Technical Report

ECMA TR/89

2nd Edition / June 2006

Common Language Infrastructure (CLI)

Partitions I to VI

Partition I: Concepts and Architecture

Partition II: Metadata Definition and Semantics

Partition III: CIL Instruction Set

Partition IV: Profiles and Libraries

Partition V: Binary Formats

Partition VI: Annexes

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

ECMA TR/89

2nd Edition / June 2006

Common Language Infrastructure (CLI)

Partitions I to VI

Partition I: Concepts and Architecture

Partition II: Metadata Definition and Semantics

Partition III: CIL Instruction Set

Partition IV: Profiles and Libraries

Partition V: Binary Formats

Partition VI: Annexes

Introduction

This Technical Report defines a collection of types that are intended to enhance the common language nature of the CLI, by facilitating language inter-operation. The collection includes generic tuples, functions, actions, optional value representation, a type that can contain a value of one of two different types, and a utility filler type.

These types are experimental and will be considered for inclusion in a future version of the CLI Standard. A reference implementation, written in C#, is included (see the accompanying file CommonGenericsLibrary.cs). This implementation source is also available from `http://kahu.zoot.net.nz/ecma`. A binary version is also available from that site, along with any updates to the proposal.

Feedback on these types is encouraged. (Please send comments to ecmacli@zoot.net.nz.)

This second edition cancels and replaces the first edition. Changes from the previous edition were made to align this Standard with ISO/IEC TR 25438:2006.

This Ecma Technical Report has been adopted by the General Assembly of June 2006.

Table of contents

1 Scope

The CLI standard libraries (ISO/IEC 23271) provide a collection of common types that can be used by multiple languages. With the addition of generics to the CLI, the standard libraries have been extended to include a number of common generic types, in particular, collections. However, at present, these libraries do not include many simple generic types found in a number of different languages. Any language which uses these common types must implement them rather than deferring to the CLI library, thereby reducing language inter-operability. This proposal addresses this issue by providing a number of these common types.

Generic tuples (product types) are standard in a number of languages: C++ (`template Pair`), Ada, Haskell, and Standard ML (SML). However, languages differ in the number of pre-defined tuple sizes supported by their standard libraries; e.g., C++ provides just one (Pair) while Haskell provides eight (sizes 2 to 9) and SML allows any size of tuple. This proposal provides nine (sizes 2 to 10).

Generic programming encourages “higher order” programming where generic functions (methods) take function (delegate) type arguments that have generic types. Examples include Ada’s `with` and generic constraints, and function arguments in Haskell and SML. In the CLI, function values are provided in the form of delegates, so this proposal defines standard generic delegate types for functions (which return a value) and procedures (which do not).

Another two types that occur in a number of languages are an *optional* type, which either contains a value of some other type or an indication that such a value is not present; and an *either* type, which holds a value of one of two possible types and an indication of which one is present. This proposal provides both of these. (Note: The optional type is similar to, but different from, the type `System.Nullable`.)

Finally, in existing generic languages, a need has been found for a *filler* type to be used when a particular generic parameter is not required for a particular use of the generic type. A standard one-value type is often provided for this purpose, often called `Unit` or `Void`. This proposal includes such a type.

2 Rationale

2.1 Reference vs. value tuples

In some languages (such as C++ and Ada) tuple types can be value or reference types, while in others (such as Haskell and SML) they are always reference types. This implementation provides only value type tuples (which can contain value or reference types). Boxed versions of these types can be used when reference versions are required. Named boxed types would, of course, help greatly here, but these are not currently provided by the CLI Standard.

A simple generic type, such as `Boxed<T>`, can be written to address this issue. Such a type has been made available in the reference implementation of this proposal. However it is not being considered for standardization at this time.

2.2 Interaction with other standard types

The CLI Standard contains six types related to those in this proposal. These are:

`System.Action<T>`: This type is just the first member of this collection's set of action types. As it is already standardized it is not included here.

`System.Comparison<T>`: This type is equivalent to this collection's `Function<T, T, Int32>`. Conversion methods are provided.

`System.Converter<T,U>`: This type is equivalent to this collection's `Function<T, U>`. Conversion methods are provided.

`System.Nullable<T>`: This type is a restricted version of this collection's `Optional<T>`. The difference is that `Nullable<T>` is restricted to non-nullable value types, whereas `Optional<T>` can also wrap reference types and nullable value types. Both these choices are useful in different applications, so there is no clash between these two types. This proposal includes operators on the `Optional<T>` type to convert to/from `Nullable<T>` values.

`System.Predicate<T>`: This type is equivalent to this collection's `Function<T, Boolean>`. However as the CLI does not support type synonyms, these two types are (unfortunately) not equivalent. A separate static class provides methods to convert between these two types. The methods are in a separate class as additional methods cannot be declared on delegate types.

`System.Collections.Generic.KeyValuePair<K,V>`: This is just a differently named equivalent of this collection's `Tuple<K,V>`. Conversion operators are provided on `Tuple<K,V>` to convert to/from `KeyValuePair<K,V>`.

3 Overview

3.1 Tuple types

Tuples are provided as value types with public fields, which mirrors their typical usage in current languages. Overloading on arity is leveraged to provide eight standard tuple types.

The fields are named `ItemA`, `ItemB`, …, `ItemI`. A single n-ary constructor is provided to easily define all fields.

An `Equals()` method is provided and defined as the component fields being equal according to their `Equals()` methods.

The corresponding `GetHashCode()` method and `op_Equality` and `op_Inequality` methods are provided.

`ToString()` is overridden to provide the typical tuple parenthesized notation.

For tuples of size 2, methods are provided to convert to/from values of type `System.Collections.Generic.KeyValuePair<K, V>`.

3.2 Function and procedure types

Standard delegates are provided for function types (`System.Function<>`) and procedures (`System.Action<>`). Delegates are defined for 0 to 5 argument functions and 0 and 2 to 5 argument procedures. (The single argument procedure is already included in the CLI Standard.)

3.3 Unit

The `Unit` type is a filler type. This is useful, for example, when an existing generic type is used, but one of its type arguments is not required for the particular application.

3.4 Optional

The `Optional<A>` type represents either a value of type A, or the lack of a value. A value of this type is immutable and provides two constructors, one providing a value, the other for providing no value. Properties are provided to test for validity and to obtain the value. An exception is thrown if an attempt to access a non-existent value.

3.5 Either

The `Either<A, B>` type represents either a value of type A, or a value of type B. A value of this type is immutable. Properties are provided to test which type of value is present and to obtain the value. An exception is thrown if an attempt to access the value of one type is made when a value of the other type is present.

4 Action delegates

4.1 System.Action delegate

[ILAsm]
```
.class public auto ansi sealed System.Action
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance void  Invoke()
}
```

[C#]
```
public delegate void Action();
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs an action.

Parameters:

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for zero-argument, methods which perform an action (return void).

4.2 System.Action<A, B> delegate

[ILAsm]
```
.class public auto ansi sealed System.Action`2<A, B>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance void  Invoke(!0 argA,
                                !1 argB)
}
```

[C#]
```
public delegate void Action<A, B>
                     (A argA,
                      B argB
                     );
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs an action using the supplied arguments.

Parameters:

Parameter	Description
argA	The first argument to the action.
argB	The second argument to the action.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for two-argument methods, which perform an action (return void).

4.3 System.Action<A, B, C> delegate

```
[ILAsm]
.class public auto ansi sealed System.Action`3<A, B, C>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance void  Invoke(!0 argA,
                                !1 argB,
                                !2 argC)
}



[C#]
public delegate void Action<A, B, C>
                     (A argA,
                      B argB,
                      C argC
                     );
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs an action using the supplied arguments.

Parameters:

Parameter	Description
argA	The first argument to the action.
argB	The second argument to the action.
argC	The third argument to the action.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for three-argument methods, which perform an action (return void).

4.4 System.Action<A, B, C, D> delegate

```
[ILAsm]
.class public auto ansi sealed System.Action`4<A, B, C, D>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance void  Invoke(!0 argA,
                                !1 argB,
                                !2 argC,
                                !3 argD)
}


[C#]
public delegate void Action<A, B, C, D>
                     (A argA,
                      B argB,
                      C argC,
                      D argD
                     );
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs an action using the supplied arguments.

Parameters:

Parameter	Description
argA	The first argument to the action.
argB	The second argument to the action.
argC	The third argument to the action.
argD	The fourth argument to the action.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for four-argument methods, which perform an action (return void).

4.5 System.Action<A, B, C, D, E> delegate

[ILAsm]
```
.class public auto ansi sealed System.Action`5<A, B, C, D, E>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance void  Invoke(!0 argA,
                                !1 argB,
                                !2 argC,
                                !3 argD,
                                !4 argE)
}
```

[C#]
```
public delegate void Action<A, B, C, D, E>
                     (A argA,
                      B argB,
                      C argC,
                      D argD,
                      E argE
                     );
```

Assembly Info:

- *Name:*

- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs an action using the supplied arguments.

Parameters:

Parameter	Description
argA	The first argument to the action.
argB	The second argument to the action.
argC	The third argument to the action.
argD	The fourth argument to the action.
argE	The fifth argument to the action.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for five-argument methods, which perform an action (return void).

5 System.DelegateCast class

```
[ILAsm]
.class public abstract sealed beforefieldinit System.DelegateCast
       extends System.Object


[C#]
public static class DelegateCast
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Summary

Provides methods for converting between `System.Function<>` and corresponding `System.Predicate<T>` , `System.Converter<T, U>` and `System.Comparison<T>` delegate types.

Inherits From: `System.Object`

Library:

Thread Safety: This type is safe for multithreaded operations.

Description

5.1 DelegateCast.ToFunction<T, Boolean>(System.Predicate<T>) method

[ILAsm]
```
.method public hidebysig static class Function`2<!!0, bool>
          ToFunction<T>(class Predicate`1<!!0> pred)
```

[C#]
```
public static Function<T, bool> ToFunction<T>(Predicate<T> pred)
```

Summary

Constructs a `System.Function<T, bool>` value equivalent to the given `System.Predicate<T>` value.

Parameters

Parameter	Description
pred	The `System.Predicate<T>` to convert.

Return Value

A `System.Function<T, bool>` value, functionally equivalent to the `System.Predicate<T>` argument.

Description

5.2 DelegateCast.ToPredicate<T>(System.Function<T, System.Boolean>) method

[ILAsm]
```
.method public hidebysig static class Predicate`1<!!0>
          ToPredicate<T>(class Function`2<!!0, bool> func)
```

[C#]
```
public static Predicate<T> ToPredicate<T>(Function<T, bool> func)
```

Summary

Constructs a `System.Predicate<T>` value equivalent to the given `System.Function<T, Boolean>` value.

Parameters

Parameter	Description
Func	The `System.Function<T, bool>` to convert.

Return Value

A `System.Predicate<T>` value functionally equivalent to the `System.Function<T, bool>` argument.

Description

5.3 DelegateCast.ToFunction<T, U>(System.Converter<T, U>) method

[ILAsm]
```
.method public hidebysig static class Function`2<!!0, !!1>
          ToFunction<T, U>(class Converter`2<!!0, !!1> conv)
```

[C#]
```
public static Function<T, U> ToFunction<T, U>(Converter<T, U> conv)
```

Summary

Constructs a `System.Function<T, U>` value equivalent to the given `System.Converter<T, U>` value.

Parameters

Parameter	Description
conv	The `System.Converter<T, U>` to convert.

Return Value

A `System.Function<T, U>` value functionally equivalent to the `System.Converter<T, U>` argument.

Description

5.4 DelegateCast.ToConverter<T, U>(System.Function<T, U>) method

[ILAsm]
```
.method public hidebysig static class Converter`2<!!0, !!1>
          ToConverter<T, U>(class Function`2<!!0, !!1> func)
```

[C#]
```
public static Converter<T, U> ToConverter<T, U>(Function<T, U> func)
```

Summary

Constructs a `System.Converter<T, U>` value equivalent to the given `System.Function<T, U>` value.

Parameters

Parameter	Description
func	The `System.Function<T, U>` to convert.

Return Value

A `System.Converter<T, U>` value functionally equivalent to the `System.Function<T, U>` argument.

Description

5.5 DelegateCast.ToFunction<T, T, System.Int32>(System.Comparison<T>) method

[ILAsm]
```
.method public hidebysig static class Function`3<!!0,!!1,int>
          ToFunction<T>(class Comparison`1<!!0> comp)
```

[C#]
```
public static Function<T, T, int> ToFunction<T>(Comparison<T> comp)
```

Summary

Constructs a `System.Function<T, T, Int32>` value equivalent to the given `System.Comparison<T>` value.

Parameters

Parameter	Description
comp	The `System.Comparison<T>` to convert.

Return Value

A `System.Function<T, T, System.Int32>` value functionally equivalent to the `System.Comparison<T>` argument.

Description

5.6 DelegateCast.ToComparison<T>(System.Function<T, T, System.Int32>) method

[ILAsm]
```
.method public hidebysig static class Comparison`1<!!0>
          ToComparison<T>(class Function`3<!!0,!!1,int> func)
```

[C#]
```
public static Comparison<T> ToComparison<T>(Function<T, T, int> func)
```

Summary

Constructs a `System.Comparison<T>` value equivalent to the given `System.Function<T, T, Int32>` value.

Parameters

Parameter	Description
func	The `System.Function<T, T, Int32>` to convert.

Return Value

A `System.Comparison<T>` value functionally equivalent to the `System.Function<T, T, Int32>` argument.

Description

6 System.Either<A, B> Structure

[ILAsm]
```
.class public sequential serializable sealed Either`2<A, B> extends
System.ValueType implements IEquatable`1<valuetype Either`2<A, B>>
```

[C#]
```
public struct Either<A, B> : IEquatable<Either<A, B>>
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*

- CLSCompliantAttribute(true)

Implements:

- System.IEquatable<Either<A, B>>

Summary

Contains a value of type A or a value of type B.

Inherits From: System.ValueType

Library:

Thread Safety: This type is not guaranteed to be safe for multithreaded operations.

Description

The System.Either<A, B> value type represents either a value of a given type A or a value of a given type B. For example, an instance of System.Either<System.Int32, System.String> contains either a System.Int32 value or a System.String value. Instances of this type are immutable.

An instance of System.Either<A, B> has four key properties; IsFirst, IsSecond, First, and Second. IsFirst and IsSecond are used to determine whether the current instance has a value of type A or a value of a given B. They return true or false, and never throw an exception. First and Second return the value of the instance, provided it is of the appropriate type (i.e., First or Second are true respectively); otherwise, they throw an exception.

Calling System.Either<A, B>.IsFirst on an instance that has the default initial value returns true and calling System.Either<A, B>.First returns the default value of type A.

See the methods System.Either<A, B>.MakeFirst and System.Either<A, B>.MakeSecond for how to create new instances initialized to values of type A or type B respectively.

See also System.Nullable<T> (for value types T) and System.Optional<T> (for all types T) that represent either a value of type T or the lack of such a value.

6.1 Either<A, B>.Equals(System.Object) method

[ILAsm]
```
.method public hidebysig virtual bool Equals(object other)
```

[C#]
```
public override bool Equals(object other)
```

Summary

Determines whether the current instance and the specified System.Object represent the same type and value.

Parameters

Parameter	Description
other	The `System.Object` to compare to the current instance.

Return Value

The following table defines the conditions under which the return value is true or false:

Return Value	Conditions
`false`	`other` is null or the current instance and `other` have different types.
`false`	`IsFirst` is `false` and `other.IsFirst` is `true`.
`false`	`IsFirst` is `true` and `other.IsFirst` is `false`.
`First.Equals(other.First)`	`IsFirst` is `true` and `other.IsFirst` is `true`
`Second.Equals(other.Second)`	`IsFirst` is `false` and `other. IsFirst` is `false`

Description

[*Note:* This method overrides `System.Object.Equals`.]

6.2 Either<A, B>.Equals(Either<A, B>) method

[ILAsm]
```
.method public hidebysig virtual bool Equals(valuetype Either`2<A, B> other)
```

[C#]
```
public bool Equals(Either<A, B> other)
```

Summary

Determines whether the current instance and the specified System.Either<A, B> represent the same value.

Parameters

Parameter	Description
other	The `System.Either<A, B>` to compare to the current instance.

Return Value

The following table defines the conditions under which the return value is true or false:

Return Value	Conditions
`false`	`IsFirst` is `false` and `other.IsFirst` is `true`.
`false`	`IsFirst` is `true` and `other.IsFirst` is `false`.
`First.Equals(other.First)`	`IsFirst` is `true` and `other.IsFirst` is `true`
`Second.Equals(other.Second)`	`IsFirst` is `false` and `other. IsFirst` is `false`

Description

[*Note:* This method implicitly implements the `System.IEquatable<Either<A, B>>.Equals` method.]

6.3 Either<A, B>.First property

[ILAsm]
```
.property !0 First()
{
  .get instance !0 get_First()
}
.method public hidebysig specialname instance !0 get_First()
```

[C#]
```
public A First { get; }
```

Summary

Gets the value of type A, if any, of the current instance.

Property Value

The value of type A of the current instance.

Behaviors

If `System.Either<A, B>.IsFirst` is `true`, the instance contains a value of type A, and `System.Either<A, B>.First` returns that value. If `System.Either<A, B>.IsFirst` is `false`, the instance has a value of type B, and an attempt to read `System.Either<A, B>.First` results in an exception.

This property is read-only.

Exceptions

Exception	Condition
`System.InvalidOperationException`	`System.Either<A, B>.IsFirst` is `false`.

6.4 Either<A, B>.GetHashCode() method

[ILAsm]
```
.method public hidebysig virtual int32 GetHashCode()
```

[C#]
```
public override int GetHashCode()
```

Summary

Generates a hash code for the value of the current instance.

Return Value

A `System.Int32` containing the hash code for the value of the current instance is returned.

Description

The algorithm used to generate the hash code is unspecified.

[*Note:* This method overrides `System.Object.GetHashCode`.]

6.5 Either<A, B>.IsFirst property

[ILAsm]
```
.property bool IsFirst()
{
  .get instance bool get_IsFirst()
}
.method public hidebysig specialname instance bool get_IsFirst()
```

[C#]
```
bool IsFirst { get; }
```

Summary

Indicates whether the current instance contains a value of type A.

Property Value

`true` if the current instance was constructed by a call to `MakeFirst` or is the default value of `Either<A, B>`; otherwise, `false`.

Behaviors

If `System.Either<A, B>.IsFirst` is `true`, the instance contains a value of type A, `System.Either<A, B>.First` returns that value, and `System.Either<A, B>.IsSecond` returns `false`. If `System.Either<A, B>.IsFirst` is `false`, the instance contains a value of type B, System.Either<A, B>.IsSecond returns true, and an attempt to read System.Either<A, B>.First results in a System.InvalidOperationException exception.

This property is read-only.

6.6 Either<A, B>.IsSecond property

[ILAsm]
```
.property bool IsSecond()
{
  .get instance bool get_IsSecond()
}
.method public hidebysig specialname instance bool get_IsSecond()
```

[C#]
```
public bool IsSecond { get; }
```

Summary

Indicates whether the current instance contains a value of type B.

Property Value

true if the current instance was constructed by a call to MakeSecond; otherwise, false.

Behaviors

If System.Either<A, B>.IsSecond is true, the instance contains a value of type B, System.Either<A, B>.Second returns that value, and System.Either<A, B>.IsFirst returns false. If System.Either<A, B>.IsSecond is false, the instance contains a value of type A, System.Either<A, B>.IsFirst returns true, and an attempt to read System.Either<A, B>.Second results in a System.InvalidOperationException exception.

This property is read-only.

6.7 Either<A, B>.op_Equality(Either<A, B>, Either<A, B>) method

[ILAsm]
```
.method public hidebysig specialname static
        bool  op_Equality(valuetype Either`2<!0, !1> left,
                          valuetype Either`2<!0, !1> right)
```

[C#]
```
public static bool operator==(Either<A, B> left, Either<A, B> right)
```

Summary

Determines whether the specified values are equal.

Parameters

Parameter	Description
left	The first `Either<A, B>` value to compare.
right	The second `Either<A, B>` value to compare.

Return Value

The following table defines the conditions under which the return value is `true` or `false`:

Return Value	Conditions	
	left.IsFirst	right.IsFirst
`left.Second.Equals(right.Second)`	`false`	`false`
`false`	`false`	`true`
`false`	`true`	`false`
`left.First.Equals(right.First)`	`true`	`true`

Description

6.8 Either<A, B>.op_Inequality(Either<A, B>, Either<A, B>) method

[ILAsm]
```
.method public hidebysig specialname static
        bool op_Inequality(valuetype Either`2<!0, !1> left,
                           valuetype Either`2<!0, !1> right)
```

[C#]
```
public static bool operator!=(Either<A, B> left, Either<A, B> right)
```

Summary

Determines whether the specified values are not equal.

Parameters

Parameter	Description
left	The first `Either<A, B>` value to compare.
right	The second `Either<A, B>` value to compare.

Return Value

The following table defines the conditions under which the return value is `true` or `false`:

Return Value	Conditions	
	left.IsFirst	right.IsFirst
`Not left.Second.Equals(right.Second)`	`false`	`false`
`true`	`false`	`true`
`true`	`true`	`false`
`Not left.First.Equals(right.First)`	`true`	`true`

Description

6.9 Either<A, B>.Second property

[ILAsm]
```
.property !1 Second()
{
  .get instance !1 get_Second()
}
.method public hidebysig specialname instance !1 get_Second()
```

[C#]
```
public B Second { get; }
```

Summary

Gets the value of type B, if any, of the current instance.

Property Value

The value of type B of the current instance.

Behaviors

If `System.Either<A, B>.IsSecond` is `true`, the instance contains a value of type B, and `System.Either<A, B>.Second` returns that value. If `System.Either<A, B>.IsSecond` is `false`, the instance has a value of type A, and an attempt to read `System.Either<A, B>.Second` results in an exception.

This property is read-only.

Exceptions

Exception	Condition
`System.InvalidOperationException`	`System.Either<A, B>.IsSecond` is `false`.

6.10 Either<A, B>.MakeFirst(A aValue) method

[ILAsm]
```
public hidebysig static valuetype Either`<!0, !1>MakeFirst(!0 aValue)
```

[C#]
```
public static Either<A, B> MakeFirst(A aValue)
```

Summary

Constructs and initializes a new instance of `System.Either<A, B>` containing the specified value of type A.

Parameters

Parameter	Description
aValue	The value for the new instance.

Description

Constructs a new instance of `System.Either<A, B>` containing the supplied value of type A. Once this constructor has executed, applying `System.Either<A, B>.IsFirst` to the new instance returns `true`and applying `System.Either<A, B>.IsSecond` to the new instance returns `false`.

6.11 Either<A, B>.MakeSecond(B bValue) method

[ILAsm]
```
public hidebysig static valuetype Either`<!0, !1>MakeSecond(!1 bValue)
```

[C#]
```
public static Either<A, B> MakeSecond(B bValue)
```

Summary

Constructs and initializes a new instance of `System.Either<A, B>` containing the specified value of type B.

Parameters

Parameter	Description
bValue	The value for the new instance.

Description

Constructs a new instance of `System.Either<A, B>` containing the supplied value of type B. Once this constructor has executed, applying `System.Either<A, B>.IsSecond` to the new instance returns `true`.

6.12 Either<A, B>.ToString() method

[ILAsm]
```
.method public hidebysig virtual string ToString()
```

[C#]
```
public override string ToString()
```

Summary

Returns a `System.String` representation of the value of the current instance.

Return Value

If `System.Either<A, B>.IsFirst` is `true`, `System.Either<A, B>.First.ToString()` is returned; otherwise, `System.Either<A, B>.Second.ToString()` is returned.

Description

[*Note:* This method overrides `System.Object.ToString`.]

6.13 Either<A, B>(A) constructor

[ILAsm]
```
public rtspecialname specialname instance void .ctor(A aValue)
```

[C#]
```
public Either(A aValue)
```

Attributes

- CLSCompliantAttribute(false)

Summary

Constructs and initializes a new instance of `System.Either<A, B>` which contains the given value of type A.

Description

Constructs a new instance of `System.Either<A, B>`. Once this constructor has executed, calling `System.Either<A, B>.IsFirst` on the new instance returns `true` and calling `System.Either<A, B>.First` returns the given value of type A.

[*Note:* If the type A is the same as the type B then this method unifies with the constructor which accepts a value of type B; therefore this method is not CLS-compliant and is marked as such.]

[*Note:* The construction implemented by this method corresponds exactly to calling the `System.Either<A, B>.MakeFirst` method.]

6.14 Either<A, B>(B) constructor

```
[ILAsm]
public rtspecialname specialname instance void .ctor(B bValue)


[C#]
public Either(B bValue)
```

Attributes

- CLSCompliantAttribute(false)

Summary

Constructs and initializes a new instance of `System.Either<A, B>` which contains the given value of type B.

Description

Constructs a new instance of `System.Either<A, B>`. Once this constructor has executed, calling `System.Either<A, B>.IsSecond` on the new instance returns `true` and calling `System.Either<A, B>.Second` returns the given value of type B.

[*Note:* If the type A is the same as the type B then this method unifies with the constructor which accepts a value of type A; therefore this method is not CLS-compliant and is marked as such.]

[*Note:* The construction implemented by this method corresponds exactly to calling the `System.Either<A, B>.MakeSecond` method.]

6.15 A Either<A, B>.op_Explicit(System.Either<A, B>) method

```
[ILAsm]
.method public hidebysig static specialname !0 op_Explicit(valuetype Either`2<!0,
!1> abValue)


[C#]
public static explicit operator A(Either<A, B> abValue)
```

Attributes

- CLSCompliantAttribute(false)

Summary

Perform an explicit conversion of a `System.Either<A, B>` value to type A.

Parameters

Parameter	Description
abValue	The `System.Either<A, B>` value to convert to type A.

Return Value

The value of type A, if any, of the specified value. Otherwise, an exception is thrown.

Description

An explicit conversion from a value of type `System.Either<A, B>` to a value of type A. If `System.Either<A, B>.IsFirst`. is `true` the value is returned, otherwise an exception is thrown.

[*Note:* If the type A is the same as the type B then this method unifies with the `op_Explicit` which returns a value of type B, therefore this method is not CLS-compliant and is marked as such.]

[*Note:* The conversion implemented by this method corresponds exactly to obtaining the value of the `System.Either<A, B>.First` property.]

Exceptions

Exception	Condition
`System.InvalidOperationException`	**`System.Either<A, B>.IsFirst`** is false.

6.16 B Either<A, B>.op_Explicit(System.Either<A, B>) method

```
[ILAsm]
.method public hidebysig static specialname !1 op_Explicit(valuetype Either`2<!0,
!1> abValue)


[C#]
public static explicit operator B(Either<A, B> abValue)
```

Attributes

- CLSCompliantAttribute(false)

Summary

Perform an explicit conversion of a `System.Either<A, B>` value to type B.

Parameters

Parameter	Description
abValue	The `System.Either<A, B>` value to convert to type B.

Return Value

The value of type B, if any, of the specified value. Otherwise, an exception is thrown.

Description

An explicit conversion from a value of type `System.Either<A, B>` to a value of type B. If `System.Either<A, B>.IsFirst` is `true` the value is returned otherwise an exception is thrown.

[*Note:* If the type A is the same as the type B then this method unifies with the `op_Explicit` which returns a value of type A, therefore this method is not CLS-compliant and is marked as such.]

[*Note:* The conversion implemented by this method corresponds exactly to obtaining the value of the `System.Either<A, B>.Second` property.]

Exceptions

Exception	Condition
`System.InvalidOperationException`	`System.Either<A, B>.IsSecond` is `false`.

6.17 Either<A, B>.op_Implicit(A) method

[ILAsm]
```
.method public hidebysig static specialname valuetype Either`2<!0, !1>
op_Implicit(!0 aValue)
```

[C#]
```
public static implicit operator Either<A, B>(A aValue)
```

Attributes

- CLSCompliantAttribute(false)

Summary

Perform an implicit conversion of a value of type A to a value of type `System.Either<A, B>`.

Parameters

Parameter	Description
aValue	The `A` value to convert to `System.Either<A, B>`.

Return Value

A `System.Either<A, B>` with the specified value of type A.

Description

Perform a conversion from *aValue* to a value of type `System.Either<A, B>`. Calling `System.Either<A, B>.IsFirst` on the returned value returns `true`, and `System.Either<A, B>.First` returns the value.

[*Note:* If the type A is the same as the type B then this method unifies with the `op_Implicit` which takes a value of type B, therefore this method is not CLS-compliant and is marked as such.]

[*Note:* The conversion implemented by this method corresponds exactly to invoking the `System.Either<A, B>.MakeFirst(A)` method.]

6.18 Either<A, B>.op_Implicit(B) method

[ILAsm]
```
.method public hidebysig static specialname valuetype Either`2<!0, !1>
op_Implicit(!1 bValue)
```

[C#]
```
public static implicit operator Either<A, B>(B bValue)
```

Attributes

- CLSCompliantAttribute(false)

Summary

Perform an implicit conversion of a value of type B to a value of type `System.Either<A, B>`.

Parameters

Parameter	Description
bValue	The `B` value to convert to `System.Either<A, B>`.

Return Value

A `System.Either<A, B>` with the specified value of type B.

Description

Perform a conversion from *bValue* to a value of type `System.Either<A, B>`. Calling `System.Either<A, B>.IsSecond` on the returned value returns `true`, and `System.Either<A, B>.Second` returns the value.

[*Note:* If the type A is the same as the type B then this method unifies with the `op_Implicit` which takes a value of type A, therefore this method is not CLS-compliant and is marked as such.]

[*Note:* The conversion implemented by this method corresponds exactly to invoking the `System.Either<A, B>.MakeSecond(B)` method.]

7 Function Delegates

7.1 System.Function<A > delegate

[ILAsm]
```
.class public auto ansi sealed System.Function`1<A>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance !0 Invoke()
}
```

[C#]
```
public delegate A Function<A>();
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - o CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that returns a value.

Parameters:

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for zero-argument methods, which perform a function (return a value).

7.2 System.Function<A, B> delegate

```
[ILAsm]
.class public auto ansi sealed System.Function`2<A, B>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance !1 Invoke(!0 argA)
}


[C#]
public delegate B Function<A, B>
                  (A argA
                  );
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs a function using the supplied argument.

Parameters:

Parameter	Description
argA	The argument to the function.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for one-argument methods, which perform a function (return a value).

7.3 System.Function<A, B, C> delegate

[ILAsm]
```
.class public auto ansi sealed System.Function`3<A, B, C>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance !2 Invoke(!0 argA,
                             !1 argB)
}
```

[C#]
```
public delegate C Function<A, B, C>
                  (A argA,
                   B argB
                  );
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs a function using the supplied arguments.

Parameters:

Parameter	Description
argA	The first argument to the function.
argB	The second argument to the function.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for two-argument methods, which perform a function (return a value).

7.4 System.Function<A, B, C, D> delegate

[ILAsm]
```
.class public auto ansi sealed System.Function`4<A, B, C, D>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance !3 Invoke(!0 argA,
                             !1 argB,
                             !2 argC)
}
```

[C#]
```
public delegate D Function<A, B, C, D>
                 (A argA,
                  B argB,
                  C argC
                 );
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs a function using the supplied arguments.

Parameters:

Parameter	Description
argA	The first argument to the function.
argB	The second argument to the function.
argC	The third argument to the function.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for three-argument methods, which perform a function (return a value).

7.5 System.Function<A, B, C, D, E> delegate

```
[ILAsm]
.class public auto ansi sealed System.Function`5<A, B, C, D, E>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance !4 Invoke(!0 argA,
                             !1 argB,
                             !2 argC,
                             !3 argD)
}



[C#]
public delegate E Function<A, B, C, D, E>
                  (A argA,
                   B argB,
                   C argC,
                   D argD
                  );
```

Assembly Info:

- *Name:*

- *Public Key:*

- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs a function using the supplied arguments.

Parameters:

Parameter	Description
argA	The first argument to the function.
argB	The second argument to the function.
argC	The third argument to the function.
argD	The fourth argument to the function.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for four-argument methods, which perform a function (return a value).

7.6 System.Function<A, B, C, D, E, F> delegate

[ILAsm]
```
.class public auto ansi sealed System.Function`6<A, B, C, D, E, F>
       extends System.Delegate
{
  .method public hidebysig newslot virtual
          instance !5 Invoke(!0 argA,
                             !1 argB,
                             !2 argC,
                             !3 argD,
                             !4 argE)
}
```

[C#]
```
public delegate F Function<A, B, C, D, E, F>
                  (A argA,
                   B argB,
                        C argC,
                        D argD,
                        E argE
                       );
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.ICloneable`

Summary:

Represents a method that performs a function using the supplied arguments.

Parameters:

Parameter	**Description**
argA	The first argument to the function.
argB	The second argument to the function.
argC	The third argument to the function.
argD	The fourth argument to the function.
argE	The fifth argument to the function.

Inherits From: `System.Delegate`

Library: BCL

Description

A generic delegate type for five-argument methods, which perform a function (return a value).

8 System.Optional<T> structure

```
[ILAsm]
.class public sequential serializable sealed
       Optional`1<T>
       extends System.ValueType
       implements System.IComparable,
                  System.IComparable`1<valuetype Optional`1<!0>>,
                  System.IEquatable`1<valuetype Optional`1<!0>>,
                  System.INullableValue



[C#]
public struct Optional<T> : IComparable,
                            IComparable<Optional<T>>,
                            IEquatable<Optional<T>>,
                            INullableValue
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- `System.IComparable`
- `System.IComparable<System.Optional<T>>`
- `System.IEquatable<System.Optional<T>>`
- `System.INullableValue`

Summary

Represents an optional value. Similar to `System.Nullable<T>` except that the generic type parameter is not constrained in any way.

Inherits From: `System.ValueType`

Library:

Thread Safety: This type is not guaranteed to be safe for multithreaded operations.

Description

The `System.Optional<T>` value type represents a value of a given type T or the lack of such a value. For example an instance of `System.Optional<System.Int32>` contains a `System.Int32` value or an indication that no value is available. Instances of this type are immutable.

An instance of `System.Optional<T>` has two properties, `HasValue` and `Value`. Property `HasValue` is used to determine whether the current instance has a value of type T. It returns `true` or `false`, and never throws an exception. Property `Value` returns the value of the instance, provided it has one (i.e., `HasValue` is `true`); otherwise, it throws an exception.

In addition to the above properties, there is a pair of methods, both overloads of `GetValueOrDefault`. The version taking no arguments returns the instance's value of type T, if it has one; otherwise, it returns the default value of type T. The version taking a value argument of type T returns the instance's value of type T, if it has one; otherwise, it returns the value argument passed to it.

Applying `System.Optional<T>.HasValue` to an instance that has the default initial value returns `false`.

`System.Optional<T>` is similar to `System.Nullable<T>` but is not constrained to containing non-nullable value types. Conversion methods, `System.Optional<T>.FromNullable` and `System.Optional<T>.ToNullable`, are provided between the types.

See also `System.Either<A, B>` that represents either a value of type A or a value of type B.

8.1 Optional<T>(T) constructor

```
[ILAsm]
public rtspecialname specialname instance void .ctor(!0 value)


[C#]
public Optional(T value)
```

Summary

Constructs and initializes a new instance of `System.Optional<T>` containing the specified value.

Parameters

Parameter	Description
value	The value for the new instance.

Description

Constructs a new instance of `System.Optional<T>` containing the supplied value of type T. Once this constructor has executed, calling `System.Optional<T>.HasValue` to the new instance returns `true` and `System.Optional<T>.Value` returns the value.

8.2 Optional<T>.CompareTo(System.Object) method

[ILAsm]
```
.method private final hidebysig virtual
        int32 CompareTo(object other)
```

[C#]
```
int CompareTo(object other)
```

Summary

Returns the sort order of the current instance compared to the specified `System.Object`.

Parameters

Parameter	Description
other	The `System.Object` to compare to the current instance.

Return Value

The return value is a negative number, zero, or a positive number reflecting the sort order of the current instance as compared to *other*. For non-zero values, the exact value returned by this method is unspecified. The following table defines the conditions under which the return value is a negative number, zero, or a positive number.

Return Value	Description
Any negative number	Current instance < *other*.
Zero	Current instance == *other*.
Any positive number	Current instance > *other*, or *other* is a null reference.

If *other* is `null`, the returned value is 0 if `System.Optional<T>.HasValue` is false, or 1 otherwise. If *other* is non-null but not an instance of `System.Optional<T>`, an exception is thrown. Otherwise, the result is computed from the `System.Optional<T>.HasValue` and `System.Optional<T>.Value` properties, as follows:

Return Value	Conditions	
	HasValue	other.HasValue
0	`false`	`false` **or** *other* is null
Some negative number	`false`	`true`
Some positive number	`true`	`false` **or** *other* is null
`Value.CompareTo(other.Value)`	`true`	`true`

Description

[*Note:* This method is implemented to support the `System.IComparable` interface.]

Exceptions

Exception	Condition
`System.ArgumentException`	*other* is not null and not an instance of `Optional<T>`. or Type `T` doesn't implement `System.IComparable`.

8.3 Optional<T>.CompareTo(Optional<T>) method

[ILAsm]
```
.method public hidebysig virtual final
        int32 CompareTo(valuetype Optional`1<!0> other)
```

[C#]
```
public int CompareTo(Optional<T> other)
```

Summary

Returns the sort order of the current instance compared to the specified object.

Parameters

Parameter	Description
other	The `System.Optional<T>` to compare to the current instance.

Return Value

A value that reflects the sort order of the current instance as compared to *other*. For non-zero values, the exact value returned by this method is unspecified. The following table defines the conditions under which the returned value is a negative number, zero, or a positive number.

Returned Value	Description
A negative value	The current instance is < *other*.
Zero	The current instance is == *other*.
A positive value	The current instance is > than *other*.

If either of the current instance or *other* is not an instance of `System.Optional<T>`, an exception is thrown. Otherwise, the result is computed from the `System.Optional<T>.HasValue` and `System.Optional<T>.Value` properties, as follows:

Return Value	Conditions	
	HasValue	other.HasValue
0	`false`	`false`
Some negative number	`false`	`true`
Some positive number	`true`	`false`
`Value.CompareTo(other.Value)`	`true`	`true`

Description

If `T` implements `System.IComparable<T>`, that interface's method is used. Otherwise, if `T` implements `System.IComparable`, that interface's method is used. Otherwise, an exception is thrown.

[*Note:* This method implicitly implements the `System.IComparable<Optional<T>>.CompareTo` method.]

Exceptions

Exception	Condition
`System.ArgumentException`	Type `T` doesn't implement `System.IComparable<T>` or `System.IComparable`.

8.4 Optional<T>.Equals(System.Object) method

[ILAsm]
```
.method public hidebysig virtual bool Equals(object other)
```

[C#]
```
public override bool Equals(object other)
```

Summary

Determines whether the current instance and the specified `System.Object` represent the same type and value.

Parameters

Parameter	Description
other	The `System.Object` to compare to the current instance.

Return Value

The following table defines the conditions under which the return value is `true` or `false`:

Return Value	Conditions	
	HasValue	other.HasValue
`false`	The current instance and *other* have different types.	
`true`	`false`	`false` **or** *other* is null
`false`	`false`	`true`
`false`	`true`	`false` **or** *other* is null
`Value.Equals(other.Value)`	`true`	`true`

Description

[*Note:* This method overrides `System.Object.Equals`.]

8.5 Optional<T>.Equals(Optional<T>) method

[ILAsm]
```
.method public hidebysig virtual final
        bool Equals(valuetype Optional`1<!0> other)
```

[C#]
```
public bool Equals(Optional<T> other)
```

Summary

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

Parameters

Parameter	Description
other	The `Optional<T>` to compare to the current instance.

Return Value

The following table defines the conditions under which the return value is `true` or `false`:

Return Value	Conditions	
	HasValue	other.HasValue
`true`	`false`	`false`
`false`	`false`	`true`
`false`	`true`	`false`
`Value.Equals(other.Value)`	`true`	`true`

Description

[*Note:* This method implicitly implements the `System.IEquatable<Optional<T>>.Equals` method.]

8.6 Optional<T>.FromNullable<U>(Nullable<U>) method

[ILAsm]
```
.method public hidebysig static valuetype Optional`1<!!0>
        FromNullable<valuetype U>
          (valuetype Nullable`1<!!0> value)
```

[C#]
```
public static Optional<U> FromNullable(Nullable<U> value)
       where U : struct
```

Summary

Perform a conversion of a `System.Nullable<U>` value to `System.Optional<U>`.

Parameters

Parameter	Description
value	The `System.Nullable<U>` value to convert to `System.Optional<U>`.

Return Value

A `System.Optional<U>` with the specified value.

Description

Perform a conversion of a `System.Nullable<U>` value to `System.Optional<U>`. The type `U` is constrained to be a value type to satisfy the requirements of `System.Nullable<U>`.

8.7 Optional<T>.FromOptional(System.Optional<T>) method

[ILAsm]
```
.method public hidebysig static !0 FromOptional(valuetype System.Optional`1<!0>
value)
```

[C#]
```
public static explicit operator T(Optional<T> value)
```

Summary

Perform a conversion of a `System.Optional<T>` value to type T.

Parameters

Parameter	Description
value	The `System.Optional<T>` value to convert to type T.

Return Value

The value, if any, of the specified optional value. Otherwise, an exception is thrown.

Description

[*Note:* This method is provided for CLS compliance and is equivalent to `System.Optional<T>.op_explicit(T)`.]

[*Note:* The conversion implemented by this method corresponds exactly to obtaining the value of the `System.Optional<T>.Value` property.]

Exceptions

Exception	Condition
`System.InvalidOperationException`	`System.Optional<T>.HasValue` is false.

8.8 Optional<T>.GetHashCode() method

[ILAsm]
```
.method public hidebysig virtual int32 GetHashCode()
```

[C#]
```
public override int GetHashCode()
```

Summary

Generates a hash code for the value of the current instance.

Return Value

If `System.Optional<T>.HasValue` is `true`, a `System.Int32` containing the hash code for the value of the current instance is returned; otherwise, 0 is returned.

Description

The algorithm used to generate the hash code when `System.Optional<T>.HasValue` is `true` is unspecified.

[*Note:* This method overrides `System.Object.GetHashCode`.]

8.9 Optional<T>.GetValueOrDefault() method

[ILAsm]
```
.method public hidebysig !0 GetValueOrDefault()
```

[C#]
```
public T GetValueOrDefault()
```

Summary

Returns the value of the current instance, or if it has none, returns the default value for the type T.

Return Value

A value of type T, which is either the value of the current instance, or if it has none, the default value for the type T.

Description

[*Note:* Another method, `System.Optional<T>.GetValueOrDefault(T),` allows a value other than the default value of type T to be returned if the current instance has no value.]

8.10 Optional<T>.GetValueOrDefault(T) method

[ILAsm]
```
.method public hidebysig !0 GetValueOrDefault(!0 alternateDefaultValue)
```

[C#]
```
public T GetValueOrDefault(T alternateDefaultValue)
```

Summary

Returns the value of the current instance, or if it has none, returns *alternateDefaultValue*.

Parameters

Parameter	**Description**
alternateDefaultValue	The value to be returned if the current instance has no value.

Return Value

A value of type T, which is either the value of the current instance, or if it has none, the value *alternateDefaultValue*.

Description

[*Note:* `System.Optional<T>.GetValueOrDefault()` allows the default value for type T to be returned if the current instance has no value.]

8.11 Optional<T>.HasValue property

```
[ILAsm]
.property bool HasValue()
{
  .get instance bool get_HasValue()
}
.method public hidebysig specialname instance bool get_HasValue()


[C#]
public bool HasValue { get; }
```

Summary

Indicates whether the current instance contains a value.

Property Value

`true` if the current instance contains a value; otherwise, `false`.

Behaviors

If `System.Optional<T>.HasValue` is `true`, the instance contains a value of type T, and `System.Optional<T>.Value` returns that value. If `System.Optional<T>.HasValue` is `false`, the instance contains no value of type T, and an attempt to read `System.Optional<T>.Value` results in a `System.InvalidOperationException` exception. A call to `System.Optional<T>.Value.GetValueOrDefault` returns the default value.

This property is read-only.

[*Note:* This method implicitly implements the `Optional<T>.INullableValue.HasValue` property.]

8.12 Optional<T>.op_Equality(Optional<T>, Optional<T>) method

[ILAsm]
```
.method public hidebysig specialname static
        bool  op_Equality(valuetype Optional`1<!0> left,
                          valuetype Optional`1<!0> right)
```

[C#]
```
static bool operator==(Optional<T> left, Optional<T> right)
```

Summary

Determines whether the specified values are equal.

Parameters

Parameter	Description
left	The first `Optional<T>` value to compare.
right	The second `Optional<T>` value to compare.

Return Value

The following table defines the conditions under which the return value is `true` or `false`:

Return Value	**Conditions**	
	left.HasValue	**right.HasValue**
`true`	`false`	`false`
`false`	`false`	`true`
`false`	`true`	`false`
`left.Value.Equals(right.Value)`	`true`	`true`

Description

8.13 Optional<T>.op_Explicit(System.Optional<T>) method

[ILAsm]
```
.method public hidebysig static specialname !0 op_Explicit(valuetype
Optional`1<!0> value)
```

[C#]
```
public static explicit operator T(Optional<T> value)
```

Summary

Perform an explicit conversion of a `System.Optional<T>` value to type T.

Parameters

Parameter	Description
value	The `System.Optional<T>` value to convert to type T.

Return Value

The value, if any, of the specified optional value. Otherwise, an exception is thrown.

Description

[*Note:* See the `System.Optional<T>.FromOptional(Optional<T>)` method for the CLS twin of this operator.]

[*Note:* The conversion implemented by this method corresponds exactly to obtaining the value of the `System.Optional<T>.Value` property.]

Exceptions

Exception	Condition
`System.InvalidOperationException`	`System.Optional<T>.HasValue` is false.

8.14 Optional<T>.op_Implicit(T) method

[ILAsm]
```
.method public hidebysig static specialname valuetype Optional`1<!0>
op_Implicit(!0 value)
```

[C#]
```
public static implicit operator Optional<T>(T value)
```

Summary

Perform an implicit conversion of a `T` value to `System.Optional<T>`.

Parameters

Parameter	Description
value	The `T` value to convert to `System.Optional<T>`.

Return Value

A `System.Optional<T>` with the specified value.

Description

[*Note:* See the `System.Optional<T>.ToOptional(T)` method for the CLS twin of this operator.]

[*Note:* The conversion implemented by this method corresponds exactly to invoking the `System.Optional<T>(T)` constructor.]

8.15 Optional<T>.op_Inequality(Optional<T>, Optional<T>) method

```
[ILAsm]
.method public hidebysig specialname static
        bool op_Inequality(valuetype Optional`1<!0> left,
                           valuetype Optional`1<!0> right)


[C#]
static bool operator!=(Optional<T> left, Optional<T> right)
```

Summary

Determines whether the specified values are not equal.

Parameters

Parameter	Description
left	The first `Optional<T>` value to compare.
right	The second `Optional<T>` value to compare.

Return Value

The following table defines the conditions under which the return value is `true` or `false`:

Return Value	Conditions	
	left.HasValue	right.HasValue
`false`	`false`	`false`
`true`	`false`	`true`
`true`	`true`	`false`
`Not left.Value.Equals(right.Value)`	`true`	`true`

Description

8.16 Optional<T>.ToNullable<U>(Optional<U>) method

[ILAsm]
```
.method public hidebysig static valuetype Nullable`1<!!0>
        ToNullable<.ctor (System.ValueType) U>
          (valuetype Optional`1<!!0> value)
```

[C#]
```
public static Nullable<U> ToNullable(Optional<U> value)
       where U : struct
```

Summary

Perform a conversion of a `System.Optional<U>` value to `System.Nullable<U>`.

Parameters

Parameter	Description
value	The `System.Optional<U>` value to convert to `System.Nullable<U>`.

Return Value

A `System.Nullable<U>` with the specified value.

Description

Perform a conversion of a `System.Optional<U>` value to `System.Nullable<U>`. The type `U` is constrained to be a value type to satisfy the requirements of `System.Nullable<U>`.

8.17 Optional<T>.ToOptional(T) method

[ILAsm]
```
.method public hidebysig static valuetype Optional`1<!0> ToOptional(!0 value)
```

[C#]
```
public static Optional<T> ToOptional(T value)
```

Summary

Perform a conversion of a `T` value to `System.Optional<T>`.

Parameters

Parameter	Description
value	The `T` value to convert to `System.Optional<T>`.

Return Value

A `System.Optional<T>` with the specified value.

Description

[*Note:* This method is provided for CLS compliance and is equivalent to `System.Optional<T>.op_Implicit(T)`.]

[*Note:* The conversion implemented by this method corresponds exactly to invoking the `System.Optional<T>(T)` constructor.]

8.18 Optional<T>.ToString() method

[ILAsm]
```
.method public hidebysig virtual string ToString()
```

[C#]
```
public override string ToString()
```

Summary

Returns a `System.String` representation of the value of the current instance.

Return Value

If `System.Optional<T>.HasValue` is `true`, `System.Optional<T>.Value.ToString()` is returned; otherwise, `System.String.Empty` is returned.

Description

[*Note:* This method overrides `System.Object.ToString`.]

8.19 Optional<T>.Value property

[ILAsm]
```
.property !0 Value()
{
  .get instance !0 get_Value()
}
.method public hidebysig specialname instance !0 get_Value()
```

[C#]
```
public T Value { get; }
```

Summary

Gets the value, if any, of the current instance.

Property Value

The value of the current instance.

Behaviors

If `System.Optional<T>.HasValue` is `true`, the instance contains a value of type T, and `System.Optional<T>.Value` returns that value. If `System.Optional<T>.HasValue` is `false`, the instance has no value of type T, and an attempt to read `System.Optional<T>.Value` results in an exception.

This property is read-only.

Exceptions

Exception	Condition
`System.InvalidOperationException`	`System.Optional<T>.HasValue` is `false`.

8.20 Optional<T>.INullableValue.Value property

[ILAsm]
```
.property object INullableValue.Value()
{
  .get instance object INullableValue.get_Value()
}
.method private hidebysig specialname newslot virtual final instance object
INullableValue get_Value()
```

[C#]
```
object INullableValue.Value { get; }
```

Summary

Gets the value, if any, of the current instance.

Property Value

The value of the current instance.

Description

[*Note:* This method is implemented to support the `System.INullableValue` interface.]

Behaviors

If `System.Optional<T>.HasValue` is `true`, the instance contains a value of type T, and `System.Optional<T>.Value` returns that value. If `System.Optional<T>.HasValue` is `false`, the instance has no value of type T, and an attempt to read `System.Optional<T>.Value` results in an exception.

This property is read-only.

Exceptions

Exception	Condition
`System.InvalidOperationException`	`System.Optional<T>.HasValue` is `false`.

9 Tuple structures System.Tuple<A, B> … System.Tuple<A, B, …, I, J>

[ILAsm]
```
.class public sequential serializable sealed Tuple`2<A, B>
       extends System.ValueType
       implements System.IEquatable`1<valuetype Tuple`2<A, B>>
.class public sequential serializable sealed Tuple`3<A, B, C>
       extends System.ValueType
       implements System.IEquatable`1<valuetype Tuple`3<A, B, C>>
…
.class public sequential serializable sealed
       Tuple`10<A, B, C, D, E, F, G, H, I, J>
       extends System.ValueType
       implements System.IEquatable`1<valuetype Tuple`10<A, B, C, D, E, F, G, H,
I, J>>
```

[C#]
```
public struct Tuple<A, B> : IEquatable<Tuple<A, B>>
public struct Tuple<A, B, C> : IEquatable<Tuple<A, B, C>>
…
public struct Tuple<A, B, C, D, E, F, G, H, I, J>
              : IEquatable<Tuple<A, B, C, D, E, F, G, H, I, J>>
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

Summary

A family of tuple types of sizes 2 though 10.

In this description `System.Tuple<A, …>` is used to represent any one of the types in this family.

Inherits From: `System.ValueType`

Library:

Thread Safety: This type is not guaranteed to be safe for multithreaded operations.

Description

A value of the `System.Tuple<A, …>` value type is a tuple of elements.

An *n*-ary tuple is a collection of *n* elements. Each element has a *position* within the tuple, and the elements may be of different types. The `System.Tuple<A, …>` family are non-abstract types with public fields for each element of the tuple.

Methods are provided to construct a tuple initializing all the elements, to access and modify individual elements, to compare tuples for equality, and to convert tuples to a string representation.

The conventional notation for a tuple is a comma-separated list of values in parentheses, and this is the form used for the string representation.

Tuples do not have an ordering defined over them, only equality; however the position of each element is relevant. For example the tuples `(42, "the ultimate answer ")` and `("the ultimate answer ", 42)` are not equal; the first one has type `System.Tuple<Int32, String>` and the second one `System.Tuple<String.Int32>`.

9.1 Tuple<A, …>() constructors

```
[ILAsm]
public rtspecialname specialname instance void .ctor(!0, !1)
public rtspecialname specialname instance void .ctor(!0, !1, !2)
…
public rtspecialname specialname instance void .ctor(!0, !1, !2, !3, !4, !5, !6,
!7, !8, !9)


[C#]
public Tuple(A, B)
public Tuple(A, B, C)
…


public Tuple(A, B, C, D, E, F, G, H, I, J)
```

Summary

Constructs and initializes a new instance of `System.Tuple<A, …>` which contains the supplied values.

Description

9.2 Tuple<A, …>.Equals(System.Object) method

```
[ILAsm]
.method public hidebysig virtual instance bool Equals(object other)


[C#]
public override bool Equals(object other)
```

Summary

Determines whether the current instance and the specified System.Object represent the same type and value.

Parameters

Parameter	Description
other	The `System.Object` to compare to the current instance.

Return Value

If *other* is a non-null, is an instance of `System.Tuple<A, …>` and its fields, in order, are equal to the fields of the current instance, this method returns `true`, otherwise it returns `false.`

Description

Two tuples are equal if every pair of corresponding fields is equal. Fields are compared using the `Equals` method of the field type, e.g., `ItemA.Equals(other.ItemA)`.

[*Note:* This method overrides `System.Object.Equals`.]

9.3 Tuple<A, …>.Equals(Tuple<A, …>) method

```
[ILAsm]
.method public hidebysig newslot virtual final instance bool
Equals(valuetype Tuple`1<A> other)
.method public hidebysig newslot virtual final instance bool
Equals(valuetype Tuple`2<A, B> other)
…
.method public hidebysig newslot virtual final instance bool
Equals(valuetype Tuple`10<A, B, C, D, E, F, G, H, I, J> other)


[C#]
public bool Equals(Tuple<A> other)
public bool Equals(Tuple<A, B> other)
…
public bool Equals(Tuple<A, B, C, D, E, F, G, H, I, J> other)
```

Summary

Determines whether the current instance and the specified `System.Tuple<A, …>` represent the same value.

Parameters

Parameter	Description
other	The `System.Tuple<A, …>` to compare to the current instance.

Return Value

If *other* is equal to the current instance this method returns `true`, otherwise it returns `false.`

Description

Two tuples are equal if every pair of corresponding fields is equal. Fields are compared using the `Equals` method of the field type, e.g. `ItemA.Equals(other.ItemA)`.

[*Note:* This method implicitly implements the `System.IEquatable<Tuple<A, …>>.Equals` method.]

9.4 Tuple<A, …>.ItemA, Tuple<A, B, …>.ItemB, … Tuple<A, B, …, I, J>.ItemJ field

[ILAsm]
```
.field public !0 ItemA
.field public !1 ItemB
…
.field public !9 ItemJ
```

[C#]
```
public A ItemA;
public B ItemB;
…
public J ItemJ;
```

Summary

The value of the given element of the tuple.

Description

Tuples are non-abstract types with public fields. Each field may be read or written.

9.5 Tuple<A, …>.GetHashCode() method

[ILAsm]
```
.method public hidebysig virtual int32 GetHashCode()
```

[C#]
```
public override int GetHashCode()
```

Summary

Generates a hash code for the value of the current instance.

Return Value

A `System.Int32` containing the hash code for the value of the current instance is returned.

Description

The algorithm used to generate the hash code is unspecified.

[*Note:* This method overrides `System.Object.GetHashCode`.]

9.6 Tuple<A, …>.op_Equality(Tuple<A, …>, Tuple<A, …>) method

[ILAsm]
```
.method public hidebysig specialname static
        bool  op_Equality(valuetype Tuple`2<!0, !1> left,
                          valuetype Tuple`2<!0, !1> right)
.method public hidebysig specialname static
        bool  op_Equality(valuetype Tuple`3<!0, !1, !2> left,
                          valuetype Tuple`3<!0, !1, !2> right)
…

.method public hidebysig specialname static
     bool op_Equality
     (valuetype Tuple`10<!0, !1, !2, !3, !4, !5, !6, !7, !8, !9> left,
      valuetype Tuple`10<!0, !1, !2, !3, !4, !5, !6, !7, !8, !9> right
     )
```

[C#]
```
public static bool operator==(Tuple<A, B> left, Tuple<A, B> right)
public static bool operator==(Tuple<A, B, C> left, Tuple<A, B, C> right)
…
public static bool operator==(Tuple<A, B, C, D, E, F, G, H, I, J> left,
                              Tuple<A, B, C, D, E, F, G, H, I, J> right
                             )
```

Summary

Determines whether the specified values are equal.

Parameters

Parameter	Description
left	The first `Tuple<A, …>` value to compare.

right	The second `Tuple<A, …>` value to compare.

Return Value

`true` if the tuples are equal, `false` otherwise.

Description

Two tuples are equal if every pair of corresponding fields is equal. Fields are compared using the `Equals` method of the field type, e.g. `ItemA.Equals(other.ItemA)`.

9.7 Tuple<A, B>.op_Implicit(KeyValuePair<A, B>) method

[ILAsm]
```
.method public hidebysig static specialname valuetype Tuple`2<!0, !1>
op_Implicit(KeyValuePair`2<!0, !1> arg)
```

[C#]
```
public static implicit operator Tuple<A, B>(KeyValuePair<A, B> arg)
```

Summary

Perform an implicit conversion from a value of type `System.Collections.Generic.KeyValuePair<A, B>` to a value of type `System.Tuple<A, B>`.

Parameters

Parameter	Description
arg	The `System.Collections.Generic.KeyValuePair<A, B>` value to convert to `System.Tuple<A, B>`.

Return Value

A `System.Tuple<A, B>` with *ItemA* having the value of *arg.Key* and *ItemB* having the value of *arg.Value*.

Description

This operator, only available on `System.Tuple<A, B>` and not other members of the tuple family, converts a value of type `System.Collections.Generic.KeyValuePair<A, B>` to a value of type `System.Tuple<A, B>`.

9.8 Tuple<A, B>.op_Implicit(Tuple<A, B>) method

[ILAsm]
```
.method public hidebysig static specialname valuetype KeyValuePair`2<!0, !1>
op_Implicit(Tuple`2<!0, !1> arg)
```

[C#]
```
public static implicit operator KeyValuePair<A, B>(Tuple<A, B> arg)
```

Summary

Perform an implicit conversion of a value of type `System.Tuple<A, B>` to a value of type `System.Collections.Generic.KeyValuePair<A, B>`.

Parameters

Parameter	Description
arg	The `System.Tuple<A, B>` value to convert to `System.Collections.Generic.KeyValuePair<A, B>`.

Return Value

A `System.Collections.Generic.KeyValuePair<A, B>` with *Key* having the value of *arg.ItemA* and *Value* having the value of *arg.ItemB*.

Description

This operator, only available on `System.Tuple<A, B>` and not on other members of the tuple family, converts a value of type `System.Tuple<A, B>` to a value of type `System.Collections.Generic.KeyValuePair<A, B>`.

9.9 Tuple<A, ...>.op_Inequality(Tuple<A, ...>, Tuple<A, ...>) method

[ILAsm]
```
.method public hidebysig specialname static
        bool  op_Inequality(valuetype Tuple`2<!0, !1> left,
                            valuetype Tuple`2<!0, !1> right)
.method public hidebysig specialname static
        bool  op_Inequality(valuetype Tuple`3<!0, !1, !2> left,
                            valuetype Tuple`3<!0, !1, !2> right)
…

.method public hidebysig specialname static
       bool op_Inequality
       (valuetype Tuple`10<!0, !1, !2, !3, !4, !5, !6, !7, !8, !9> left,
        valuetype Tuple`10<!0, !1, !2, !3, !4, !5, !6, !7, !8, !9> right
       )
```

[C#]
```
public static bool operator!=(Tuple<A, B> left, Tuple<A, B> right)
public static bool operator!=(Tuple<A, B, C> left, Tuple<A, B, C> right)
…
public static bool operator!=(Tuple<A, B, C, D, E, F, G, H, I, J> left,
                              Tuple<A, B, C, D, E, F, G, H, I, J> right
                             )
```

Summary

Determines whether the specified values are not equal.

Parameters

Parameter	Description
left	The first `Tuple<A, …>` value to compare.
right	The second `Tuple<A, …>` value to compare.

Return Value

`false` if the tuples are equal, `true` otherwise.

Description

Two tuples are equal if every pair of corresponding fields are equal. Fields are compared using the `Equals` method of the field type, e.g. `ItemA.Equals(other.ItemA)`.

9.10 Tuple<A, …>.ToString() method

[ILAsm]
```
.method public hidebysig virtual string ToString()
```

[C#]
```
public override string ToString()
```

Summary

Returns a `System.String` representation of the value of the current instance.

Return Value

A string in standard tuple syntax "`(ItemA, ItemB, …)`". Each element of the tuple is converted using the `ToString` method of the element's type.

Description

[*Note:* This method overrides `System.Object.ToString`.]

10 System.Unit enum

[ILAsm]
```
.class public serializable sealed Unit extends System.Enum
```

[C#]
```
public enum Unit
```

Assembly Info:

- *Name:*
- *Public Key:*
- *Version:*
- *Attributes:*
    - CLSCompliantAttribute(true)

Summary

The `Unit` type is a unary enumeration type. It may be used, for example, as a generic type argument when an existing generic type is used but one of its type arguments is not required for the particular application.

Inherits From: `System.Enum`

Library:

10.1 Unit.Unit field

[ILAsm]
```
.field public static literal valuetype System.Unit Unit
```

[C#]
```
Unit
```

Summary

The only member of the enumeration.