C-Sharp Mapping for Dictionaries

Ice for .NET supports three different mappings for dictionaries. By default, dictionaries are mapped to `System.Collections.Generic.Dictionary<T>`. You can use metadata directives to map dictionaries to two other types:

- `System.Collections.Generic.SortedDictionary`
- Types derived from `Ice.DictionaryBase`, which is a drop-in replacement for `System.Collections.DictionaryBase` (this mapping is provided mainly for compatibility with Ice versions prior to 3.3)

On this page:

Mapping to Predefined Containers for Dictionaries in C#

Here is the definition of our EmployeeMap once more:

Slice

```
dictionary<long, Employee> EmployeeMap;
```

By default, the Slice-to-C# compiler maps the dictionary to the following type:

C#

```
System.Collections.Generic.Dictionary<long, Employee>
```

You can use the `"clr:generic:SortedDictionary"` metadata directive to change the mapping to a sorted dictionary:

Slice

```
["clr:generic:SortedDictionary"]
dictionary<long, Employee> EmployeeMap;
```

With this definition, the type of the dictionary becomes:

C#

```
System.Collections.Generic.SortedDictionary<long, Employee>
```

`DictionaryBase` mapping for Dictionaries in C#

The `DictionaryBase` mapping is provided mainly for compatibility with Ice versions prior to 3.3. Internally, `DictionaryBase` is implemented using `System.Collections.Generic.Dictionary<T>`, so it offers the same performance trade-offs as `Dictionary<T>`. (For value types, `Ice.DictionaryBase` is considerably faster than `System.Collections.DictionaryBase`, however.)

`Ice.DictionaryBase` is not as type-safe as `Dictionary<T>` because, in order to remain source code compatible with `System.Collections.DictionaryBase`, it provides methods that accept elements of type `object`. This means that, if you pass an element of the wrong type, the problem will be diagnosed only at run time, instead of at compile time. For this reason, we suggest that you do not use the `DictionaryBase` mapping for new code.

To enable the `DictionaryBase` mapping, you must use the `"clr:collection"` metadata directive:

Slice

```
["clr:collection"] dictionary<long, Employee> EmployeeMap;
```

With this directive, `slice2cs` generates a type that derives from `Ice.CollectionBase`:

C#

```
public class EmployeeMap : Ice.DictionaryBase<long, Employee>, System.ICloneable
{
    public void AddRange(EmployeeMap m);
    public object Clone();
}
```

Note that the generated `EmployeeMap` class derives from `Ice.DictionaryBase`, which provides a super-set of the interface of the .NET `System.Collections.DictionaryBase` class. Apart from methods inherited from `DictionaryBase`, the class provides a `Clone` method and an `AddRange` method that allows you to append the contents of one dictionary to another. If the target dictionary contains a key that is also in the source dictionary, the target dictionary's value is preserved. For example:

C#

```
Employee e1 = new Employee();
e1.number = 42;
e1.firstName = "Herb";
e1.lastName = "Sutter";

EmployeeMap em1 = new EmployeeMap();
em[42] = e;

Employee e2 = new Employee();
e2.number = 42;
e2.firstName = "Stan";
e2.lastName = "Lipmann";

EmployeeMap em2 = new EmployeeMap();
em[42] = e2;

// Add contents of em2 to em1
//
em1.AddRange(em2);

// Equal keys preserve the original value
//
Debug.Assert(em1[42].firstName.Equals("Herb"));
```

The `DictionaryBase` class provides the following methods:

C#

```
public abstract class DictionaryBase<KT, VT>
    : System.Collections.IDictionary
{
    public DictionaryBase();

    public int Count { get; }

    public void Add(KT key, VT value);
    public void Add(object key, object value);

    public void CopyTo(System.Array a, int index);

    public void Remove(KT key);
    public void Remove(object key);

    public void Clear();

    public System.Collections.ICollection Keys { get; }
    public System.Collections.ICollection Values { get; }

    public VT this[KT key] { get; set; }
    public object this[object key] { get; set; }

    public bool Contains(KT key);
    public bool Contains(object key);

    public override int GetHashCode();
    public override bool Equals(object other);
    public static bool operator==(DictionaryBase<KT, VT> lhs, DictionaryBase<KT, VT> rhs);
    public static bool operator!=(DictionaryBase<KT, VT> lhs, DictionaryBase<KT, VT> rhs);

    public System.Collections.IEnumerator GetEnumerator();

    public bool IsFixedSize { get; }
    public bool IsReadOnly { get; }
    public bool IsSynchronized { get; }
    public object SyncRoot { get; }
}
```

The methods have the same semantics as the corresponding methods in the .NET Framework. The `Equals` method returns true if two dictionaries contain the same number of entries and, for each entry, the key and value are the same (as determined by their `Equals` methods).

The `Clone` method performs a shallow copy.

The class also implements the inherited `IsFixedSize`, `IsReadOnly`, and `IsSynchronized` properties (which return false), and the `SyncRoot` property (which returns `this`).

See Also

- Metadata
- C-Sharp Mapping for Identifiers
- C-Sharp Mapping for Modules
- C-Sharp Mapping for Built-In Types
- C-Sharp Mapping for Enumerations
- C-Sharp Mapping for Structures
- C-Sharp Mapping for Sequences
- C-Sharp Collection Comparison
- C-Sharp Mapping for Constants
- C-Sharp Mapping for Exceptions