains true if a new record was created; otherwise, false.

Returns

BasicRecordStore.RecordAccess

A BasicRecordStore.RecordAccess for the found or newly created record.

IsBalanced()

Checks if the binary tree is balanced.

```
public virtual bool IsBalanced()
```

Returns

bool

true if the tree is balanced; otherwise, false.

Remarks

The base implementation validates if the tree is height-balanced, meaning the height difference between the left and right subtrees of any node is no more than 1 (consistent with AVL tree properties). Derived classes that implement different balancing strategies (e.g., Red-Black Trees) should override this method

to provide their own validation logic. This can be a computationally expensive operation, primarily intended for debugging and validation.

PrintTree(TextWriter)

Prints a visual representation of the tree structure to the specified TextWriter.

```
public void PrintTree(TextWriter textWriter)
```

Parameters

`textWriter` TextWriter

The TextWriter to output the tree structure to.

Exceptions

ArgumentNullException

`textWriter` is null.

TryGetRecordAccess(int, out RecordAccess?)

Tries to get the accessor for a record with the specified identifier.

```
public override bool TryGetRecordAccess(int identifier, out BasicRecordStore.RecordAccess?
access)
```

Parameters

`identifier` int

The identifier to locate.

`access` BasicRecordStore.RecordAccess

When this method returns, contains the BasicRecordStore.RecordAccess object if the identifier was found; otherwise, null.

Returns

bool

true if a record with the specified identifier was found; otherwise, false.

Exceptions

ArgumentOutOfRangeException

`identifier` is equal to MinValue, which is reserved.

Class BitSet

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Provides a mutable bit set for efficiently building large bit vectors.

```
public sealed class BitSet
```

Inheritance

object ← BitSet

Inherited Members

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

Constructors

BitSet(int)

Provides a mutable bit set for efficiently building large bit vectors.

```
public BitSet(int arrayAlignment = 65536)
```

Parameters

`arrayAlignment` int

The alignment size for the internal buffer chunks. Must be a power of 2. Defaults to 65536.

Properties

ArrayAlignment

Gets the alignment size used by the internal buffer.

```
public int ArrayAlignment { get; }
```

Property Value

int

Methods

Add(bool)

Adds a bit to the end of the bit set.

```
public void Add(bool bit)
```

Parameters

`bit` bool

The bit to add (`true` for 1, `false` for 0).

Get(int)

Gets the bit value at the specified index.

```
public bool Get(int index)
```

Parameters

`index` int

The zero-based index of the bit to get.

Returns

bool

The bit value (`true` for 1, `false` for 0) at the specified index.

Exceptions

ArgumentOutOfRangeException

`index` is less than 0 or greater than or equal to the number of bits in the set.

ToImmutable()

Creates an immutable, read-only version of this bit set.

```
public ImmutableBitSet ToImmutable()
```

Returns

ImmutableBitSet

A new ImmutableBitSet containing the current data.

ToImmutable(BitSet)

Creates an immutable, read-only version of a specified BitSet.

```
public static ImmutableBitSet ToImmutable(BitSet bitSet)
```

Parameters

`bitSet` BitSet

The mutable bit set to convert.

Returns

ImmutableBitSet

A new ImmutableBitSet containing the data from the provided bit set.

Exceptions

ArgumentNullException

`bitSet` is null.

ToRankSelect(BitSet, bool)

Creates an immutable version of a specified BitSet, optimized for high-performance Rank and Select operations.

```
public static RankSelectBitSet ToRankSelect(BitSet bitSet, bool createAuxDir = true)
```

Parameters

`bitSet` BitSet

The mutable bit set to convert.

`createAuxDir` bool

A value indicating whether to immediately create the auxiliary directories required for fast Rank and Select operations.

Returns

RankSelectBitSet

A new RankSelectBitSet containing the data from the provided bit set and pre-calculated auxiliary indexes.

Remarks

If `createAuxDir` is set to `false`, this method is very fast, but the returned RankSelectBitSet will not be ready for Rank/Select operations until its auxiliary directories are created and set via the `SetAuxDir()` method. If set to `true` (the default), the auxiliary directories are created during this call, which takes more processing time but results in a fully initialized and ready-to-use object.

Exceptions

ArgumentNullException

`bitSet` is null.

ToRankSelect(bool)

Creates an immutable version of a specified BitSet, optimized for high-performance Rank and Select operations.

```
public RankSelectBitSet ToRankSelect(bool createAuxDir = true)
```

Parameters

`createAuxDir` bool

A value indicating whether to immediately create the auxiliary directories required for fast Rank and Select operations.

Returns

RankSelectBitSet

A new RankSelectBitSet containing the data from the provided bit set and pre-calculated auxiliary indexes.

Remarks

If `createAuxDir` is set to `false`, this method is very fast, but the returned RankSelectBitSet will not be ready for Rank/Select operations until its auxiliary directories are created and set via the `SetAuxDir()` method. If set to `true` (the default), the auxiliary directories are created during this call, which takes more processing time but results in a fully initialized and ready-to-use object.

Class BitwiseVectorDictionary

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of KeyRecordDictionary that uses a high-performance, array-based trie structure.

```
public abstract class BitwiseVectorDictionary : LevelOrderBitsDictionary,
KeyRecordDictionary.IKeyAccess
```

Inheritance

object ← KeyRecordDictionary ← LevelOrderBitsDictionary ← BitwiseVectorDictionary

Implements

KeyRecordDictionary.IKeyAccess

Inherited Members

LevelOrderBitsDictionary.AsStringSpecialized(Encoding) ,
LevelOrderBitsDictionary.GetRecordAccess(int, bool) ,
LevelOrderBitsDictionary.Contains(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.SearchExactly(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.SearchCommonPrefix(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.SearchLongestPrefix(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.SearchByPrefix(ReadOnlySpan<byte>, bool) ,
LevelOrderBitsDictionary.EnumerateAll(bool) ,
LevelOrderBitsDictionary.SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool) ,
LevelOrderBitsDictionary.FindFirst(out int, out byte[]) ,
LevelOrderBitsDictionary.FindLast(out int, out byte[]) ,
LevelOrderBitsDictionary.FindNext(int, out int, out byte[]) ,
LevelOrderBitsDictionary.FindNext(ReadOnlySpan<byte>, out int, out byte[]) ,
LevelOrderBitsDictionary.FindPrevious(int, out int, out byte[]) ,
LevelOrderBitsDictionary.FindPrevious(ReadOnlySpan<byte>, out int, out byte[]) ,
LevelOrderBitsDictionary.TryGetKey(int, out byte[]) , LevelOrderBitsDictionary.GetKey(int) ,
LevelOrderBitsDictionary.Add(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.TryAdd(ReadOnlySpan<byte>, out int) , LevelOrderBitsDictionary.Remove(int) ,
LevelOrderBitsDictionary.Remove(ReadOnlySpan<byte>) , KeyRecordDictionary.SearchDirection ,
KeyRecordDictionary.Additional1 , KeyRecordDictionary.Additional2 ,
KeyRecordDictionary.Create<T>(KeyRecordDictionary.SearchDirectionType) ,
KeyRecordDictionary.Create<T>(IEnumerable<byte[]>, KeyRecordDictionary.SearchDirectionType) ,

KeyRecordDictionary.Create<T>(IEnumerable<string>, Encoding, KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Serialize<T>(T, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, string, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, Stream, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Deserialize<T>(byte[]) , KeyRecordDictionary.Deserialize<T>(string) , KeyRecordDictionary.Deserialize<T>(Stream) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class represents a trie data structure that is flattened into a series of arrays using a level-order (breadth-first) build process. The tree's topology is stored using integer arrays for parent/child relationships, while a bit vector is used to mark terminal nodes. This approach prioritizes lookup performance and a relatively straightforward implementation over the extreme memory compactness of structures like LOUDS.

Important Usage Notes:

- **Read-Only Structure:** The dictionary is built once from a collection of keys and is effectively read-only afterward. Key addition and removal operations are not supported and will throw a NotSupportedException.
- **Key Reconstruction:** Getting a key by its identifier requires traversing the structure from a node up to the root.

Methods

Create(IEnumerable<byte[]>, SearchDirectionType)

Creates a new BitwiseVectorDictionary from a collection of keys.

```
public static BitwiseVectorDictionary Create(IEnumerable<byte[]> keys,
KeyRecordDictionary.SearchDirectionType searchDirection = SearchDirectionType.LTR)
```

Parameters

`keys` IEnumerable<byte[]>

An enumerable collection of byte arrays to use as the initial keys. The collection does not need to be pre-sorted. Duplicate keys are permitted, but only the first occurrence will be added. The collection

must not contain any elements that are null or empty byte array.

`searchDirection` KeyRecordDictionary.SearchDirectionType

The key search direction for the new dictionary.

Returns

BitwiseVectorDictionary

A new, optimized, and read-only BitwiseVectorDictionary instance.

Remarks

The input keys are sorted and used to construct the Bitwise-Vector trie representation.

Exceptions

ArgumentNullException

`keys` is null.

Deserialize(Stream)

Deserializes a dictionary from a stream.

```
public static BitwiseVectorDictionary Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

BitwiseVectorDictionary

A new instance of BitwiseVectorDictionary.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The stream data is corrupted or in an unsupported format.

Serialize(BitwiseVectorDictionary, Stream, SerializationOptions?)

Serializes the state of the dictionary into a stream.

```
public static void Serialize(BitwiseVectorDictionary dictionary, Stream stream,
KeyRecordDictionary.SerializationOptions? options = null)
```

Parameters

`dictionary` BitwiseVectorDictionary

The dictionary instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` KeyRecordDictionary.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is a custom binary format specific to this Bitwise-Vector implementation.

Exceptions

ArgumentNullException

`dictionary` or `stream` is null.

Class DirectedAcyclicGraphDictionary

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of KeyRecordDictionary that uses a Directed Acyclic Word Graph (DAWG).

```
public abstract class DirectedAcyclicGraphDictionary : KeyRecordDictionary,
KeyRecordDictionary.IKeyAccess
```

Inheritance

object ← KeyRecordDictionary ← DirectedAcyclicGraphDictionary

Implements

KeyRecordDictionary.IKeyAccess

Inherited Members

KeyRecordDictionary.SearchDirection , KeyRecordDictionary.Additional1 , KeyRecordDictionary.Additional2 , KeyRecordDictionary.Create<T>(KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Create<T>(IEnumerable<byte[]>, KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Create<T>(IEnumerable<string>, Encoding, KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Serialize<T>(T, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, string, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, Stream, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Deserialize<T>(byte[]) , KeyRecordDictionary.Deserialize<T>(string) , KeyRecordDictionary.Deserialize<T>(Stream) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class stores keys in a DAWG, which is a data structure that compresses a standard Trie by merging common prefixes and suffixes to reduce redundancy.

Important Usage Notes:

- **Read-Only Structure:** The dictionary is built once from a collection of keys and is effectively read-only afterward. The `Add` and `Remove` operations for keys are not supported and will throw a NotSupportedException.

Methods

Add(ReadOnlySpan<byte>)

This operation is not supported. The dictionary is read-only after creation.

```
public int Add(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

Returns

int

Exceptions

NotSupportedException

Always thrown.

AsStringSpecialized(Encoding?)

Returns a string-specialized wrapper for the dictionary's key access interface.

```
public KeyRecordDictionary.StringSpecialized AsStringSpecialized(Encoding? encoding = null)
```

Parameters

`encoding` Encoding

The encoding to use for string conversions. Defaults to UTF-8.

Returns

KeyRecordDictionary.StringSpecialized

A new KeyRecordDictionary.StringSpecialized instance providing string-based access to the dictionary.

Contains(ReadOnlySpan<byte>)

Determines whether the specified key exists in the graph.

```
public virtual bool Contains(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

The key to locate.

Returns

bool

true if the key is found; otherwise, false.

Create(IEnumerable<byte[]>, SearchDirectionType)

Creates a new DirectedAcyclicGraphDictionary from a collection of keys.

```
public static DirectedAcyclicGraphDictionary Create(IEnumerable<byte[]> keys,
KeyRecordDictionary.SearchDirectionType searchDirection = SearchDirectionType.LTR)
```

Parameters

`keys` IEnumerable<byte[]>

An enumerable collection of byte arrays to use as the initial keys. The collection does not need to be pre-sorted. Duplicate keys are permitted, but only the first occurrence will be added. The collection must not contain any elements that are null or empty byte array.

`searchDirection` KeyRecordDictionary.SearchDirectionType

The key search direction for the new dictionary.

Returns

DirectedAcyclicGraphDictionary

A new, optimized, and read-only DirectedAcyclicGraphDictionary instance.

Remarks

The input keys are sorted and used to build a minimal, compressed graph structure.

Exceptions

ArgumentNullException

`keys` is null.

Deserialize(Stream)

Deserializes a dictionary from a stream.

```
public static DirectedAcyclicGraphDictionary Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

DirectedAcyclicGraphDictionary

A new instance of DirectedAcyclicGraphDictionary.

Exceptions

InvalidDataException

The stream data is corrupted or in an unsupported format.

ArgumentNullException

`stream` is null.

EnumerateAll(bool)

Enumerates all keys in the dictionary in lexicographical order.

```
public virtual IEnumerable<(int, byte[])> EnumerateAll(bool reverse = false)
```

Parameters

`reverse` bool

If true, returns keys in reverse lexicographical order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of all (identifier, key) tuples.

FindFirst(out int, out byte[])

Finds the first key in the dictionary in lexicographical order.

```
public virtual bool FindFirst(out int identifier, out byte[] key)
```

Parameters

`identifier` int

When this method returns, the identifier of the first key.

`key` byte[]

When this method returns, the first key.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindLast(out int, out byte[])

Finds the last key in the dictionary in lexicographical order.

```
public virtual bool FindLast(out int identifier, out byte[] key)
```

Parameters

`identifier` int

When this method returns, the identifier of the last key.

`key` byte[]

When this method returns, the last key.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindNext(int, out int, out byte[])

Finds the next key in lexicographical order after the specified identifier.

```
public virtual bool FindNext(int currentIdentifier, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` byte[]

When this method returns, the next key.

Returns

bool

true if a next key was found; otherwise, false.

FindNext(ReadOnlySpan<byte>, out int, out byte[])

Finds the next key in lexicographical order after the specified key.

```
public virtual bool FindNext(ReadOnlySpan<byte> currentKey, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentKey` ReadOnlySpan<byte>

The key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` byte[]

When this method returns, the next key.

Returns

bool

true if a next key was found; otherwise, false.

FindPrevious(int, out int, out byte[])

Finds the previous key in lexicographical order before the specified identifier.

```
public virtual bool FindPrevious(int currentIdentifier, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` byte[]

When this method returns, the previous key.

Returns

bool

true if a previous key was found; otherwise, false.

FindPrevious(ReadOnlySpan<byte>, out int, out byte[])

Finds the previous key in lexicographical order before the specified key.

```
public virtual bool FindPrevious(ReadOnlySpan<byte> currentKey, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentKey` ReadOnlySpan<byte>

The key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` byte[]

When this method returns, the previous key.

Returns

bool

true if a previous key was found; otherwise, false.

GetKey(int)

Gets the key associated with the specified identifier.

```
public virtual byte[] GetKey(int identifier)
```

Parameters

`identifier` int

The identifier of the key to get.

Returns

byte[]

The key as a byte array.

Exceptions

ArgumentException

The specified identifier does not exist.

GetRecordAccess(int, bool)

Gets an accessor for the list of records associated with a given key identifier.

```
public KeyRecordDictionary.IRecordAccess GetRecordAccess(int identifier, bool isTransient
= false)
```

Parameters

`identifier` int

The identifier of the key whose records are to be accessed. This must be a valid key identifier obtained from a search or add operation.

`isTransient` bool

If true, accesses the transient (in-memory) record store; otherwise, accesses the persistent record store.

Returns

KeyRecordDictionary.IRecordAccess

An KeyRecordDictionary.IRecordAccess handle for the specified record list.

Remarks

Note that while new records can be added to a key's record list, new keys cannot be added to the dictionary itself after creation.

Exceptions

ArgumentOutOfRangeException

`identifier` is negative.

ArgumentException

The specified `identifier` is invalid or does not exist.

Remove(int)

This operation is not supported. The dictionary is read-only after creation.

```
public bool Remove(int identifier)
```

Parameters

`identifier` int

Returns

bool

Exceptions

NotSupportedException

Always thrown.

Remove(ReadOnlySpan<byte>)

This operation is not supported. The dictionary is read-only after creation.

```
public bool Remove(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

Returns

bool

Exceptions

NotSupportedException

Always thrown.

SearchByPrefix(ReadOnlySpan<byte>, bool)

Finds all keys that start with the given prefix.

```
public virtual IEnumerable<(int, byte[])> SearchByPrefix(ReadOnlySpan<byte> sequence, bool
reverse = false)
```

Parameters

`sequence` ReadOnlySpan<byte>

The prefix to search for.

`reverse` bool

If true, returns results in reverse lexicographical order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of (identifier, key) tuples for all keys starting with the prefix.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchByPrefix(ReadOnlySpan<byte>, bool).

SearchCommonPrefix(ReadOnlySpan<byte>)

Finds all keys in the dictionary that are prefixes of the given sequence.

```
public virtual IEnumerable<(int, byte[])> SearchCommonPrefix(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The sequence to search within.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of (identifier, key) tuples for all matching prefixes.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchCommonPrefix(ReadOnlySpan<byte>).

SearchExactly(ReadOnlySpan<byte>)

Searches for an exact match of the given sequence and returns its identifier.

```
public virtual int SearchExactly(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The key to search for.

Returns

int

The positive identifier for the key if found; otherwise, a non-positive value.

SearchLongestPrefix(ReadOnlySpan<byte>)

Finds the longest key in the dictionary that is a prefix of the given sequence.

```
public virtual (int, byte[]) SearchLongestPrefix(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The sequence to search within.

Returns

(int, byte[])

An enumerable collection of (identifier, key) tuples for all matching prefixes.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchLongestPrefix(ReadOnlySpan<byte>).

SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>,

bool)

Performs a wildcard search for keys matching a pattern.

```
public virtual IEnumerable<(int, byte[])> SearchWildcard(ReadOnlySpan<byte> sequence,
ReadOnlySpan<char> cards, bool reverse = false)
```

Parameters

`sequence` ReadOnlySpan<byte>

The byte sequence of the pattern.

`cards` ReadOnlySpan<char>

A sequence of characters ('?' for single, '*' for multiple wildcards) corresponding to the pattern.

`reverse` bool

If true, returns results in reverse order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of matching (identifier, key) tuples.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool).

Serialize(DirectedAcyclicGraphDictionary, Stream, SerializationOptions?)

Serializes the state of the dictionary into a stream.

```
public static void Serialize(DirectedAcyclicGraphDictionary dictionary, Stream stream,
KeyRecordDictionary.SerializationOptions? options = null)
```

Parameters

`dictionary` DirectedAcyclicGraphDictionary

The dictionary instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` KeyRecordDictionary.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is a custom binary format specific to this graph implementation.

Exceptions

ArgumentNullException

`dictionary` or `stream` is null.

TryAdd(ReadOnlySpan<byte>, out int)

This operation is not supported. The dictionary is read-only after creation.

```
public bool TryAdd(ReadOnlySpan<byte> key, out int identifier)
```

Parameters

`key` ReadOnlySpan<byte>

`identifier` int

Returns

bool

Exceptions

NotSupportedException

Always thrown.

TryGetKey(int, out byte[])

Tries to get the key associated with the specified identifier.

```
public virtual bool TryGetKey(int identifier, out byte[] key)
```

Parameters

`identifier` int

The identifier of the key to get.

`key` byte[]

When this method returns, contains the key if found; otherwise, an empty array.

Returns

bool

true if the key was found; otherwise, false.

Class DoubleArrayDictionary

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of KeyRecordDictionary that uses a mutable Double-Array trie structure.

```
public abstract class DoubleArrayDictionary : KeyRecordDictionary,
KeyRecordDictionary.IKeyAccess
```

Inheritance

object ← KeyRecordDictionary ← DoubleArrayDictionary

Implements

KeyRecordDictionary.IKeyAccess

Inherited Members

KeyRecordDictionary.SearchDirection , KeyRecordDictionary.Additional1 , KeyRecordDictionary.Additional2 , KeyRecordDictionary.Create<T>(KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Create<T>(IEnumerable<byte[]>, KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Create<T>(IEnumerable<string>, Encoding, KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Serialize<T>(T, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, string, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, Stream, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Deserialize<T>(byte[]) , KeyRecordDictionary.Deserialize<T>(string) , KeyRecordDictionary.Deserialize<T>(Stream) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class provides very fast key lookups by representing a trie data structure in two compact arrays ('BASE' and 'CHECK').

Important Usage Notes:

- **Mutable Structure:** Unlike the DAWG or LOUDS implementations, this dictionary is mutable. Keys can be added and removed after the initial creation.

- **Fragmentation:** Repeated additions and removals can lead to fragmentation of the internal arrays, which may degrade performance and increase memory usage over time. To eliminate fragmentation and optimize the structure, it is recommended to periodically rebuild the dictionary using the static Compact(DoubleArrayDictionary) method.

Methods

Add(ReadOnlySpan<byte>)

Adds a key to the dictionary. If the key already exists, its existing identifier is returned.

```
public virtual int Add(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

The key to add.

Returns

int

The identifier for the added or existing key.

Exceptions

ArgumentException

`key` is empty.

AsStringSpecialized(Encoding?)

Returns a string-specialized wrapper for the dictionary's key access interface.

```
public KeyRecordDictionary.StringSpecialized AsStringSpecialized(Encoding? encoding = null)
```

Parameters

`encoding` Encoding

The encoding to use for string conversions. Defaults to UTF-8.

Returns

KeyRecordDictionary.StringSpecialized

A new KeyRecordDictionary.StringSpecialized instance providing string-based access to the dictionary.

Clear()

Removes all keys and records from the dictionary and resets its internal state.

```
public void Clear()
```

Compact(DoubleArrayDictionary)

Rebuilds the dictionary to eliminate fragmentation and optimize its internal structure.

```
public static DoubleArrayDictionary Compact(DoubleArrayDictionary original)
```

Parameters

`original` DoubleArrayDictionary

The original, potentially fragmented dictionary to compact.

Returns

DoubleArrayDictionary

A new, compacted DoubleArrayDictionary instance containing the same keys and records.

Remarks

This method should be used when performance or memory usage degrades after a large number of additions and removals. It works by creating a new dictionary from the keys of the original, effectively rebuilding the internal arrays.

Important: The integer identifiers assigned to keys in the new, compacted dictionary will be different from those in the original. Any external references to the old identifiers will be invalid after compaction.

Exceptions

ArgumentNullException

`original` is null.

Contains(ReadOnlySpan<byte>)

Determines whether the specified key exists in the dictionary.

```
public virtual bool Contains(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

The key to locate.

Returns

bool

true if the key is found; otherwise, false.

Create(IEnumerable<byte[]>, SearchDirectionType)

Creates a new DoubleArrayDictionary from a collection of keys.

```
public static DoubleArrayDictionary Create(IEnumerable<byte[]> keys,
KeyRecordDictionary.SearchDirectionType searchDirection = SearchDirectionType.LTR)
```

Parameters

`keys` IEnumerable<byte[]>

An enumerable collection of byte arrays to use as the initial keys. The collection does not need to be pre-sorted. Duplicate keys are permitted, but only the first occurrence will be added. The collection must not contain any elements that are null or empty byte array.

`searchDirection` KeyRecordDictionary.SearchDirectionType

The key search direction for the new dictionary.

Returns

DoubleArrayDictionary

A new, mutable DoubleArrayDictionary instance.

Exceptions

ArgumentNullException

`keys` is null.

Deserialize(Stream)

Deserializes a dictionary from a stream.

```
public static DoubleArrayDictionary Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

DoubleArrayDictionary

A new instance of DoubleArrayDictionary.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The stream data is corrupted or in an unsupported format.

EnumerateAll(bool)

Enumerates all keys in the dictionary in lexicographical order.

```
public virtual IEnumerable<(int, byte[])> EnumerateAll(bool reverse = false)
```

Parameters

`reverse` bool

If true, returns keys in reverse lexicographical order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of all (identifier, key) tuples.

FindFirst(out int, out byte[])

Finds the first key in the dictionary in lexicographical order.

```
public virtual bool FindFirst(out int identifier, out byte[] key)
```

Parameters

`identifier` int

When this method returns, the identifier of the first key.

`key` byte[]

When this method returns, the first key.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindLast(out int, out byte[])

Finds the last key in the dictionary in lexicographical order.

```
public virtual bool FindLast(out int identifier, out byte[] key)
```

Parameters

`identifier` int

When this method returns, the identifier of the last key.

`key` byte[]

When this method returns, the last key.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindNext(int, out int, out byte[])

Finds the next key in lexicographical order after the specified identifier.

```
public virtual bool FindNext(int currentIdentifier, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` byte[]

When this method returns, the next key.

Returns

bool

true if a next key was found; otherwise, false.

FindNext(ReadOnlySpan<byte>, out int, out byte[])

Finds the next key in lexicographical order after the specified key.

```
public virtual bool FindNext(ReadOnlySpan<byte> currentKey, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentKey` ReadOnlySpan<byte>

The key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` byte[]

When this method returns, the next key.

Returns

bool

true if a next key was found; otherwise, false.

FindPrevious(int, out int, out byte[])

Finds the previous key in lexicographical order before the specified identifier.

```
public virtual bool FindPrevious(int currentIdentifier, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` byte[]

When this method returns, the previous key.

Returns

bool

true if a previous key was found; otherwise, false.

FindPrevious(ReadOnlySpan<byte>, out int, out byte[])

Finds the previous key in lexicographical order before the specified key.

```
public virtual bool FindPrevious(ReadOnlySpan<byte> currentKey, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentKey` ReadOnlySpan<byte>

The key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` byte[]

When this method returns, the previous key.

Returns

bool

true if a previous key was found; otherwise, false.

GetKey(int)

Gets the key associated with the specified identifier.

```
public virtual byte[] GetKey(int identifier)
```

Parameters

`identifier` int

The identifier of the key to get.

Returns

byte[]

The key as a byte array.

Exceptions

ArgumentOutOfRangeException

The specified identifier is out of the valid range.

ArgumentException

The specified identifier is invalid.

GetRecordAccess(int, bool)

Gets an accessor for the list of records associated with a given key identifier.

```
public KeyRecordDictionary.IRecordAccess GetRecordAccess(int identifier, bool isTransient
= false)
```

Parameters

`identifier` int

The identifier of the key whose records are to be accessed. This must be a valid key identifier obtained from a search or add operation.

`isTransient` bool

If true, accesses the transient (in-memory) record store; otherwise, accesses the persistent record store.

Returns

KeyRecordDictionary.IRecordAccess

An KeyRecordDictionary.IRecordAccess handle for the specified record list.

Remarks

Note that while new records can be added to a key's record list, new keys cannot be added to the dictionary itself after creation.

Exceptions

ArgumentException

The specified `identifier` is invalid or does not exist.

Remove(int)

Removes a key and its associated records from the dictionary using its identifier.

```
public virtual bool Remove(int identifier)
```

Parameters

`identifier` int

The identifier of the key to remove.

Returns

bool

true if the key was found and removed; otherwise, false.

Remove(ReadOnlySpan<byte>)

Removes a key and its associated records from the dictionary.

```
public virtual bool Remove(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

The key to remove.

Returns

bool

true if the key was found and removed; otherwise, false.

SearchByPrefix(ReadOnlySpan<byte>, bool)

Finds all keys that start with the given prefix.

```
public virtual IEnumerable<(int, byte[])> SearchByPrefix(ReadOnlySpan<byte> sequence, bool
reverse = false)
```

Parameters

`sequence` ReadOnlySpan<byte>

The prefix to search for.

`reverse` bool

If true, returns results in reverse lexicographical order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of (identifier, key) tuples for all keys starting with the prefix.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchByPrefix(ReadOnlySpan<byte>, bool).

SearchCommonPrefix(ReadOnlySpan<byte>)

Finds all keys in the dictionary that are prefixes of the given sequence.

```
public virtual IEnumerable<(int, byte[])> SearchCommonPrefix(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The sequence to search within.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of (identifier, key) tuples for all matching prefixes.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchCommonPrefix(ReadOnlySpan<byte>).

SearchExactly(ReadOnlySpan<byte>)

Searches for an exact match of the given sequence and returns its identifier.

```
public virtual int SearchExactly(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The key to search for.

Returns

int

The positive identifier for the key if found; otherwise, a non-positive value.

SearchLongestPrefix(ReadOnlySpan<byte>)

Finds the longest key in the dictionary that is a prefix of the given sequence.

```
public virtual (int, byte[]) SearchLongestPrefix(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The sequence to search within.

Returns

(int, byte[])

A tuple containing the identifier and key of the longest matching prefix, or (-1, []) if no match is found.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchLongestPrefix(ReadOnlySpan<byte>).

SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool)

Performs a wildcard search for keys matching a pattern.

```
public virtual IEnumerable<(int, byte[])> SearchWildcard(ReadOnlySpan<byte> sequence,
ReadOnlySpan<char> cards, bool reverse = false)
```

Parameters

`sequence` ReadOnlySpan<byte>

The byte sequence of the pattern.

`cards` ReadOnlySpan<char>

A sequence of characters ('?' for single, '*' for multiple wildcards) corresponding to the pattern.

`reverse` bool

If true, returns results in reverse order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of matching (identifier, key) tuples.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool).

Serialize(DoubleArrayDictionary, Stream, SerializationOptions?)

Serializes the state of the dictionary into a stream.

```
public static void Serialize(DoubleArrayDictionary dictionary, Stream stream,
KeyRecordDictionary.SerializationOptions? options = null)
```

Parameters

`dictionary` DoubleArrayDictionary

The dictionary instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` KeyRecordDictionary.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is a custom binary format specific to this Double-Array trie implementation.

Exceptions

ArgumentNullException

`dictionary` or `stream` is null.

TryAdd(ReadOnlySpan<byte>, out int)

Tries to add a key to the dictionary.

```
public virtual bool TryAdd(ReadOnlySpan<byte> key, out int identifier)
```

Parameters

`key` ReadOnlySpan<byte>

The key to add.

`identifier` int

When this method returns, contains the identifier for the new key. If the key already existed, this will be the existing identifier.

Returns

bool

true if the key was newly added; false if the key already existed.

Exceptions

ArgumentException

`key` is empty.

TryGetKey(int, out byte[])

Tries to get the key associated with the specified identifier.

```
public virtual bool TryGetKey(int identifier, out byte[] key)
```

Parameters

`identifier` int

The identifier of the key to get.

`key` byte[]

When this method returns, contains the key if found; otherwise, an empty array.

Returns

bool

true if the key was found; otherwise, false.

Class HashMapRecordStore

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of BasicRecordStore that uses a hash map as its underlying data structure.

```
public sealed class HashMapRecordStore : BasicRecordStore, ICloneable
```

Inheritance

object ← BasicRecordStore ← HashMapRecordStore

Implements

ICloneable

Inherited Members

BasicRecordStore.Serialize<T>(T, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, string, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, Stream, BasicRecordStore.SerializationOptions) , BasicRecordStore.Deserialize<T>(byte[]) , BasicRecordStore.Deserialize<T>(string) , BasicRecordStore.Deserialize<T>(Stream) , BasicRecordStore.Clear() , BasicRecordStore.Add(int) , BasicRecordStore.GetRecordAccess(int) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class uses a Dictionary<TKey, TValue> to store records. It provides fast, average-case O(1) time complexity for record lookups, additions, and removals. Unlike the tree-based stores, this implementation does not store or enumerate records in a sorted order.

Serialization Limits: The binary serialization format for this class imposes certain constraints on the data. Exceeding these limits will result in an InvalidDataException during serialization.

- **Max Records per List:** A single identifier can have a maximum of 16,777,215 records in its linked list.
- **Max Record Content Size:** The `Content` byte array of any single record cannot exceed 65,535 bytes.

Constructors

HashMapRecordStore()

Initializes a new, empty instance of the HashMapRecordStore class.

```
public HashMapRecordStore()
```

Methods

Clone()

Creates a deep copy of the HashMapRecordStore.

```
public object Clone()
```

Returns

object

A new HashMapRecordStore instance with the same keys and record data as the original.

Remarks

The method creates a new dictionary and copies all key-value pairs and their associated record lists, ensuring that the new store is independent of the original.

Contains(int)

Determines whether a record with the specified identifier exists in the hash map.

```
public override bool Contains(int identifier)
```

Parameters

`identifier` int

The identifier to locate.

Returns

bool

true if a record with the specified identifier is found; otherwise, false.

Deserialize(Stream)

Deserializes a HashMapRecordStore from a stream.

```
public static HashMapRecordStore Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

HashMapRecordStore

A new instance of HashMapRecordStore reconstructed from the stream.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The stream data is corrupted, in an unsupported format, or contains invalid values.

Enumerate()

Returns an enumerator that iterates through all records in the store.

```
public override IEnumerable<(int, BasicRecordStore.RecordAccess)> Enumerate()
```

Returns

IEnumerable<(int, BasicRecordStore.RecordAccess)>

An IEnumerable<T> of tuples, each containing an identifier and its corresponding BasicRecordStore.RecordAccess.

Remarks

The order of enumeration is not guaranteed to be sorted, as it depends on the internal implementation of the hash map.

GetRecordAccess(int, out bool)

Gets the accessor for a record's list of data. If an entry for the identifier does not exist, it is created.

```
public override BasicRecordStore.RecordAccess GetRecordAccess(int identifier, out
bool createNew)
```

Parameters

`identifier` int

The identifier of the record list.

`createNew` bool

When this method returns, contains true if a new entry was created; otherwise, false.

Returns

BasicRecordStore.RecordAccess

A BasicRecordStore.RecordAccess for the found or newly created record list.

Remove(int)

Removes the record (and its associated list of data) with the specified identifier from the hash map.

```
public override void Remove(int identifier)
```

Parameters

`identifier` int

The identifier of the record to remove.

Serialize(HashMapRecordStore, Stream, SerializationOptions?)

Serializes the entire state of the hash map store into a stream.

```
public static void Serialize(HashMapRecordStore store, Stream stream,
BasicRecordStore.SerializationOptions? options = null)
```

Parameters

`store` HashMapRecordStore

The HashMapRecordStore instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` BasicRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is specific to this hash map implementation, using Brotli compression and an XxHash32 checksum for data integrity.

Exceptions

ArgumentNullException

`store` or `stream` is null.

TryGetRecordAccess(int, out RecordAccess?)

Tries to get the accessor for a record's list of data.

```
public override bool TryGetRecordAccess(int identifier, out BasicRecordStore.RecordAccess?
access)
```

Parameters

`identifier` int

The identifier to locate.

`access` BasicRecordStore.RecordAccess

When this method returns, contains the BasicRecordStore.RecordAccess object if the identifier was found; otherwise, null.

Returns

bool

true if an entry with the specified identifier was found; otherwise, false.

Class ImmutableBitSet

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Represents a read-only, immutable bit set.

```
public class ImmutableBitSet
```

Inheritance

object ← ImmutableBitSet

Derived

RankSelectBitSet

Inherited Members

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

Constructors

ImmutableBitSet(ulong[], int)

Represents a read-only, immutable bit set.

```
public ImmutableBitSet(ulong[] buffer, int count)
```

Parameters

`buffer` ulong[]

The underlying ulong array that stores the bits.

`count` int

The total number of bits in the set.

Properties

Buffer

Gets the underlying ulong array that stores the bits.

```
public ulong[] Buffer { get; }
```

Property Value

ulong[]

Count

Gets the total number of bits in the set.

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

Property Value

int

this[int]

Gets the bit value at the specified index.

```
public bool this[int index] { get; }
```

Parameters

`index` int

The zero-based index of the bit to get.

Property Value

bool

The bit value (`true` for 1, `false` for 0).

Exceptions

ArgumentOutOfRangeException

`index` is out of range.

Methods

At(int)

Gets the bit value at the specified index.

```
public bool At(int index)
```

Parameters

`index` int

The zero-based index of the bit to get.

Returns

bool

The bit value (`true` for 1, `false` for 0).

ToRankSelect(ImmutableBitSet, bool)

Creates a new RankSelectBitSet from the specified ImmutableBitSet.

```
public static RankSelectBitSet ToRankSelect(ImmutableBitSet bitSet, bool createAuxDir
= true)
```

Parameters

`bitSet` ImmutableBitSet

The source immutable bit set to convert.

`createAuxDir` bool

A value indicating whether to immediately create the auxiliary directories required for fast Rank and Select operations.

Returns

RankSelectBitSet

A new RankSelectBitSet instance.

Remarks

If `createAuxDir` is set to `false`, this method is very fast, but the returned RankSelectBitSet will not be ready for Rank/Select operations until its auxiliary directories are created and set via the `SetAuxDir()` method. If set to `true` (the default), the auxiliary directories are created during this call, which takes more processing time but results in a fully initialized and ready-to-use object.

Exceptions

ArgumentNullException

`bitSet` is null.

ToRankSelect(bool)

Creates a new RankSelectBitSet from the specified ImmutableBitSet.

```
public RankSelectBitSet ToRankSelect(bool createAuxDir = true)
```

Parameters

`createAuxDir` bool

A value indicating whether to immediately create the auxiliary directories required for fast Rank and Select operations.

Returns

RankSelectBitSet

A new RankSelectBitSet instance.

Remarks

If `createAuxDir` is set to `false`, this method is very fast, but the returned RankSelectBitSet will not be ready for Rank/Select operations until its auxiliary directories are created and set via the `SetAuxDir()` method. If set to `true` (the default), the auxiliary directories are created during this call, which takes more processing time but results in a fully initialized and ready-to-use object.

Class KeyRecordDictionary

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

An abstract base class for advanced key-record dictionaries.

```
public abstract class KeyRecordDictionary
```

Inheritance

object ← KeyRecordDictionary

Derived

DirectedAcyclicGraphDictionary, DoubleArrayDictionary, LevelOrderBitsDictionary

Inherited Members

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

Remarks

This class provides a framework for mapping byte array keys to linked lists of data records. It features a dual-storage system (transient and persistent) for records, supports different key search directions (LTR/RTL), and allows for custom metadata storage. Concrete implementations are expected to provide the specific key management structure (e.g., a trie).

Constructors

KeyRecordDictionary(SearchDirectionType)

An abstract base class for advanced key-record dictionaries.

```
protected KeyRecordDictionary(KeyRecordDictionary.SearchDirectionType searchDirection)
```

Parameters

`searchDirection` KeyRecordDictionary.SearchDirectionType

The key search direction for the new dictionary.

Remarks

This class provides a framework for mapping byte array keys to linked lists of data records. It features a dual-storage system (transient and persistent) for records, supports different key search directions (LTR/RTL), and allows for custom metadata storage. Concrete implementations are expected to provide the specific key management structure (e.g., a trie).

Properties

Additional1

Gets or sets a small, general-purpose byte array for additional persistent data.

```
public byte[] Additional1 { get; set; }
```

Property Value

byte[]

Remarks

The size of the byte array cannot exceed 4096 bytes.

Exceptions

ArgumentNullException

The assigned value is null.

ArgumentOutOfRangeException

The length of the assigned byte array is greater than 4096.

Additional2

Gets or sets a large, general-purpose byte array for additional persistent data.

```
public byte[] Additional2 { get; set; }
```

Property Value

byte[]

Remarks

The size of the byte array cannot exceed 1,073,741,824 bytes (1 GB).

Exceptions

ArgumentNullException

The assigned value is null.

ArgumentOutOfRangeException

The length of the assigned byte array is greater than 1 GB.

SearchDirection

Gets the search direction (Left-To-Right or Right-To-Left) for key operations.

```
public KeyRecordDictionary.SearchDirectionType SearchDirection { get; }
```

Property Value

KeyRecordDictionary.SearchDirectionType

Methods

Create<T>(SearchDirectionType)

Creates a new instance of a derived dictionary type with an empty set of keys.

```
public static T Create<T>(KeyRecordDictionary.SearchDirectionType searchDirection =
SearchDirectionType.LTR) where T : KeyRecordDictionary
```

Parameters

`searchDirection` KeyRecordDictionary.SearchDirectionType

The key search direction for the new dictionary.

Returns

T

A new instance of the specified dictionary type.

Type Parameters

`T`

The concrete type of the dictionary to create.

Create<T>(IEnumerable<byte[]>, SearchDirectionType)

Creates a new instance of a derived dictionary type, populated with an initial set of keys.

```
public static T Create<T>(IEnumerable<byte[]> keys, KeyRecordDictionary.SearchDirectionType
searchDirection = SearchDirectionType.LTR) where T : KeyRecordDictionary
```

Parameters

`keys` IEnumerable<byte[]>

An enumerable collection of byte arrays to use as the initial keys. The collection does not need to be pre-sorted. Duplicate keys are permitted, but only the first occurrence will be added. The collection must not contain any elements that are null or empty byte array.

`searchDirection` KeyRecordDictionary.SearchDirectionType

The key search direction for the new dictionary.

Returns

T

A new instance of the specified dictionary type.

Type Parameters

`T`

The concrete type of the dictionary to create.

Remarks

This method uses reflection to invoke the static `Create` method on the concrete derived type `T`.

Exceptions

ArgumentNullException

`keys` is null.

Create<T>(IEnumerable<string>, Encoding?, SearchDirectionType)

Creates a new instance of a derived dictionary type, populated with an initial set of string keys.

```
public static T Create<T>(IEnumerable<string> keys, Encoding? encoding = null,
KeyRecordDictionary.SearchDirectionType searchDirection = SearchDirectionType.LTR) where T
: KeyRecordDictionary
```

Parameters

`keys` IEnumerable<string>

An enumerable collection of strings to use as the initial keys. The collection does not need to be pre-sorted. Duplicate keys are permitted, but only the first occurrence will be added. The collection must not contain any elements that are null or empty strings.

`encoding` Encoding

The encoding to use for converting strings to bytes. Defaults to UTF-8.

`searchDirection` KeyRecordDictionary.SearchDirectionType

The key search direction for the new dictionary.

Returns

T

A new instance of the specified dictionary type.

Type Parameters

`T`

The concrete type of the dictionary to create.

Exceptions

ArgumentNullException

`keys` is null.

Deserialize<T>(byte[])

Deserializes a dictionary from a byte array.

```
public static T Deserialize<T>(byte[] data) where T : KeyRecordDictionary
```

Parameters

`data` byte[]

The byte array containing the serialized data.

Returns

T

A new instance of `T`.

Type Parameters

`T`

The concrete type of the dictionary to deserialize.

Remarks

This static method uses reflection to invoke the static `Deserialize` method on the specified type `T`.

Exceptions

ArgumentNullException

`data` is null.

Deserialize<T>(Stream)

Deserializes a dictionary from a stream.

```
public static T Deserialize<T>(Stream stream) where T : KeyRecordDictionary
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

T

A new instance of `T`.

Type Parameters

`T`

The concrete type of the dictionary to deserialize.

Exceptions

ArgumentNullException

`stream` is null.

NotSupportedException

The type `T` does not have a public static `Deserialize(Stream)` method.

Deserialize<T>(string)

Deserializes a dictionary from a file.

```
public static T Deserialize<T>(string file) where T : KeyRecordDictionary
```

Parameters

`file` string

The path of the file to read from.

Returns

T

A new instance of `T`.

Type Parameters

`T`

The concrete type of the dictionary to deserialize.

Remarks

This static method uses reflection to invoke the static `Deserialize` method on the specified type `T`.

Exceptions

ArgumentNullException

`file` is null.

ArgumentException

`file` is empty or whitespace.

Serialize<T>(T, SerializationOptions?)

Serializes the specified dictionary instance into a byte array.

```
public static byte[] Serialize<T>(T obj, KeyRecordDictionary.SerializationOptions? options =
null) where T : KeyRecordDictionary
```

Parameters

`obj` T

The dictionary instance to serialize.

`options` KeyRecordDictionary.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Returns

byte[]

A byte array containing the serialized data.

Type Parameters

`T`

The concrete type of the dictionary.

Remarks

This static method uses reflection to invoke the static `Serialize` method on the concrete derived type `T`.

Exceptions

ArgumentNullException

`obj` is null.

Serialize<T>(T, Stream, SerializationOptions?)

Serializes the specified dictionary instance into a stream.

```
public static void Serialize<T>(T obj, Stream stream,
KeyRecordDictionary.SerializationOptions? options = null) where T : KeyRecordDictionary
```

Parameters

`obj` T

The dictionary instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` KeyRecordDictionary.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Type Parameters

`T`

The concrete type of the dictionary.

Exceptions

ArgumentNullException

`obj` or `stream` is null.

NotSupportedException

The type `T` does not have a public static `Serialize(T, Stream)` method.

Serialize<T>(T, string, SerializationOptions?)

Serializes the specified dictionary instance to a file.

```
public static void Serialize<T>(T obj, string file,
KeyRecordDictionary.SerializationOptions? options = null) where T : KeyRecordDictionary
```

Parameters

`obj` T

The dictionary instance to serialize.

`file` string

The path of the file to create.

`options` KeyRecordDictionary.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Type Parameters

`T`

The concrete type of the dictionary.

Remarks

This static method uses reflection to invoke the static `Serialize` method on the concrete derived type `T`.

Exceptions

ArgumentNullException

`obj` or `file` is null.

ArgumentException

`file` is empty or whitespace.

Interface KeyRecordDictionary.IKeyAccess

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Defines the primary public contract for all key-based operations within the dictionary.

```
public interface KeyRecordDictionary.IKeyAccess
```

Methods

Add(ReadOnlySpan<byte>)

Adds a key to the dictionary. If the key already exists, its existing identifier is returned.

```
int Add(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

The key to add.

Returns

int

The identifier for the added or existing key.

Exceptions

ArgumentException

`key` is empty.

AsStringSpecialized(Encoding?)

Returns a string-specialized wrapper for this key access interface.

```
KeyRecordDictionary.StringSpecialized AsStringSpecialized(Encoding? encoding = null)
```

Parameters

`encoding` Encoding

The encoding to use for string operations. Defaults to UTF-8.

Returns

KeyRecordDictionary.StringSpecialized

A new KeyRecordDictionary.StringSpecialized instance.

Contains(ReadOnlySpan<byte>)

Determines whether the dictionary contains the specified key.

```
bool Contains(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

The key to locate.

Returns

bool

true if the key is found; otherwise, false.

EnumerateAll(bool)

Enumerates all keys in the dictionary.

```
IEnumerable<(int, byte[])> EnumerateAll(bool reverse = false)
```

Parameters

`reverse` bool

If true, returns keys in reverse lexicographical order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of all (identifier, key) tuples.

FindFirst(out int, out byte[])

Finds the first key in the dictionary according to the current sort order.

```
bool FindFirst(out int identifier, out byte[] key)
```

Parameters

`identifier` int

When this method returns, the identifier of the first key.

`key` byte[]

When this method returns, the first key.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindLast(out int, out byte[])

Finds the last key in the dictionary according to the current sort order.

```
bool FindLast(out int identifier, out byte[] key)
```

Parameters

`identifier` int

When this method returns, the identifier of the last key.

`key` byte[]

When this method returns, the last key.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindNext(int, out int, out byte[])

Finds the next key in sequence after the specified identifier.

```
bool FindNext(int currentIdentifier, out int foundIdentifier, out byte[] foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` byte[]

When this method returns, the next key.

Returns

bool

true if a next key was found; otherwise, false.

FindNext(ReadOnlySpan<byte>, out int, out byte[])

Finds the next key in sequence after the specified key.

```
bool FindNext(ReadOnlySpan<byte> currentKey, out int foundIdentifier, out byte[] foundKey)
```

Parameters

`currentKey` ReadOnlySpan<byte>

The key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` byte[]

When this method returns, the next key.

Returns

bool

true if a next key was found; otherwise, false.

FindPrevious(int, out int, out byte[])

Finds the previous key in sequence before the specified identifier.

```
bool FindPrevious(int currentIdentifier, out int foundIdentifier, out byte[] foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` byte[]

When this method returns, the previous key.

Returns

bool

true if a previous key was found; otherwise, false.

FindPrevious(ReadOnlySpan<byte>, out int, out byte[])

Finds the previous key in sequence before the specified key.

```
bool FindPrevious(ReadOnlySpan<byte> currentKey, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentKey` ReadOnlySpan<byte>

The key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` byte[]

When this method returns, the previous key.

Returns

bool

true if a previous key was found; otherwise, false.

GetKey(int)

Gets the key associated with the specified identifier.

```
byte[] GetKey(int identifier)
```

Parameters

`identifier` int

The identifier of the key to get.

Returns

byte[]

The key as a byte array.

Exceptions

KeyNotFoundException

The specified identifier does not exist.

GetRecordAccess(int, bool)

Gets an accessor for the list of records associated with a given key identifier.

```
KeyRecordDictionary.IRecordAccess GetRecordAccess(int identifier, bool isTransient = false)
```

Parameters

`identifier` int

The identifier of the key whose records are to be accessed.

`isTransient` bool

If true, accesses the transient record store; otherwise, accesses the persistent record store.

Returns

KeyRecordDictionary.IRecordAccess

An KeyRecordDictionary.IRecordAccess handle for the specified record list.

Remove(int)

Removes a key from the dictionary using its identifier.

```
bool Remove(int identifier)
```

Parameters

`identifier` int

The identifier of the key to remove.

Returns

bool

true if the key was found and removed; otherwise, false.

Remove(ReadOnlySpan<byte>)

Removes a key from the dictionary.

```
bool Remove(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

The key to remove.

Returns

bool

true if the key was found and removed; otherwise, false.

SearchByPrefix(ReadOnlySpan<byte>, bool)

Finds all keys that start with the given prefix.

```
IEnumerable<(int, byte[])> SearchByPrefix(ReadOnlySpan<byte> sequence, bool reverse = false)
```

Parameters

`sequence` ReadOnlySpan<byte>

The prefix to search for.

`reverse` bool

If true, returns results in reverse lexicographical order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of (identifier, key) tuples for all keys starting with the prefix.

Examples

If the dictionary contains the keys "a", "app", and "apple", and the input sequence is "ap", this method will return two keys: "app" and "apple".

SearchCommonPrefix(ReadOnlySpan<byte>)

Finds all keys in the dictionary that are prefixes of the given sequence.

```
IEnumerable<(int, byte[])> SearchCommonPrefix(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The sequence to search within.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of (identifier, key) tuples for all matching prefixes.

Examples

If the dictionary contains the keys "a", "app", and "apple", and the input sequence is "applepie", this method will return all three keys: "a", "app", and "apple".

SearchExactly(ReadOnlySpan<byte>)

Searches for an exact match of the given sequence and returns its identifier.

```
int SearchExactly(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The key to search for.

Returns

int

The identifier for the key if found; otherwise, a value indicating not found (typically 0 or -1).

Examples

If the dictionary contains the keys "a", "app", and "apple", and the input sequence is "apple", this method will return a key: "apple".

SearchLongestPrefix(ReadOnlySpan<byte>)

Finds the longest key in the dictionary that is a prefix of the given sequence.

```
(int, byte[]) SearchLongestPrefix(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The sequence to search within.

Returns

(int, byte[])

A tuple containing the identifier and key of the longest matching prefix, or a default value if no match is found.

Examples

If the dictionary contains the keys "a", "app", and "apple", and the input sequence is "applepie", this method will return a key: "apple".

SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool)

Performs a wildcard search for keys matching a pattern.

```
IEnumerable<(int, byte[])> SearchWildcard(ReadOnlySpan<byte> sequence, ReadOnlySpan<char>
cards, bool reverse = false)
```

Parameters

`sequence` ReadOnlySpan<byte>

The byte sequence of the pattern.

`cards` ReadOnlySpan<char>

A sequence of characters ('?' for single, '*' for multiple wildcards) corresponding to the pattern.

`reverse` bool

If true, returns results in reverse order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of matching (identifier, key) tuples.

Examples

If the dictionary contains the keys "a", "app", and "apple", and the input sequence is "a?p*", this method will return two keys: "app" and "apple". The search pattern is provided as two separate arguments (`sequence` and `cards`) to unambiguously support searching for any possible byte value (0-255). The `sequence` span contains the literal byte values for the pattern, while the `cards` span defines the role of each corresponding position. This design allows a search to include literal byte values that might otherwise be interpreted as wildcard characters (e.g., the byte value 63, which is the ASCII code for '?').

TryAdd(ReadOnlySpan<byte>, out int)

Tries to add a key to the dictionary.

```
bool TryAdd(ReadOnlySpan<byte> key, out int identifier)
```

Parameters

`key` ReadOnlySpan<byte>

The key to add.

`identifier` int

When this method returns, contains the identifier for the new key. If the key already existed, this will be the existing identifier.

Returns

bool

true if the key was newly added; false if the key already existed.

Exceptions

ArgumentException

`key` is empty.

TryGetKey(int, out byte[])

Tries to get the key associated with the specified identifier.

```
bool TryGetKey(int identifier, out byte[] key)
```

Parameters

`identifier` int

The identifier of the key to get.

`key` byte[]

When this method returns, contains the key associated with the identifier, if found; otherwise, an empty array.

Returns

bool

true if the key was found; otherwise, false.

Interface KeyRecordDictionary.IRecord

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Defines a public contract for a single data record.

```
public interface KeyRecordDictionary.IRecord
```

Properties

Content

Gets or sets the byte array content of this record.

```
byte[] Content { get; set; }
```

Property Value

byte[]

ExByte

Gets or sets an extra, user-definable byte of metadata associated with this record (0-255). Not supported by all underlying record stores, only supported by PrimitiveRecordStore.

```
int ExByte { get; set; }
```

Property Value

int

Next

Gets the next record in the linked list, or null if this is the last record.

```
KeyRecordDictionary.IRecord? Next { get; }
```

Property Value

KeyRecordDictionary.IRecord

Methods

InsertAfter(ReadOnlySpan<byte>)

Inserts a new record immediately after this record in the list.

```
KeyRecordDictionary.IRecord InsertAfter(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record to insert.

Returns

KeyRecordDictionary.IRecord

An KeyRecordDictionary.IRecord handle for the newly inserted record.

InsertBefore(ReadOnlySpan<byte>)

Inserts a new record immediately before this record in the list.

```
KeyRecordDictionary.IRecord InsertBefore(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record to insert.

Returns

KeyRecordDictionary.IRecord

An KeyRecordDictionary.IRecord handle for the newly inserted record.

Remove()

Removes this record from the list it belongs to.

```
void Remove()
```

Interface KeyRecordDictionary.IRecordAccess

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Defines a public contract for accessing and managing a list of records associated with a single key.

```
public interface KeyRecordDictionary.IRecordAccess :
IEnumerable<KeyRecordDictionary.IRecord>, IEnumerable
```

Inherited Members

IEnumerable<KeyRecordDictionary.IRecord>.GetEnumerator()

Methods

Add(ReadOnlySpan<byte>)

Adds a new record with the specified content to the end of the list.

```
KeyRecordDictionary.IRecord Add(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record.

Returns

KeyRecordDictionary.IRecord

An KeyRecordDictionary.IRecord handle for the newly added record.

Clear()

Removes all records from this list.

```
void Clear()
```

Enum KeyRecordDictionary.SearchDirectionType

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Specifies the direction for key matching and sorting operations.

```
public enum KeyRecordDictionary.SearchDirectionType
```

Fields

`LTR = 0`

Left-To-Right search, representing standard lexicographical order.

`RTL = 1`

Right-To-Left search, useful for suffix matching and reverse lexicographical order.

Class KeyRecordDictionary.SerializationOptions

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Provides options to control the serialization process for dictionary data structures.

```
public class KeyRecordDictionary.SerializationOptions
```

Inheritance

object ← KeyRecordDictionary.SerializationOptions

Inherited Members

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

Properties

CompressionLevel

Gets or sets the compression level to use for serialization.

```
public CompressionLevel CompressionLevel { get; set; }
```

Property Value

CompressionLevel

Remarks

Higher compression levels can result in smaller file sizes but may take longer to process. Defaults to Fastest for a balance of speed and size.

Default

Gets a singleton instance of KeyRecordDictionary.SerializationOptions with default values.

```
public static KeyRecordDictionary.SerializationOptions Default { get; }
```

Property Value

KeyRecordDictionary.SerializationOptions

Remarks

Use this property to avoid creating a new options object when default settings are sufficient.

IncludeLoudsParentPointers

[LOUDS-specific] Gets or sets a value indicating whether to include the parent pointer array in the serialization output. Defaults to `false`.

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

Property Value

bool

Remarks

Setting this to `false` will reduce the final file size, but it may significantly increase the time required for deserialization as the parent structure needs to be rebuilt.

Class KeyRecordDictionary.StringSpecialized

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Provides a string-specialized wrapper around an KeyRecordDictionary.IKeyAccess instance, simplifying operations by handling string-to-byte encoding and decoding.

```
public sealed class KeyRecordDictionary.StringSpecialized
```

Inheritance

object ← KeyRecordDictionary.StringSpecialized

Inherited Members

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

Properties

Dictionary

Gets the underlying KeyRecordDictionary.IKeyAccess dictionary.

```
public KeyRecordDictionary.IKeyAccess Dictionary { get; }
```

Property Value

KeyRecordDictionary.IKeyAccess

Encoding

Gets the Encoding used for string conversions.

```
public Encoding Encoding { get; }
```

Property Value

Encoding

Methods

Add(string)

Adds a key to the dictionary. If the key already exists, its existing identifier is returned.

```
public int Add(string key)
```

Parameters

`key` string

The string key to add.

Returns

int

The identifier for the added or existing key.

Exceptions

ArgumentNullException

`key` is null.

ArgumentException

`key` is empty.

Contains(string)

Determines whether the dictionary contains the specified key.

```
public bool Contains(string key)
```

Parameters

`key` string

The string key to locate.

Returns

bool

true if the key is found; otherwise, false.

Exceptions

ArgumentNullException

`key` is null.

EnumerateAll(bool)

Enumerates all keys in the dictionary as strings.

```
public IEnumerable<(int, string)> EnumerateAll(bool reverse = false)
```

Parameters

`reverse` bool

If true, returns keys in reverse order.

Returns

IEnumerable<(int, string)>

An enumerable collection of all (identifier, key) string tuples.

FindFirst(out int, out string)

Finds the first key in the dictionary according to the current sort order.

```
public bool FindFirst(out int identifier, out string key)
```

Parameters

`identifier` int

When this method returns, the identifier of the first key.

`key` string

When this method returns, the first key as a string.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindLast(out int, out string)

Finds the last key in the dictionary according to the current sort order.

```
public bool FindLast(out int identifier, out string key)
```

Parameters

`identifier` int

When this method returns, the identifier of the last key.

`key` string

When this method returns, the last key as a string.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindNext(int, out int, out string)

Finds the next key in sequence after the specified identifier.

```
public bool FindNext(int currentIdentifier, out int foundIdentifier, out string foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` string

When this method returns, the next key as a string.

Returns

bool

true if a next key was found; otherwise, false.

FindNext(string, out int, out string)

Finds the next key in sequence after the specified key.

```
public bool FindNext(string currentKey, out int foundIdentifier, out string foundKey)
```

Parameters

`currentKey` string

The string key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` string

When this method returns, the next key as a string.

Returns

bool

true if a next key was found; otherwise, false.

Exceptions

ArgumentNullException

`currentKey` is null.

FindPrevious(int, out int, out string)

Finds the previous key in sequence before the specified identifier.

```
public bool FindPrevious(int currentIdentifier, out int foundIdentifier, out
string foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` string

When this method returns, the previous key as a string.

Returns

bool

true if a previous key was found; otherwise, false.

FindPrevious(string, out int, out string)

Finds the previous key in sequence before the specified key.

```
public bool FindPrevious(string currentKey, out int foundIdentifier, out string foundKey)
```

Parameters

`currentKey` string

The string key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` string

When this method returns, the previous key as a string.

Returns

bool

true if a previous key was found; otherwise, false.

Exceptions

ArgumentNullException

`currentKey` is null.

GetKey(int)

Gets the key associated with the specified identifier.

```
public string GetKey(int identifier)
```

Parameters

`identifier` int

The identifier of the key to get.

Returns

string

The key as a string.

GetRecordAccess(int, bool)

Gets an accessor for the list of records associated with a given key identifier.

```
public KeyRecordDictionary.IRecordAccess GetRecordAccess(int identifier, bool isTransient
= false)
```

Parameters

`identifier` int

The identifier of the key whose records are to be accessed.

`isTransient` bool

If true, accesses the transient record store; otherwise, accesses the persistent record store.

Returns

KeyRecordDictionary.IRecordAccess

An KeyRecordDictionary.IRecordAccess handle for the specified record list.

Remove(string)

Removes a key from the dictionary.

```
public bool Remove(string key)
```

Parameters

`key` string

The string key to remove.

Returns

bool

true if the key was found and removed; otherwise, false.

Exceptions

ArgumentNullException

`key` is null.

SearchByPrefix(string, bool)

Finds all keys that start with the given prefix.

```
public IEnumerable<(int, string)> SearchByPrefix(string text, bool reverse = false)
```

Parameters

`text` string

The prefix to search for.

`reverse` bool

If true, returns results in reverse order.

Returns

IEnumerable<(int, string)>

An enumerable collection of (identifier, key) string tuples.

Remarks

This is the string-specialized version of this method. For detailed behavior and examples, see SearchByPrefix(ReadOnlySpan<byte>, bool).

Exceptions

ArgumentNullException

`text` is null.

SearchCommonPrefix(string)

Finds all keys in the dictionary that are prefixes of the given text.

```
public IEnumerable<(int, string)> SearchCommonPrefix(string text)
```

Parameters

`text` string

The text to search within.

Returns

IEnumerable<(int, string)>

An enumerable collection of (identifier, key) string tuples for all matching prefixes.

Remarks

This is the string-specialized version of this method. For detailed behavior and examples, see SearchCommonPrefix(ReadOnlySpan<byte>).

Exceptions

ArgumentNullException

`text` is null.

SearchExactly(string)

Searches for an exact match of the given text and returns its identifier.

```
public int SearchExactly(string text)
```

Parameters

`text` string

The string key to search for.

Returns

int

The identifier for the key if found; otherwise, a value indicating not found.

Exceptions

ArgumentNullException

`text` is null.

SearchLongestPrefix(string)

Finds the longest key in the dictionary that is a prefix of the given text.

```
public (int, string) SearchLongestPrefix(string text)
```

Parameters

`text` string

The text to search within.

Returns

(int, string)

A tuple containing the identifier and string key of the longest matching prefix, or (-1, "") if no match is found.

Remarks

This is the string-specialized version of this method. For detailed behavior and examples, see SearchLongestPrefix(ReadOnlySpan<byte>).

Exceptions

ArgumentNullException

`text` is null.

SearchWildcard(string, char, char, bool)

Performs a wildcard search using user-defined wildcard characters.

```
public IEnumerable<(int, string)> SearchWildcard(string pattern, char cardQ = '?', char
cardA = '*', bool reverse = false)
```

Parameters

`pattern` string

The search pattern.

`cardQ` char

The character to be treated as a single-character wildcard ('?'), which matches any single character.

`cardA` char

The character to be treated as a multi-character wildcard ('*'), which matches any sequence of zero or more characters.

`reverse` bool

If true, returns results in reverse order.

Returns

IEnumerable<(int, string)>

An enumerable collection of matching (identifier, key) string tuples.

Remarks

This is a convenience method that provides a simple way to perform wildcard searches. It internally translates the pattern for use by the more advanced SearchWildcard(string, string, bool) overload. A call like

```
SearchWildcard("Hell?*World")
```

is internally converted to an equivalent call:

```
SearchWildcard("Hell?*World", "....?*.....")
```

.
The second argument, "cards", specifies that '?' and '*' should be treated as wildcards, while '.' indicates a literal character match.

Encoding Limitation: The multi-character wildcard ('*') is only fully supported for single-byte encodings (codepage 1252, 20127 and 28591). The single-character wildcard ('?') is supported for some encodings (codepage 1200, 1201, 1252, 12000, 20127 and 28591).

Exceptions

ArgumentNullException

`pattern` is null.

NotSupportedException

The current encoding is not supported for wildcard searches.

SearchWildcard(string, string, bool)

Performs a wildcard search using a standardized wildcard pattern.

```
public IEnumerable<(int, string)> SearchWildcard(string text, string cards, bool reverse
= false)
```

Parameters

`text` string

The search pattern as a string.

`cards` string

A string of the same length as `text`'s byte representation, where '?' is a single wildcard, '*' is a multiple wildcard, and '.' is a literal match.

`reverse` bool

If true, returns results in reverse order.

Returns

IEnumerable<(int, string)>

An enumerable collection of matching (identifier, key) string tuples.

Remarks

This is the string-specialized version of this method. For detailed behavior and examples, see SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool).

Encoding Limitation: The multi-character wildcard ('*') is only fully supported for single-byte encodings (codepage 1252, 20127 and 28591). The single-character wildcard ('?') is supported for some encodings (codepage 1200, 1201, 1252, 12000, 20127 and 28591).

Exceptions

ArgumentNullException

`text` is null.

ArgumentNullException

`cards` is null.

ArgumentException

`cards` and `cards` are different length.

NotSupportedException

The current encoding is not supported for wildcard searches.

TryAdd(string, out int)

Tries to add a key to the dictionary.

```
public bool TryAdd(string key, out int identifier)
```

Parameters

`key` string

The string key to add.

`identifier` int

When this method returns, contains the identifier for the new key. If the key already existed, this will be the existing identifier.

Returns

bool

true if the key was newly added; false if the key already existed.

Exceptions

ArgumentNullException

`key` is null.

ArgumentException

`key` is empty.

TryGetKey(int, out string)

Tries to get the key associated with the specified identifier.

```
public bool TryGetKey(int identifier, out string key)
```

Parameters

`identifier` int

The identifier of the key to get.

key string

When this method returns, contains the key as a string, if found; otherwise, an empty string.

Returns

bool

true if the key was found; otherwise, false.

Class LevelOrderBitsDictionary

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

An abstract base class for read-only, trie-based dictionaries that are constructed in level-order.

```
public abstract class LevelOrderBitsDictionary : KeyRecordDictionary,
KeyRecordDictionary.IKeyAccess
```

Inheritance

object ← KeyRecordDictionary ← LevelOrderBitsDictionary

Implements

KeyRecordDictionary.IKeyAccess

Derived

BitwiseVectorDictionary, LoudsDictionary

Inherited Members

KeyRecordDictionary.SearchDirection , KeyRecordDictionary.Additional1 , KeyRecordDictionary.Additional2 , KeyRecordDictionary.Create<T>(KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Create<T>(IEnumerable<byte[]>, KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Create<T>(IEnumerable<string>, Encoding, KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Serialize<T>(T, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, string, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, Stream, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Deserialize<T>(byte[]) , KeyRecordDictionary.Deserialize<T>(string) , KeyRecordDictionary.Deserialize<T>(Stream) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class provides the common infrastructure for trie implementations that are "flattened" into arrays based on a breadth-first (level-order) traversal. The core data is stored in arrays for labels, parent pointers, and terminal node flags.

Concrete implementations must provide a strategy for navigating the tree structure, primarily by implementing the FindRange1(int) method.

Important Usage Notes: This class and its derivatives are read-only after creation. Key addition and removal operations are not supported.

Methods

Add(ReadOnlySpan<byte>)

This operation is not supported. The dictionary is read-only after creation.

```
public int Add(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

Returns

int

Exceptions

NotSupportedException

Always thrown.

AsStringSpecialized(Encoding?)

Returns a string-specialized wrapper for the dictionary's key access interface.

```
public KeyRecordDictionary.StringSpecialized AsStringSpecialized(Encoding? encoding = null)
```

Parameters

`encoding` Encoding

The encoding to use for string conversions. Defaults to UTF-8.

Returns

KeyRecordDictionary.StringSpecialized

A new KeyRecordDictionary.StringSpecialized instance providing string-based access to the dictionary.

Contains(ReadOnlySpan<byte>)

Determines whether the specified key exists in the graph.

```
public virtual bool Contains(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

The key to locate.

Returns

bool

true if the key is found; otherwise, false.

EnumerateAll(bool)

Enumerates all keys in the dictionary in lexicographical order.

```
public virtual IEnumerable<(int, byte[])> EnumerateAll(bool reverse = false)
```

Parameters

`reverse` bool

If true, returns keys in reverse lexicographical order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of all (identifier, key) tuples.

FindFirst(out int, out byte[])

Finds the first key in the dictionary in lexicographical order.

```
public virtual bool FindFirst(out int identifier, out byte[] key)
```

Parameters

`identifier` int

When this method returns, the identifier of the first key.

`key` byte[]

When this method returns, the first key.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindLast(out int, out byte[])

Finds the last key in the dictionary in lexicographical order.

```
public virtual bool FindLast(out int identifier, out byte[] key)
```

Parameters

`identifier` int

When this method returns, the identifier of the last key.

`key` byte[]

When this method returns, the last key.

Returns

bool

true if the dictionary is not empty; otherwise, false.

FindNext(int, out int, out byte[])

Finds the next key in lexicographical order after the specified identifier.

```
public virtual bool FindNext(int currentIdentifier, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` byte[]

When this method returns, the next key.

Returns

bool

true if a next key was found; otherwise, false.

FindNext(ReadOnlySpan<byte>, out int, out byte[])

Finds the next key in lexicographical order after the specified key.

```
public virtual bool FindNext(ReadOnlySpan<byte> currentKey, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentKey` ReadOnlySpan<byte>

The key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the next key.

`foundKey` byte[]

When this method returns, the next key.

Returns

bool

true if a next key was found; otherwise, false.

FindPrevious(int, out int, out byte[])

Finds the previous key in lexicographical order before the specified identifier.

```
public virtual bool FindPrevious(int currentIdentifier, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentIdentifier` int

The identifier to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` byte[]

When this method returns, the previous key.

Returns

bool

true if a previous key was found; otherwise, false.

FindPrevious(ReadOnlySpan<byte>, out int, out byte[])

Finds the previous key in lexicographical order before the specified key.

```
public virtual bool FindPrevious(ReadOnlySpan<byte> currentKey, out int foundIdentifier, out
byte[] foundKey)
```

Parameters

`currentKey` ReadOnlySpan<byte>

The key to start the search from.

`foundIdentifier` int

When this method returns, the identifier of the previous key.

`foundKey` byte[]

When this method returns, the previous key.

Returns

bool

true if a previous key was found; otherwise, false.

GetKey(int)

Gets the key associated with the specified identifier.

```
public virtual byte[] GetKey(int identifier)
```

Parameters

`identifier` int

The identifier of the key to get.

Returns

byte[]

The key as a byte array.

Exceptions

ArgumentOutOfRangeException

The specified identifier does not exist.

GetRecordAccess(int, bool)

Gets an accessor for the list of records associated with a given key identifier.

```
public KeyRecordDictionary.IRecordAccess GetRecordAccess(int identifier, bool isTransient
= false)
```

Parameters

`identifier` int

The identifier of the key whose records are to be accessed. This must be a valid key identifier obtained from a search or add operation.

`isTransient` bool

If true, accesses the transient (in-memory) record store; otherwise, accesses the persistent record store.

Returns

KeyRecordDictionary.IRecordAccess

An KeyRecordDictionary.IRecordAccess handle for the specified record list.

Remarks

Note that while new records can be added to a key's record list, new keys cannot be added to the dictionary itself after creation.

Exceptions

ArgumentOutOfRangeException

`identifier` is negative.

ArgumentException

The specified `identifier` is invalid or does not exist.

Remove(int)

This operation is not supported. The dictionary is read-only after creation.

```
public bool Remove(int identifier)
```

Parameters

`identifier` int

Returns

bool

Exceptions

NotSupportedException

Always thrown.

Remove(ReadOnlySpan<byte>)

This operation is not supported. The dictionary is read-only after creation.

```
public bool Remove(ReadOnlySpan<byte> key)
```

Parameters

`key` ReadOnlySpan<byte>

Returns

bool

Exceptions

NotSupportedException

Always thrown.

SearchByPrefix(ReadOnlySpan<byte>, bool)

Finds all keys that start with the given prefix.

```
public virtual IEnumerable<(int, byte[])> SearchByPrefix(ReadOnlySpan<byte> sequence, bool
reverse = false)
```

Parameters

`sequence` ReadOnlySpan<byte>

The prefix to search for.

`reverse` bool

If true, returns results in reverse lexicographical order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of (identifier, key) tuples for all keys starting with the prefix.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchByPrefix(ReadOnlySpan<byte>, bool).

SearchCommonPrefix(ReadOnlySpan<byte>)

Finds all keys in the dictionary that are prefixes of the given sequence.

```
public virtual IEnumerable<(int, byte[])> SearchCommonPrefix(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The sequence to search within.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of (identifier, key) tuples for all matching prefixes.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchCommonPrefix(ReadOnlySpan<byte>).

SearchExactly(ReadOnlySpan<byte>)

Searches for an exact match of the given sequence and returns its identifier.

```
public virtual int SearchExactly(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The key to search for.

Returns

int

The positive identifier for the key if found; otherwise, a non-positive value.

SearchLongestPrefix(ReadOnlySpan<byte>)

Finds the longest key in the dictionary that is a prefix of the given sequence.

```
public virtual (int, byte[]) SearchLongestPrefix(ReadOnlySpan<byte> sequence)
```

Parameters

`sequence` ReadOnlySpan<byte>

The sequence to search within.

Returns

(int, byte[])

A tuple containing the identifier and key of the longest matching prefix, or (-1, []) if no match is found.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchLongestPrefix(ReadOnlySpan<byte>).

SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool)

Performs a wildcard search for keys matching a pattern.

```
public virtual IEnumerable<(int, byte[])> SearchWildcard(ReadOnlySpan<byte> sequence,
ReadOnlySpan<char> cards, bool reverse = false)
```

Parameters

`sequence` ReadOnlySpan<byte>

The byte sequence of the pattern.

`cards` ReadOnlySpan<char>

A sequence of characters ('?' for single, '*' for multiple wildcards) corresponding to the pattern.

`reverse` bool

If true, returns results in reverse order.

Returns

IEnumerable<(int, byte[])>

An enumerable collection of matching (identifier, key) tuples.

Remarks

This is the implementation of interface method. For detailed behavior and examples, see SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool).

TryAdd(ReadOnlySpan<byte>, out int)

This operation is not supported. The dictionary is read-only after creation.

```
public bool TryAdd(ReadOnlySpan<byte> key, out int identifier)
```

Parameters

`key` ReadOnlySpan<byte>

`identifier` int

Returns

bool

Exceptions

NotSupportedException

Always thrown.

TryGetKey(int, out byte[])

Tries to get the key associated with the specified identifier.

```
public virtual bool TryGetKey(int identifier, out byte[] key)
```

Parameters

`identifier` int

The identifier of the key to get.

`key` byte[]

When this method returns, contains the key if found; otherwise, an empty array.

Returns

bool

true if the key was found; otherwise, false.

Class LoudsDictionary

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of KeyRecordDictionary that uses a memory-efficient LOUDS (Level-Order Unary Degree Sequence) trie.

```
public abstract class LoudsDictionary : LevelOrderBitsDictionary,
KeyRecordDictionary.IKeyAccess
```

Inheritance

object ← KeyRecordDictionary ← LevelOrderBitsDictionary ← LoudsDictionary

Implements

KeyRecordDictionary.IKeyAccess

Inherited Members

LevelOrderBitsDictionary.AsStringSpecialized(Encoding) ,
LevelOrderBitsDictionary.GetRecordAccess(int, bool) ,
LevelOrderBitsDictionary.Contains(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.SearchExactly(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.SearchCommonPrefix(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.SearchLongestPrefix(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.SearchByPrefix(ReadOnlySpan<byte>, bool) ,
LevelOrderBitsDictionary.EnumerateAll(bool) ,
LevelOrderBitsDictionary.SearchWildcard(ReadOnlySpan<byte>, ReadOnlySpan<char>, bool) ,
LevelOrderBitsDictionary.FindFirst(out int, out byte[]) ,
LevelOrderBitsDictionary.FindLast(out int, out byte[]) ,
LevelOrderBitsDictionary.FindNext(int, out int, out byte[]) ,
LevelOrderBitsDictionary.FindNext(ReadOnlySpan<byte>, out int, out byte[]) ,
LevelOrderBitsDictionary.FindPrevious(int, out int, out byte[]) ,
LevelOrderBitsDictionary.FindPrevious(ReadOnlySpan<byte>, out int, out byte[]) ,
LevelOrderBitsDictionary.TryGetKey(int, out byte[]) , LevelOrderBitsDictionary.GetKey(int) ,
LevelOrderBitsDictionary.Add(ReadOnlySpan<byte>) ,
LevelOrderBitsDictionary.TryAdd(ReadOnlySpan<byte>, out int) , LevelOrderBitsDictionary.Remove(int) ,
LevelOrderBitsDictionary.Remove(ReadOnlySpan<byte>) , KeyRecordDictionary.SearchDirection ,
KeyRecordDictionary.Additional1 , KeyRecordDictionary.Additional2 ,
KeyRecordDictionary.Create<T>(KeyRecordDictionary.SearchDirectionType) ,
KeyRecordDictionary.Create<T>(IEnumerable<byte[]>, KeyRecordDictionary.SearchDirectionType) ,

KeyRecordDictionary.Create<T>(IEnumerable<string>, Encoding, KeyRecordDictionary.SearchDirectionType) , KeyRecordDictionary.Serialize<T>(T, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, string, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Serialize<T>(T, Stream, KeyRecordDictionary.SerializationOptions) , KeyRecordDictionary.Deserialize<T>(byte[]) , KeyRecordDictionary.Deserialize<T>(string) , KeyRecordDictionary.Deserialize<T>(Stream) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class represents a trie data structure using a compact, bit-level representation to achieve significant memory savings. The structure is built from a sorted set of keys and is highly optimized for prefix-based searches.

Important Usage Notes:

- **Read-Only Structure:** Like the DAWG implementation, this dictionary is built once and is effectively read-only. Key addition and removal operations are not supported and will throw a NotSupported Exception.
- **Key Reconstruction:** Getting a key by its identifier requires traversing the structure from a node up to the root.

Methods

Create(IEnumerable<byte[]>, SearchDirectionType)

Creates a new LoudsDictionary from a collection of keys.

```
public static LoudsDictionary Create(IEnumerable<byte[]> keys,
KeyRecordDictionary.SearchDirectionType searchDirection = SearchDirectionType.LTR)
```

Parameters

`keys` IEnumerable<byte[]>

An enumerable collection of byte arrays to use as the initial keys. The collection does not need to be pre-sorted. Duplicate keys are permitted, but only the first occurrence will be added. The collection must not contain any elements that are null or empty byte array.

`searchDirection` KeyRecordDictionary.SearchDirectionType

The key search direction for the new dictionary.

Returns

LoudsDictionary

A new, optimized, and read-only LoudsDictionary instance.

Remarks

The input keys are sorted and used to construct the LOUDS trie representation.

Exceptions

ArgumentNullException

`keys` is null.

Deserialize(Stream)

Deserializes a dictionary from a stream.

```
public static LoudsDictionary Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

LoudsDictionary

A new instance of LoudsDictionary.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The stream data is corrupted or in an unsupported format.

Serialize(LoudsDictionary, Stream, SerializationOptions?)

Serializes the state of the dictionary into a stream.

```
public static void Serialize(LoudsDictionary dictionary, Stream stream,
KeyRecordDictionary.SerializationOptions? options = null)
```

Parameters

`dictionary` LoudsDictionary

The dictionary instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` KeyRecordDictionary.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is a custom binary format specific to this LOUDS implementation.

Exceptions

ArgumentNullException

`dictionary` or `stream` is null.

Class PrimitiveRecordStore

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Provides a low-level, memory-efficient data store for variable-length byte records.

```
public sealed class PrimitiveRecordStore
```

Inheritance

object ← PrimitiveRecordStore

Inherited Members

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

Remarks

This store uses a series of fixed-size byte arrays ("chunks") to avoid allocations on the Large Object Heap. It internally manages a system of linked lists, where each list is a sequence of records. The entire store can be serialized to and deserialized from a compressed, checksummed binary format.

Record Size Limit: Please note that the `Content` for any single record cannot exceed 4,096 bytes. This limit is enforced during operations such as adding or updating records.

Important Usage Note: This is an append-only style store. When records are removed (i.e., unlinked from their respective lists), the underlying space they occupied is **not** reclaimed or reused. As a result, memory usage will only grow as new records are added. To compact the store and reclaim the space from deleted records, the store must be rebuilt. The correct procedure is to create a new, empty store instance and transfer all valid records from the old store to the new one, for instance by iterating through a known set of root identifiers.

Methods

Clear()

Clears all data from the store and resets it to its initial state.

```
public void Clear()
```

Deserialize(byte[])

Deserializes a store from a byte array.

```
public static PrimitiveRecordStore Deserialize(byte[] data)
```

Parameters

`data` byte[]

The byte array containing the serialized data.

Returns

PrimitiveRecordStore

A new instance of PrimitiveRecordStore.

Exceptions

ArgumentNullException

`data` is null.

InvalidDataException

The data is in an unsupported format or is corrupted.

Deserialize(Stream)

Deserializes a store from a stream.

```
public static PrimitiveRecordStore Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

PrimitiveRecordStore

A new instance of PrimitiveRecordStore.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The data is in an unsupported format or is corrupted.

Deserialize(string)

Deserializes a store from a file.

```
public static PrimitiveRecordStore Deserialize(string file)
```

Parameters

`file` string

The path of the file to read from.

Returns

PrimitiveRecordStore

A new instance of PrimitiveRecordStore.

Exceptions

ArgumentNullException

`file` is null.

ArgumentException

`file` is empty or whitespace.

InvalidDataException

The data is in an unsupported format or is corrupted.

GetRecordAccess(int)

Gets a PrimitiveRecordStore.RecordAccess handle for a linked list of records.

```
public PrimitiveRecordStore.RecordAccess GetRecordAccess(int identifier = 0)
```

Parameters

`identifier` int

The identifier of the record list. Specify 0 to create a new list and get its identifier.

Returns

PrimitiveRecordStore.RecordAccess

A PrimitiveRecordStore.RecordAccess object to manage the specified record list.

Exceptions

ArgumentOutOfRangeException

`identifier` is negative.

ArgumentException

The specified `identifier` is invalid or does not exist.

Serialize(PrimitiveRecordStore, SerializationOptions?)

Serializes the specified store into a byte array.

```
public static byte[] Serialize(PrimitiveRecordStore store,
PrimitiveRecordStore.SerializationOptions? options = null)
```

Parameters

`store` PrimitiveRecordStore

The store to serialize.

`options` PrimitiveRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Returns

byte[]

A byte array containing the serialized data.

Exceptions

ArgumentNullException

`store` is null.

Serialize(PrimitiveRecordStore, Stream, SerializationOptions?)

Serializes the specified store into a stream.

```
public static void Serialize(PrimitiveRecordStore store, Stream stream,
PrimitiveRecordStore.SerializationOptions? options = null)
```

Parameters

`store` PrimitiveRecordStore

The store to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` PrimitiveRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Exceptions

ArgumentNullException

`store` or `stream` is null.

Serialize(PrimitiveRecordStore, string, SerializationOptions?)

Serializes the specified store to a file.

```
public static void Serialize(PrimitiveRecordStore store, string file,
PrimitiveRecordStore.SerializationOptions? options = null)
```

Parameters

`store` PrimitiveRecordStore

The store to serialize.

`file` string

The path of the file to create.

`options` PrimitiveRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Exceptions

ArgumentNullException

`store` or `file` is null.

ArgumentException

`file` is empty or whitespace.

Class PrimitiveRecordStore.Record

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Represents a handle to a single record within the store.

```
public sealed class PrimitiveRecordStore.Record
```

Inheritance

object ← PrimitiveRecordStore.Record

Inherited Members

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

Remarks

This object is a proxy that provides methods to read, update, and navigate the record's data and relationships. Operations performed on this object modify the underlying data in the parent PrimitiveRecordStore.

Properties

Actual

Gets the actual size of the record's content in bytes.

```
public int Actual { get; }
```

Property Value

int

Capacity

Gets the total allocated storage capacity for the record's content in bytes.

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

Property Value

int

ExByte

Gets or sets an extra, user-definable byte of metadata associated with this record. The value must be in the range of 0 to 255.

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

Property Value

int

Exceptions

ArgumentOutOfRangeException

> The assigned value is less than 0 or greater than 255.

Identifier

Gets the unique, persistent identifier for this record within the store.

```
public int Identifier { get; }
```

Property Value

int

Next

Gets the next record in the linked list, or null if this is the last record.

```
public PrimitiveRecordStore.Record? Next { get; }
```

Property Value

PrimitiveRecordStore.Record

Methods

Clear()

Clears the content of the record, setting its length to zero.

```
public void Clear()
```

InsertAfter(ReadOnlySpan<byte>)

Inserts a new record with the specified content immediately after this record in the list.

```
public PrimitiveRecordStore.Record InsertAfter(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record to insert.

Returns

PrimitiveRecordStore.Record

A PrimitiveRecordStore.Record handle for the newly inserted record.

InsertBefore(ReadOnlySpan<byte>)

Inserts a new record with the specified content immediately before this record in the list.

```
public PrimitiveRecordStore.Record InsertBefore(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record to insert.

Returns

PrimitiveRecordStore.Record

A PrimitiveRecordStore.Record handle for the newly inserted record.

Read()

Reads the content of the record.

```
public ReadOnlySpan<byte> Read()
```

Returns

ReadOnlySpan<byte>

A read-only span of bytes representing the record's content.

Remove()

Removes this record from the linked list it belongs to.

```
public void Remove()
```

Update(ReadOnlySpan<byte>)

Updates the content of the record.

```
public void Update(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The new content for the record.

Class PrimitiveRecordStore.RecordAccess

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Provides access to a linked list of records within a PrimitiveRecordStore.

```
public sealed class PrimitiveRecordStore.RecordAccess :
IEnumerable<PrimitiveRecordStore.Record>, IEnumerable
```

Inheritance

object ← PrimitiveRecordStore.RecordAccess

Implements

IEnumerable<PrimitiveRecordStore.Record>, IEnumerable

Inherited Members

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

Remarks

This class implements IEnumerable<T> to allow iterating over the records in the list. The collection will throw an InvalidOperationException if modified during enumeration.

Properties

Identifier

Gets the identifier for the head of the linked list that this object accesses.

```
public int Identifier { get; }
```

Property Value

int

Store

Gets the parent PrimitiveRecordStore that this accessor belongs to.

```
public PrimitiveRecordStore Store { get; }
```

Property Value

PrimitiveRecordStore

Methods

Add(ReadOnlySpan<byte>)

Adds a new record with the specified content to the end of the list.

```
public PrimitiveRecordStore.Record Add(ReadOnlySpan<byte> content)
```

Parameters

`content` ReadOnlySpan<byte>

The content of the new record.

Returns

PrimitiveRecordStore.Record

A PrimitiveRecordStore.Record handle for the newly added record.

Exceptions

InvalidOperationException

The operation is not supported for this accessor instance.

ArgumentOutOfRangeException

The length of `content` exceeds the maximum allowed size (4096 bytes).

Clear()

Removes all records from this list.

```
public void Clear()
```

Exceptions

InvalidOperationException

The operation is not supported for this accessor instance.

GetEnumerator()

Returns an enumerator that iterates through the records in the list.

```
public IEnumerator<PrimitiveRecordStore.Record> GetEnumerator()
```

Returns

IEnumerator <PrimitiveRecordStore.Record>

An enumerator for this record list.

Class PrimitiveRecordStore.SerializationOptions

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Provides options to control the serialization process for data store.

```
public class PrimitiveRecordStore.SerializationOptions
```

Inheritance

object ← PrimitiveRecordStore.SerializationOptions

Inherited Members

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

Properties

CompressionLevel

Gets or sets the compression level to use for serialization.

```
public CompressionLevel CompressionLevel { get; set; }
```

Property Value

CompressionLevel

Remarks

Higher compression levels can result in smaller file sizes but may take longer to process. Defaults to Fastest for a balance of speed and size.

Default

Gets a singleton instance of PrimitiveRecordStore.SerializationOptions with default values.

```
public static PrimitiveRecordStore.SerializationOptions Default { get; }
```

Property Value

PrimitiveRecordStore.SerializationOptions

Remarks

Use this property to avoid creating a new options object when default settings are sufficient.

Class RankSelectBitSet

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

An immutable bit set optimized for high-performance Rank and Select operations.

```
public sealed class RankSelectBitSet : ImmutableBitSet
```

Inheritance

object ← ImmutableBitSet ← RankSelectBitSet

Inherited Members

ImmutableBitSet.Buffer , ImmutableBitSet.Count , ImmutableBitSet.this[int] , ImmutableBitSet.At(int) , ImmutableBitSet.ToRankSelect(bool) , ImmutableBitSet.ToRankSelect(ImmutableBitSet, bool) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This data structure uses auxiliary indexes (lookup tables) to perform Rank (counting bits) and Select (finding the n-th bit) operations in near-constant time, making it suitable for succinct data structures like LOUDS Tries.

Constructors

RankSelectBitSet(ulong[], int)

An immutable bit set optimized for high-performance Rank and Select operations.

```
public RankSelectBitSet(ulong[] buffer, int count)
```

Parameters

`buffer` ulong[]

The underlying ulong array that stores the bits.

`count` int

The total number of bits in the set.

Remarks

This data structure uses auxiliary indexes (lookup tables) to perform Rank (counting bits) and Select (finding the n-th bit) operations in near-constant time, making it suitable for succinct data structures like LOUDS Tries.

Methods

CreateAuxDir(ImmutableBitSet)

Creates the auxiliary directories required for fast Rank and Select operations from a given bit set.

```
public static (int[], short[], int[]) CreateAuxDir(ImmutableBitSet bitSet)
```

Parameters

`bitSet` ImmutableBitSet

The immutable bit set to process.

Returns

(int[], short[], int[])

A tuple containing the generated directories for rank and select.

GetAuxDir()

Gets the pre-calculated auxiliary directories for Rank and Select operations.

```
public (int[], short[], int[]) GetAuxDir()
```

Returns

(int[], short[], int[])

A tuple containing the rank primary directory, rank secondary directory, and the select directory.

LastRank1()

Gets the total number of set bits (1s) in the entire bit set.

```
public int LastRank1()
```

Returns

int

The total count of set bits.

Rank0(int)

Calculates the rank of a bit '0' at a specified position. Rank0 is the total number of unset bits (0s) up to, but not including, the given position `k`.

```
public int Rank0(int k)
```

Parameters

`k` int

The zero-based position index.

Returns

int

The number of unset bits before position `k`.

Rank1(int)

Calculates the rank of a bit '1' at a specified position. Rank is the total number of set bits (1s) up to and including the given position `k`.

```
public int Rank1(int k)
```

Parameters

k int

The zero-based position index.

Returns

int

The number of set bits up to and including position k.

Select0(int)

Finds the position of the k-th unset bit (0).

```
public int Select0(int k)
```

Parameters

k int

The one-based rank of the unset bit to find (e.g., k=1 for the first '0').

Returns

int

The zero-based index of the k-th unset bit, or -1 if not found.

Select1(int)

Finds the position of the k-th set bit (1).

```
public int Select1(int k)
```

Parameters

k int

The one-based rank of the set bit to find (e.g., k=1 for the first '1').

Returns

int

The zero-based index of the k-th set bit, or -1 if not found.

SetAuxDir((int[], short[], int[]))

Sets the pre-calculated auxiliary directories for Rank and Select operations.

```
public void SetAuxDir((int[], short[], int[]) auxDir)
```

Parameters

`auxDir` (int[], short[], int[])

A tuple containing the rank primary directory, rank secondary directory, and the select directory.

Class ScapegoatTreeRecordStore

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of BinaryTreeRecordStore that uses a self-balancing Scapegoat tree.

```
public sealed class ScapegoatTreeRecordStore : BinaryTreeRecordStore, ICloneable
```

Inheritance

object ← BasicRecordStore ← BinaryTreeRecordStore ← ScapegoatTreeRecordStore

Implements

ICloneable

Inherited Members

BinaryTreeRecordStore.Clear() , BinaryTreeRecordStore.Contains(int) , BinaryTreeRecordStore.GetRecordAccess(int, out bool) , BinaryTreeRecordStore.TryGetRecordAccess(int, out BasicRecordStore.RecordAccess) , BinaryTreeRecordStore.Enumerate() , BinaryTreeRecordStore.PrintTree(TextWriter) , BasicRecordStore.Serialize<T>(T, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, string, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, Stream, BasicRecordStore.SerializationOptions) , BasicRecordStore.Deserialize<T>(byte[]) , BasicRecordStore.Deserialize<T>(string) , BasicRecordStore.Deserialize<T>(Stream) , BasicRecordStore.Add(int) , BasicRecordStore.GetRecordAccess(int) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

A Scapegoat tree is a self-balancing binary search tree that provides amortized time efficiency. Instead of rebalancing on every operation, it rebuilds subtrees only when an insertion or deletion causes a node (the "scapegoat") to become too unbalanced. Note: This implementation can be slow on certain operations that trigger a full subtree rebuild and is generally not recommended for performance-critical applications.

Serialization Limits: The binary serialization format for this class imposes certain constraints on the data. Exceeding these limits will result in an InvalidDataException during serialization.

- **Max Records per List:** A single identifier can have a maximum of 16,777,215 records in its linked list.

- **Max Record Content Size:** The `Content` byte array of any single record cannot exceed 65,535 bytes.

Methods

Clone()

Creates a deep copy of the ScapegoatTreeRecordStore.

```
public object Clone()
```

Returns

object

A new ScapegoatTreeRecordStore instance with the same structure and record data as the original.

Remarks

The method creates a new tree and copies all records, ensuring that the new store is independent of the original.

Deserialize(Stream)

Deserializes a ScapegoatTreeRecordStore from a stream.

```
public static ScapegoatTreeRecordStore Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

ScapegoatTreeRecordStore

A new instance of ScapegoatTreeRecordStore reconstructed from the stream.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The stream data is corrupted, in an unsupported format, or contains invalid values.

IsBalanced()

Overrides the base method to check if the tree conforms to the alpha-weight-balanced property of a tree.

```
public override bool IsBalanced()
```

Returns

bool

true if no subtree violates the alpha-weight balance condition; otherwise, false.

Remove(int)

Removes the node with the specified identifier from the tree. May trigger a rebuild of the entire tree if the number of nodes drops significantly.

```
public override void Remove(int identifier)
```

Parameters

`identifier` int

The identifier of the node to remove.

Exceptions

ArgumentOutOfRangeException

`identifier` is MinValue.

Serialize(ScapegoatTreeRecordStore, Stream, SerializationOptions?)

Serializes the entire state of the tree store into a stream.

```
public static void Serialize(ScapegoatTreeRecordStore store, Stream stream,
BasicRecordStore.SerializationOptions? options = null)
```

Parameters

`store` ScapegoatTreeRecordStore

The ScapegoatTreeRecordStore instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` BasicRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is specific to this tree implementation, using Brotli compression and an XxHash32 checksum for data integrity.

Exceptions

ArgumentNullException

`store` or `stream` is null.

Class TreapRecordStore

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

A concrete implementation of BinaryTreeRecordStore that uses a Treap.

```
public class TreapRecordStore : BinaryTreeRecordStore, ICloneable
```

Inheritance

object ← BasicRecordStore ← BinaryTreeRecordStore ← TreapRecordStore

Implements

ICloneable

Inherited Members

BinaryTreeRecordStore.Clear() , BinaryTreeRecordStore.Contains(int) , BinaryTreeRecordStore.GetRecordAccess(int, out bool) , BinaryTreeRecordStore.TryGetRecordAccess(int, out BasicRecordStore.RecordAccess) , BinaryTreeRecordStore.Enumerate() , BinaryTreeRecordStore.PrintTree(TextWriter) , BasicRecordStore.Serialize<T>(T, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, string, BasicRecordStore.SerializationOptions) , BasicRecordStore.Serialize<T>(T, Stream, BasicRecordStore.SerializationOptions) , BasicRecordStore.Deserialize<T>(byte[]) , BasicRecordStore.Deserialize<T>(string) , BasicRecordStore.Deserialize<T>(Stream) , BasicRecordStore.Add(int) , BasicRecordStore.GetRecordAccess(int) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

A Treap is a randomized binary search tree. It maintains the binary search tree property for its keys (identifiers) and the max-heap property for randomly assigned priorities. This combination maintains balance with high probability, providing good average-case performance for insertions, deletions, and lookups.

Serialization Limits: The binary serialization format for this class imposes certain constraints on the data. Exceeding these limits will result in an InvalidDataException during serialization.

- **Max Records per List:** A single identifier can have a maximum of 16,777,215 records in its linked list.
- **Max Record Content Size:** The `Content` byte array of any single record cannot exceed 65,535 bytes.

Methods

Clone()

Creates a deep copy of the TreapRecordStore.

```
public virtual object Clone()
```

Returns

object

A new TreapRecordStore instance with the same record data as the original.

Remarks

The method creates a new tree and copies all records. The new tree will have newly randomized priorities for its nodes, so its internal structure may differ from the original, but it will be functionally equivalent.

Deserialize(Stream)

Deserializes a TreapRecordStore from a stream.

```
public static TreapRecordStore Deserialize(Stream stream)
```

Parameters

`stream` Stream

The stream to read the serialized data from.

Returns

TreapRecordStore

A new instance of TreapRecordStore reconstructed from the stream.

Exceptions

ArgumentNullException

`stream` is null.

InvalidDataException

The stream data is corrupted, in an unsupported format, or contains invalid values.

IsBalanced()

Overrides the base method to check if the tree is a valid tree.

```
public override bool IsBalanced()
```

Returns

bool

true if the tree satisfies both the binary search tree property (for identifiers) and the max-heap property (for priorities); otherwise, false.

Remove(int)

Removes the node with the specified identifier from the tree, performing rotations to maintain the heap property before removal.

```
public override void Remove(int identifier)
```

Parameters

`identifier` int

The identifier of the node to remove.

Exceptions

ArgumentOutOfRangeException

`identifier` is MinValue.

Serialize(TreapRecordStore, Stream, SerializationOptions?)

Serializes the entire state of the tree, including node priorities, into a stream.

```
public static void Serialize(TreapRecordStore store, Stream stream,
BasicRecordStore.SerializationOptions? options = null)
```

Parameters

`store` TreapRecordStore

The TreapRecordStore instance to serialize.

`stream` Stream

The stream to write the serialized data to.

`options` BasicRecordStore.SerializationOptions

Options to control the serialization process. If null, the settings from Default will be used.

Remarks

The serialization format is specific to this tree implementation, using Brotli compression and an XxHash32 checksum for data integrity.

Exceptions

ArgumentNullException

`store` or `stream` is null.

Class ValueBuffer<T>

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Represents a buffer for value types, implemented using a list of array chunks.

```
public sealed class ValueBuffer<T> : ICollection<T>, IEnumerable<T>, IEnumerable where T
: struct
```

Type Parameters

T

The type of element, which must be a struct.

Inheritance

object ← ValueBuffer<T>

Implements

ICollection<T>, IEnumerable<T>, IEnumerable

Inherited Members

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

Remarks

This class is designed to manage large collections of value types without allocating on the Large Object Heap (LOH). It achieves this by storing elements in a series of smaller array "chunks". The size of these chunks, specified by ElementsPerChunk, must be a power of 2.

Constructors

ValueBuffer(int, bool)

Initializes a new instance of the ValueBuffer<T> class.

```
public ValueBuffer(int elementsPerChunk, bool extendAutomatically = false)
```

Parameters

`elementsPerChunk` int

The size of each internal array chunk. This value must be a power of 2.

`extendAutomatically` bool

A value indicating whether to automatically extend the buffer's capacity when an out-of-bounds index is accessed.

Exceptions

ArgumentOutOfRangeException

`elementsPerChunk` is not positive.

ArgumentException

`elementsPerChunk` is not a power of 2.

Properties

Capacity

Gets the total number of elements that can be stored across all currently allocated chunks.

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

Property Value

int

Count

Gets the number of elements contained in the ValueBuffer<T>. This property is an implementation for the ICollection<T> interface and is an alias for the Used property.

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

Property Value

int

ElementsPerChunk

Gets the size of each internal array chunk.

```
public int ElementsPerChunk { get; }
```

Property Value

int

ExtendAutomatically

Gets a value indicating whether the buffer will automatically allocate new chunks when an index outside the current capacity is accessed.

```
public bool ExtendAutomatically { get; }
```

Property Value

bool

IsReadOnly

Gets a value indicating whether the ValueBuffer<T> is read-only.

```
public bool IsReadOnly { get; }
```

Property Value

bool

this[int]

Gets or sets the element at the specified index.

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

Parameters

`index` int

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

Property Value

T

A reference to the element at the specified index.

Remarks

Accessing an index greater than or equal to the current Used count will update the count to `index + 1`. If ExtendAutomatically is true, accessing an index beyond the current Capacity will allocate new chunks.

Exceptions

ArgumentOutOfRangeException

Thrown if `index` is negative, or if `index` is greater than or equal to Capacity and ExtendAutomatically is false.

NumberOfChunks

Gets the number of internal array chunks that are currently allocated.

```
public int NumberOfChunks { get; }
```

Property Value

int

Used

Gets the number of elements that are considered to be in use within the buffer. This value is updated when an element is written to an index greater than the current used count.

```
public int Used { get; }
```

Property Value

int

Methods

Add(T)

Adds an element to the end of the ValueBuffer<T>.

```
public void Add(T value)
```

Parameters

`value` T

The element to add.

AppendChunkDirectly(T[])

Appends a pre-allocated array chunk directly to the internal list of buffers, increasing the capacity.

```
public void AppendChunkDirectly(T[] chunk)
```

Parameters

`chunk` T[]

The array to append. Its length must be exactly equal to ElementsPerChunk.

Remarks

This is an advanced optimization method that avoids an array allocation and copy by using a caller-provided array.

Exceptions

ArgumentNullException

`chunk` is null.

ArgumentException

The length of `chunk` is not equal to ElementsPerChunk.

Clear()

Removes all elements from the ValueBuffer<T> and releases all internal array chunks.

```
public void Clear()
```

Contains(T)

Determines whether the ValueBuffer<T> contains a specific value.

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

Parameters

`item` T

The object to locate in the ValueBuffer<T>.

Returns

bool

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

CopyTo(T[], int)

Copies the elements of the ValueBuffer<T> to an Array, starting at a particular Array index.

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

Parameters

`array` T[]

The one-dimensional Array that is the destination of the elements copied from ValueBuffer<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 source buffer is greater than the available space from `arrayIndex` to the end of the destination `array`.

EnumerateChunks()

Returns an enumerable collection of the internal array chunks.

```
public IEnumerable<T[]> EnumerateChunks()
```

Returns

IEnumerable<T[]>

An IEnumerable<T> (`T` is an array) that allows iteration over the raw internal buffers.

Remarks

This method is intended for advanced scenarios where direct read-only access to the underlying chunks is required.

ExtendCapacity(int)

Ensures that the capacity of this buffer is at least the specified value.

```
public int ExtendCapacity(int minCapacity)
```

Parameters

`minCapacity` int

The minimum required capacity.

Returns

int

The new capacity of the buffer, which will be a multiple of ElementsPerChunk.

Exceptions

ArgumentOutOfRangeException

`minCapacity` is negative.

Fill(T, int, int)

Fills a range of elements in the buffer with a specified value.

```
public void Fill(T value, int index, int count)
```

Parameters

`value` T

The value to assign to each element in the range.

`index` int

The zero-based starting index of the range to fill.

`count` int

The number of elements in the range to fill.

Exceptions

ArgumentOutOfRangeException

`index` or `count` is negative.

ArgumentException

The sum of `index` and `count` is greater than the buffer's capacity and ExtendAutomatically is false.

GetEnumerator()

Returns an enumerator that iterates through the used elements in the buffer.

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

Returns

IEnumerator <T>

An enumerator for the ValueBuffer<T>.

Remove(T)

Finds the first occurrence of an item and replaces it with the default value of type `T`.

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

Parameters

`item` T

The item to locate and replace.

Returns

bool

true if the item was found and replaced; otherwise, false.

Remarks

This method does not actually remove the element in a way that would shift subsequent elements. It only overwrites the found element with `default(T)`. The Count and Used properties are not changed by this operation.

SetUsedDirectly(int, bool)

Directly sets the number of used elements and optionally shrinks the buffer's capacity.

```
public void SetUsedDirectly(int used, bool shrinkAutomatically = false)
```

Parameters

`used` int

The new count of used elements. This value must be between 0 and Capacity.

`shrinkAutomatically` bool

If true, deallocates unused array chunks from the end of the buffer to fit the new used count.

Remarks

This is an advanced method and should be used with caution. Setting an incorrect value can lead to an inconsistent state or data corruption.

Exceptions

ArgumentOutOfRangeException

`used` is negative or greater than Capacity.

ToArray()

Copies the used elements from the ValueBuffer<T> to a new, single contiguous array.

```
public T[] ToArray()
```

Returns

T[]

A new array containing the elements from the buffer.

Class Xoroshiro128PlusPlus

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Represents a 64-bit pseudo-random number generator based on the xoroshiro128++ algorithm.

```
public class Xoroshiro128PlusPlus : ICloneable
```

Inheritance

object ← Xoroshiro128PlusPlus

Implements

ICloneable

Inherited Members

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

Remarks

This generator is not cryptographically secure.

Constructors

Xoroshiro128PlusPlus()

Initializes a new instance of the Xoroshiro128PlusPlus class, using a unique seed value derived from a new GUID.

```
public Xoroshiro128PlusPlus()
```

Xoroshiro128PlusPlus(Xoroshiro128PlusPlus)

Initializes a new instance of the Xoroshiro128PlusPlus class by copying the state from another instance.

```
public Xoroshiro128PlusPlus(Xoroshiro128PlusPlus obj)
```

Parameters

`obj` Xoroshiro128PlusPlus

The Xoroshiro128PlusPlus instance to copy the state from.

Remarks

This constructor is primarily intended for replicating the state of the generator for parallel processing scenarios. After creating a copy, call the Jump() method on one of the instances to ensure that each generator produces a non-overlapping sequence of random numbers.

Exceptions

ArgumentNullException

`obj` is null.

Xoroshiro128PlusPlus(ulong)

Initializes a new instance of the Xoroshiro128PlusPlus class, using the specified seed value.

```
public Xoroshiro128PlusPlus(ulong seed)
```

Parameters

`seed` ulong

A number used to calculate a starting value for the pseudo-random number sequence.

Methods

Clone()

Creates a new object that is a shallow copy of the current instance.

```
public object Clone()
```

Returns

object

A new object that is a shallow copy of this instance.

Remarks

This constructor is primarily intended for replicating the state of the generator for parallel processing scenarios. After creating a copy, call the Jump() method on one of the instances to ensure that each generator produces a non-overlapping sequence of random numbers.

Jump()

Advances the state of the generator by 2^64 steps.

```
public void Jump()
```

Remarks

This is equivalent to generating 2^64 random numbers and discarding them, but it is executed much faster. It is useful for creating non-overlapping subsequences for parallel random number generation.

Next()

Returns a non-negative random integer.

```
public int Next()
```

Returns

int

A 32-bit signed integer that is greater than or equal to 0 and less than MaxValue.

Next(int)

Returns a non-negative random integer that is less than the specified maximum.

```
public int Next(int maxValue)
```

Parameters

`maxValue` int

The exclusive upper bound of the random number to be generated. `maxValue` must be greater than or equal to 0.

Returns

int

A 32-bit signed integer that is greater than or equal to 0, and less than `maxValue`.

Remarks

If `maxValue` is 0, this method will always return 0.

Next(int, int)

Returns a random integer that is within a specified range.

```
public int Next(int minValue, int maxValue)
```

Parameters

`minValue` int

The inclusive lower bound of the random number returned.

`maxValue` int

The exclusive upper bound of the random number returned. `maxValue` must be greater than or equal to `minValue`.

Returns

int

A 32-bit signed integer greater than or equal to `minValue` and less than `maxValue`.

Remarks

If `minValue` equals `maxValue`, this method returns `minValue`.

Exceptions

ArgumentOutOfRangeException

`minValue` is greater than `maxValue`.

NextBytes(Span<byte>)

Fills the elements of a specified span of bytes with random numbers.

```
public void NextBytes(Span<byte> buffer)
```

Parameters

`buffer` Span<byte>

The span of bytes to fill with random numbers.

NextDouble()

Returns a random double-precision floating-point number that is greater than or equal to 0.0, and less than 1.0.

```
public double NextDouble()
```

Returns

double

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

NextInt64()

Returns a non-negative random 64-bit integer.

```
public long NextInt64()
```

Returns

long

A 64-bit signed integer that is greater than or equal to 0 and less than MaxValue.

NextInt64(long)

Returns a non-negative random 64-bit integer that is less than the specified maximum.

```
public long NextInt64(long maxValue)
```

Parameters

`maxValue` long

The exclusive upper bound of the random number to be generated. `maxValue` must be greater than or equal to 0.

Returns

long

A 64-bit signed integer that is greater than or equal to 0, and less than `maxValue`.

Remarks

If `maxValue` is 0, this method will always return 0.

NextInt64(long, long)

Returns a random 64-bit integer that is within a specified range.

```
public long NextInt64(long minValue, long maxValue)
```

Parameters

`minValue` long

The inclusive lower bound of the random number returned.

`maxValue` long

The exclusive upper bound of the random number returned. `maxValue` must be greater than or equal to `minValue`.

Returns

long

A 64-bit signed integer greater than or equal to `minValue` and less than `maxValue`.

Remarks

If `minValue` equals `maxValue`, this method returns `minValue`.

Exceptions

ArgumentOutOfRangeException

`minValue` is greater than `maxValue`.

NextSingle()

Returns a random single-precision floating-point number that is greater than or equal to 0.0, and less than 1.0.

```
public float NextSingle()
```

Returns

float

A single-precision floating-point number that is greater than or equal to 0.0f and less than 1.0f.

Shuffle<T>(IList<T>)

Randomizes the order of the elements in a list.

```
public void Shuffle<T>(IList<T> values)
```

Parameters

`values` IList<T>

The list whose elements to shuffle.

Type Parameters

`T`

The type of the elements of the list.

Remarks

This method uses the Fisher-Yates shuffle algorithm.

Exceptions

ArgumentNullException

`values` is null.

Shuffle<T>(Span<T>)

Randomizes the order of the elements in a span.

```
public void Shuffle<T>(Span<T> values)
```

Parameters

`values` Span<T>

The span whose elements to shuffle.

Type Parameters

`T`

The type of the elements of the span.

Remarks

This method uses the Fisher-Yates shuffle algorithm.

Class Xoshiro256PlusPlus

Namespace: BelNytheraSeiche.TrieDictionary

Assembly: BelNytheraSeiche.TrieDictionary.dll

Represents a 64-bit pseudo-random number generator based on the xoshiro256++ algorithm.

```
public class Xoshiro256PlusPlus : ICloneable
```

Inheritance

object ← Xoshiro256PlusPlus

Implements

ICloneable

Inherited Members

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

Remarks

This generator is not cryptographically secure.

Constructors

Xoshiro256PlusPlus()

Initializes a new instance of the Xoshiro256PlusPlus class, using a unique seed value derived from a new GUID.

```
public Xoshiro256PlusPlus()
```

Xoshiro256PlusPlus(Xoshiro256PlusPlus)

Initializes a new instance of the Xoshiro256PlusPlus class by copying the state from another instance.

```
public Xoshiro256PlusPlus(Xoshiro256PlusPlus obj)
```

Parameters

`obj` Xoshiro256PlusPlus

The Xoshiro256PlusPlus instance to copy the state from.

Remarks

This constructor is primarily intended for replicating the state of the generator for parallel processing scenarios. After creating a copy, call the Jump() method on one of the instances to ensure that each generator produces a non-overlapping sequence of random numbers.

Exceptions

ArgumentNullException

`obj` is null.

Xoshiro256PlusPlus(ulong)

Initializes a new instance of the Xoshiro256PlusPlus class, using the specified seed value.

```
public Xoshiro256PlusPlus(ulong seed)
```

Parameters

`seed` ulong

A number used to calculate a starting value for the pseudo-random number sequence.

Methods

Clone()

Creates a new object that is a shallow copy of the current instance.

```
public object Clone()
```

Returns

object

A new object that is a shallow copy of this instance.

Remarks

This constructor is primarily intended for replicating the state of the generator for parallel processing scenarios. After creating a copy, call the Jump() method on one of the instances to ensure that each generator produces a non-overlapping sequence of random numbers.

Jump()

Advances the state of the generator by 4^64 steps.

```
public void Jump()
```

Remarks

This is equivalent to generating 4^64 random numbers and discarding them, but it is executed much faster. It is useful for creating non-overlapping subsequences for parallel random number generation.

Next()

Returns a non-negative random integer.

```
public int Next()
```

Returns

int

A 32-bit signed integer that is greater than or equal to 0 and less than MaxValue.

Next(int)

Returns a non-negative random integer that is less than the specified maximum.

```
public int Next(int maxValue)
```

Parameters

`maxValue` int

The exclusive upper bound of the random number to be generated. `maxValue` must be greater than or equal to 0.

Returns

int

A 32-bit signed integer that is greater than or equal to 0, and less than `maxValue`.

Remarks

If `maxValue` is 0, this method will always return 0.

Next(int, int)

Returns a random integer that is within a specified range.

```
public int Next(int minValue, int maxValue)
```

Parameters

`minValue` int

The inclusive lower bound of the random number returned.

`maxValue` int

The exclusive upper bound of the random number returned. `maxValue` must be greater than or equal to `minValue`.

Returns

int

A 32-bit signed integer greater than or equal to `minValue` and less than `maxValue`.

Remarks

If `minValue` equals `maxValue`, this method returns `minValue`.

Exceptions

ArgumentOutOfRangeException

`minValue` is greater than `maxValue`.

NextBytes(Span<byte>)

Fills the elements of a specified span of bytes with random numbers.

```
public void NextBytes(Span<byte> buffer)
```

Parameters

`buffer` Span<byte>

The span of bytes to fill with random numbers.

NextDouble()

Returns a random double-precision floating-point number that is greater than or equal to 0.0, and less than 1.0.

```
public double NextDouble()
```

Returns

double

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

NextInt64()

Returns a non-negative random 64-bit integer.

```
public long NextInt64()
```

Returns

long

A 64-bit signed integer that is greater than or equal to 0 and less than MaxValue.

NextInt64(int)

Returns a non-negative random 64-bit integer that is less than the specified maximum.

```
public long NextInt64(int maxValue)
```

Parameters

`maxValue` int

The exclusive upper bound of the random number to be generated. `maxValue` must be greater than or equal to 0.

Returns

long

A 64-bit signed integer that is greater than or equal to 0, and less than `maxValue`.

Remarks

If `maxValue` is 0, this method will always return 0.

NextInt64(long, long)

Returns a random 64-bit integer that is within a specified range.

```
public long NextInt64(long minValue, long maxValue)
```

Parameters

`minValue` long

The inclusive lower bound of the random number returned.

`maxValue` long

The exclusive upper bound of the random number returned. `maxValue` must be greater than or equal to `minValue`.

Returns

long

A 64-bit signed integer greater than or equal to `minValue` and less than `maxValue`.

Remarks

If `minValue` equals `maxValue`, this method returns `minValue`.

Exceptions

ArgumentOutOfRangeException

`minValue` is greater than `maxValue`.

NextSingle()

Returns a random single-precision floating-point number that is greater than or equal to 0.0, and less than 1.0.

```
public float NextSingle()
```

Returns

float

A single-precision floating-point number that is greater than or equal to 0.0f and less than 1.0f.

Shuffle<T>(IList<T>)

Randomizes the order of the elements in a list.

```
public void Shuffle<T>(IList<T> values)
```

Parameters

`values` IList<T>

The list whose elements to shuffle.

Type Parameters

`T`

The type of the elements of the list.

Remarks

This method uses the Fisher-Yates shuffle algorithm.

Exceptions

ArgumentNullException

`values` is null.

Shuffle<T>(Span<T>)

Randomizes the order of the elements in a span.

```
public void Shuffle<T>(Span<T> values)
```

Parameters

`values` Span<T>

The span whose elements to shuffle.

Type Parameters

`T`

The type of the elements of the span.

Remarks

This method uses the Fisher-Yates shuffle algorithm.